

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

ОТЧЕТ

по дисциплине «Учебная практика»

**Тема: «изучение инструмента Windows Forms для создания приложений
на платформе .NET Framework с использованием программного
продукта Microsoft Visual Studio 2022.»**

Студент гр. 2302

Фролов А. Э.

Преподаватель:

Калмычков В. А.

Санкт-Петербург

2024

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студент: Фролов А. Э.

Группа 2302

Тема практики: изучение инструмента Windows Forms для создания приложений на платформе .NET Framework с использованием программного продукта Microsoft Visual Studio 2022.

Задание на практику: разработать два оконных приложения с использованием инструмента Windows Forms. Первое приложение должно осуществлять анимацию движения объекта по траектории. Второе приложение должно демонстрировать процесс построения фрактала и дерева, которое его хранит. Оба приложения должны иметь различные настройки их функционала.

Сроки прохождения практики: 03.07.2024 – 16.07.2024

Дата сдачи отчета: 16.07.2024

Дата защиты отчета: 16.07.2024

АННОТАЦИЯ

Отчет содержит информацию о работе, сделанной в рамках прохождения учебной практики. Все разделы структурированы по их смысловому содержанию. Для удобства понимания в процессе описания приведены дополняющие изображения.

Работа была выполнена на языке C# с использованием инструмента WindowsForms внутри программного продукта Microsoft Visual Studio 2022.

SUMMARY

The report contains information about the work done as part of the educational practice. All sections are structured according to their semantic content. For convenience of understanding, additional images are provided in the description process.

The work was done in C# language using the Windows Forms tool inside the Microsoft Visual Studio 2022 software product.

Содержание

ВВЕДЕНИЕ.....	5
1. Ознакомительные задания.....	6
1.1. Задание №1.	6
1.2. Задание №2.	6
1.3. Задание №3.	7
1.4. Задание №4.	9
1.5. Задание №5.	10
1.6. Задание №6.	12
1.7. Задание №7. DynGO.	13
2. Первое задание – Анимация движения объекта.	15
2.1. Формулировка задания.	15
2.2. Математическая постановка.....	15
2.3. Описание используемых элементов интерфейса.....	15
2.4. Описание настроек элементов интерфейса.....	16
2.5. Описание используемых графических примитивов.	16
2.6. Текст программы.	17
2.7. Примеры работы программы.	23
3. Задание №2 – Построение фрактала и фрактального дерева.	26
3.1. Формулировка задания.	26
3.2. Математическая постановка.....	26
3.3. Описание используемых элементов интерфейса.....	27
3.4. Описание настроек элементов интерфейса.....	27
3.5. Описание используемых графических примитивов.	27
3.6. Текст программы.	28
3.7. Примеры работы программы.	38
ЗАКЛЮЧЕНИЕ	39
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	40

ВВЕДЕНИЕ

Отчет к работе поделен на три раздела. Первый раздел содержит результат выполнения ознакомительных заданий. Второй раздел содержит результат выполнения первого практического задания. Третий раздел содержит результат выполнения второго практического задания.

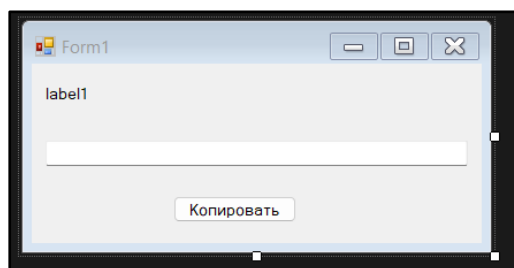
В конце приведены выводы о проделанной работе и полученных в результате навыках.

1. Ознакомительные задания.

1.1. Задание №1.

Познакомимся с элементами «TextBox», «Label» и «Button».

Добавим на форму элемент «Button» с текстом «Копировать», текстовое поле «TextBox» и элемент «label»:



Реализуем копирование текста из «TextBox» в элемент «Label». Добавим следующий код:

```
Ссылка 1
private void Form1_Load(object sender, EventArgs e)
{
    label1.Text = "";
}

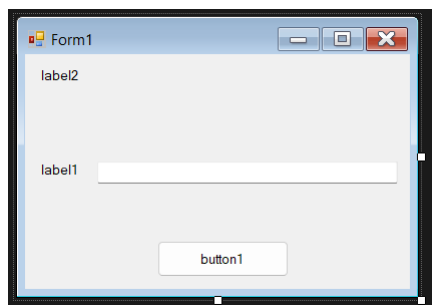
Ссылка 1
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = textBox1.Text;
}
```

Все готово.

1.2. Задание №2.

Познакомимся с элементами «ToolTip» и «MessageBox».

Первым делом добавим на форму элемент «Button» с текстом «button1», текстовое поле «TextBox» и два элемента «Label»:



Добавим на форму элемент «ToolTip». Код для инициализации приведен ниже:

```

Ссылка: 1
private void Form1_Load(object sender, EventArgs e)
{
    this.Text = "Форма приветствия";
    label1.Text = "Name: ";
    label2.Text = "Напишите ваше имя.";
    button1.Text = "Ввод";

    tooltip1.SetToolTip(textBox1, "Введите\ваше имя");
    tooltip1.IsBalloon = true;
}

```

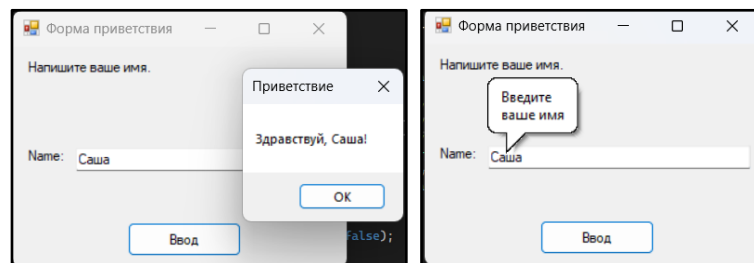
Для реализации всплывающего окна добавим следующий код:

```

Ссылка: 1
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Здравствуй, " + textBox1.Text + "!", "Приветствие");
}

```

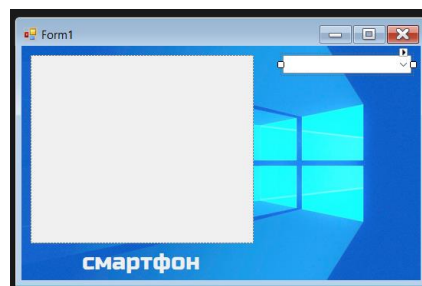
В результате получаем следующий функционал (всплывающее окно при нажатии на кнопку и поле с подсказкой при наведении на текстовую форму):



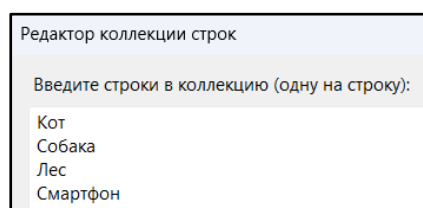
1.3. Задание №3.

Познакомимся с элементом «PictureBox» и «ComboBox».

В первой части задания нужно реализовать галерею. Для этого добавим на форму элементы «ComboBox», «label» и «PictureBox». Организуем из следующим образом:



В элемент «ComboBox» добавим список:



Код для инициализации:

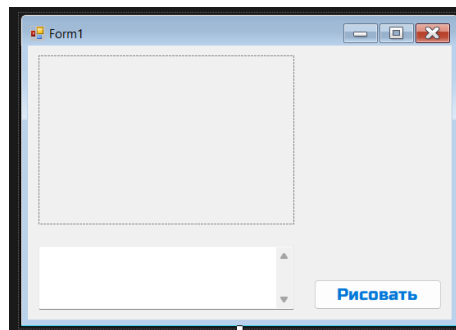
```
Ссылка 1
private void Form1_Load(object sender, EventArgs e)
{
    this.Text = "Фото галерея";
    label1.Text = "";
    comboBox1.Text = "Список";
}
```

Код для переключения изображений:

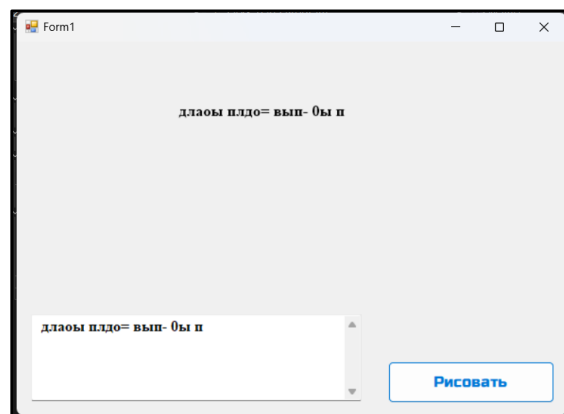
```
Ссылка 1
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    switch (comboBox1.SelectedIndex) {
        case 0:
            pictureBox1.Image = Image.FromFile("C:\\Users\\DNS\\Documents\\Alexander_Frol...");
            label1.Text = "Кот";
            break;
        case 1:
            pictureBox1.Image = Image.FromFile("C:\\Users\\DNS\\Documents\\Alexander_Frol...");
            label1.Text = "Собака";
            break;
        case 2:
            pictureBox1.Image = Image.FromFile("C:\\Users\\DNS\\Documents\\Alexander_Frol...");
            label1.Text = "Лес";
            break;
        case 3:
            pictureBox1.Image = Image.FromFile("C:\\Users\\DNS\\Documents\\Alexander_Frol...");
            label1.Text = "Смартфон";
            break;
        default:
            break;
    }
}
```

Все готово.

Во второй части задания нужно реализовать рисование текста в поле «PictureBox». Форма имеет следующий вид:



Результат:

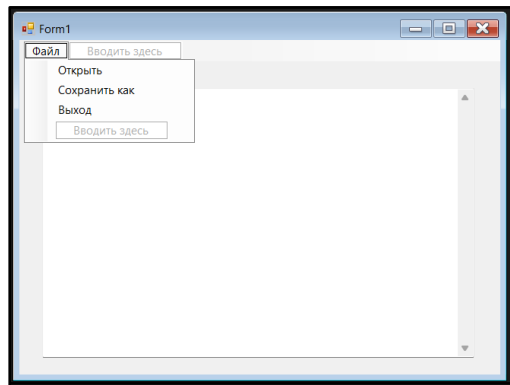


1.4. Задание №4.

Познакомимся с элементом «ToolStripMenuItem».

Добавим на форму элементы «PictureBox» и «ToolStripMenuItem».

Организуем из следующим образом:



Код для инициализации окна:

```
Ссылка: 1
private void Form1_Load(object sender, EventArgs e)
{
    this.Text = "Текстовый редактор";
    openFileDialog1.FileName = "D:\\ВУЗ\\Text2.txt";
    openFileDialog1.Filter = "Текстовые файлы (*.txt)|*.txt|All files (*.*)|*.*";
    saveFileDialog1.Filter = "Текстовые файлы (*.txt)|*.txt|All files (*.*)|*.*";
}
```

Код для работы кнопки «Открыть»:

```
Ссылка: 1
private void открытьToolStripMenuItem_Click(object sender, EventArgs e)
{
    openFileDialog1.ShowDialog();
    if (openFileDialog1.FileName == null) return;
    try
    {
        StreamReader MyReader = new System.IO.StreamReader(openFileDialog1.FileName,
            System.Text.Encoding.GetEncoding(1251));
        textBox1.Text = MyReader.ReadToEnd();
        MyReader.Close();
    }
    catch (System.IO.FileNotFoundException ex)
    {
        MessageBox.Show(ex.Message + "\nФайл не найден", "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
}
```

Код для работы кнопки «Сохранить как»:

```

Ссылка 1
private void сохранитьКакToolStripMenuItem_Click(object sender, EventArgs e)
{
    saveFileDialog1.FileName = openFileDialog1.FileName;
    if (saveFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK) Save();
}

Ссылка 2
void Save()
{
    try
    {
        StreamWriter MyWriter = new StreamWriter(
            saveFileDialog1.FileName, false,
            System.Text.Encoding.GetEncoding(1251));
        MyWriter.Write(textBox1.Text);
        MyWriter.Close();
        textBox1.Modified = false;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
}

Ссылка 1
private void выходToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.Close();
}

```

Код для работы кнопки «Выход»:

```

Ссылка 0
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    if (textBox1.Modified == false) return;

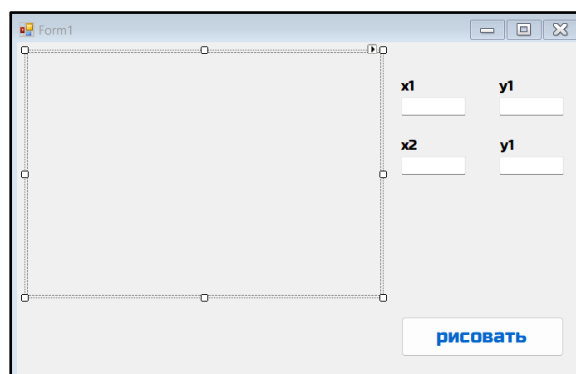
    var MeBox = MessageBox.Show(
        "Текст был изменён. \nСохранить изменения?", "Простой редактор",
        MessageBoxButtons.YesNoCancel, MessageBoxIcon.Exclamation);
    if (MeBox == System.Windows.Forms.DialogResult.No) return;
    if (MeBox == System.Windows.Forms.DialogResult.Cancel) e.Cancel = true;
    if (MeBox == System.Windows.Forms.DialogResult.Yes)
    {
        if (saveFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK)
        {
            Save(); return;
        }
        else e.Cancel = true;
    }
}

```

1.5. Задание №5.

Познакомимся с рисованием геометрических фигур в элементе «PictureBox».

Сконструируем следующую форму:



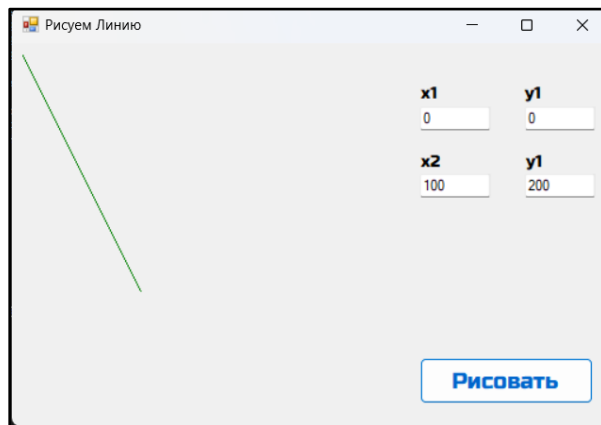
Рисование линии сделаем с помощью следующей функции:

```

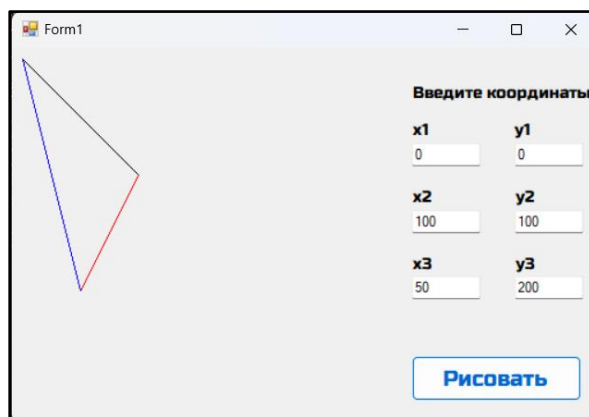
Ссылка: 1
private void pictureBox1_Paint(object sender, PaintEventArgs e)
{
    e.Graphics.DrawLine(System.Drawing.Pens.Green, m_p[1], m_p[2], m_p[3], m_p[4]);
}

```

Добавим функционал кнопке и полям. В результате получим следующий функционал:



Модифицируем предыдущую форму, чтобы можно было создать треугольник:



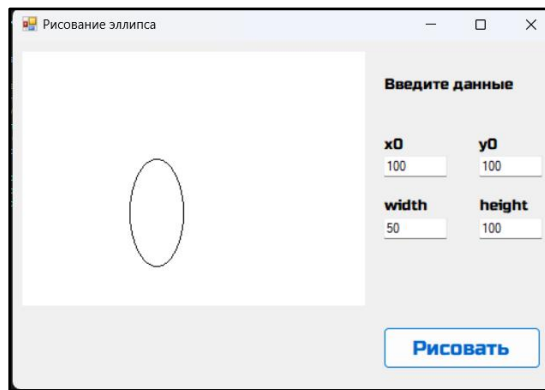
Теперь изменим форму так, чтобы можно было рисовать эллипс. Добавим следующую функцию:

```

Ссылка: 1
private void pictureBox1_Paint(object sender, PaintEventArgs e)
{
    if (index == 1)
    {
        Pen перо = new Pen(Color.Black);
        e.Graphics.DrawEllipse(перо, m_p[1], m_p[2], m_p[3], m_p[4]);
    }
}

```

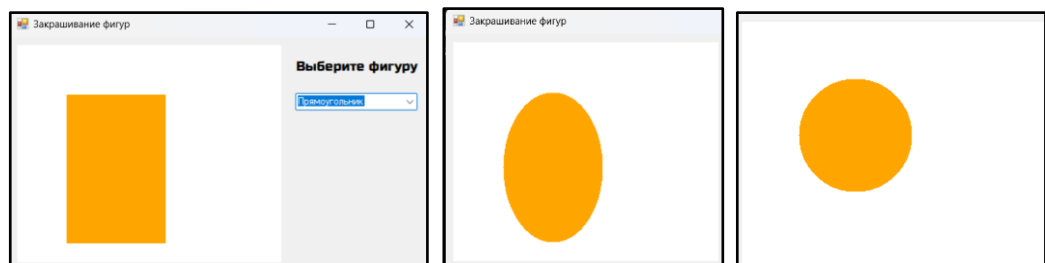
В результате получим:



В заключение сделаем возможность выбирать несколько разных фигур, при том с заливкой внутри них. Добавим функции FillEllipse():

```
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    Graphics Гرافика = pictureBox1.CreateGraphics();
    Brush Заливка = new SolidBrush(Color.Orange);
    Гرافика.Clear(Color.White);
    switch (comboBox1.SelectedIndex)
    {
        case 0:
            Гرافика.FillRectangle(Заливка, 60, 60, 120, 180); break;
        case 1:
            Гرافика.FillEllipse(Заливка, 60, 60, 120, 180); break;
        case 2:
            Гرافика.FillEllipse(Заливка, 60, 60, 120, 120); break;
    }
}
```

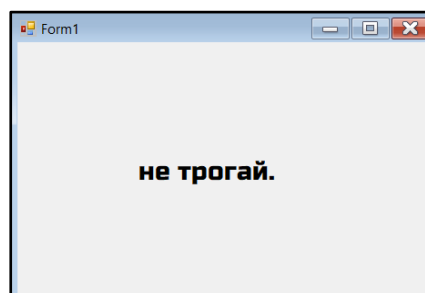
Результат:



1.6. Задание №6.

Познакомимся с программной стилизацией текста и реализуем всплывающее окно с ошибкой.

Сконструируем форму:

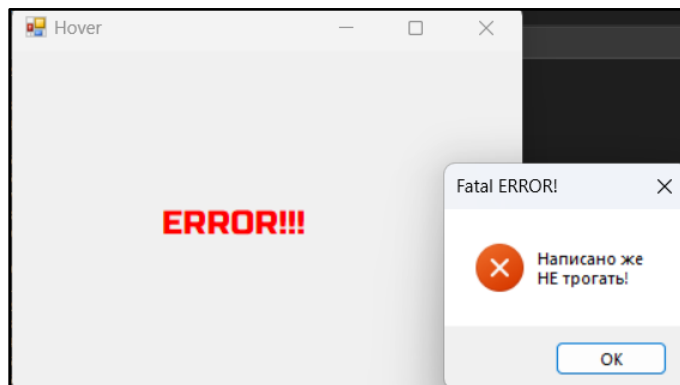


Добавим следующий код:

```
Ссылка 1
private void Form1_Load(object sender, EventArgs e)
{
    this.Text = "Hover";
    label1.TextAlign = ContentAlignment.MiddleCenter;
    label1.Text = "Не трогай.";
}

Ссылка 1
private void label1_MouseHover(object sender, EventArgs e)
{
    label1.TextAlign = ContentAlignment.MiddleCenter;
    label1.Text = "ERROR!!!";
    label1.ForeColor = Color.Red;
    MessageBox.Show("Написано же\nНЕ трогать!", "Fatal ERROR!",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

Результат:

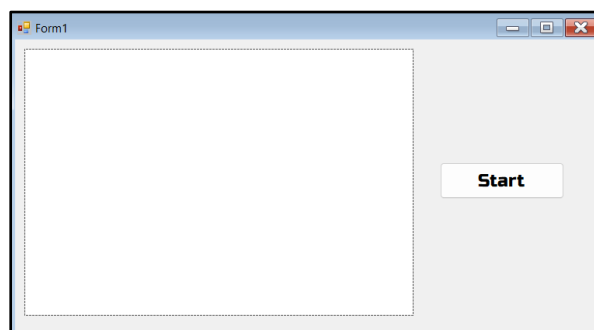


1.7. Задание №7. DynGO.

Требуется реализовать анимацию движения объекта внутри окружности по траектории, задаваемой следующим уравнением:

$$\begin{cases} x = r(k-1) \left(\cos t + \frac{\cos((k-1)t)}{k-1} \right) \\ y = r(k-1) \left(\sin t - \frac{\sin((k-1)t)}{k-1} \right) \end{cases}$$

Сконструируем форму:



Для реализации анимации будем использовать графику. Воспользуемся функциями инициализации графики `CreateGraphics()` и очистки графики `Clear()`. Код анимации приведен ниже:

```

Ссылка 1
private void Paint_Circle(int cX, int cY, int centX, int centY, int radius, int x, int y)
{
    Graphics Грaфiкa = pictureBox1.CreateGraphics();
    Грaфiкa.DrawEllipse(Pens.Black, centX + cX - radius, cY - radius - centY, radius * 2, radius * 2);
    Грaфiкa.DrawLine(Pens.Black, centX + cX, cY - centY, cX + x, cY + y);
}

Ссылка 1
private void Paint_Graphic(int cX, int cY, int r2, int x, int y, Point[] p)
{
    Graphics Грaфiкa = pictureBox1.CreateGraphics();
    Грaфiкa.Clear(Color.White);
    Paint_Circle(cX, cY, 0, 0, r2, x, y);
    Грaфiкa.DrawLines(Pens.Red, p); // траектория
}

```

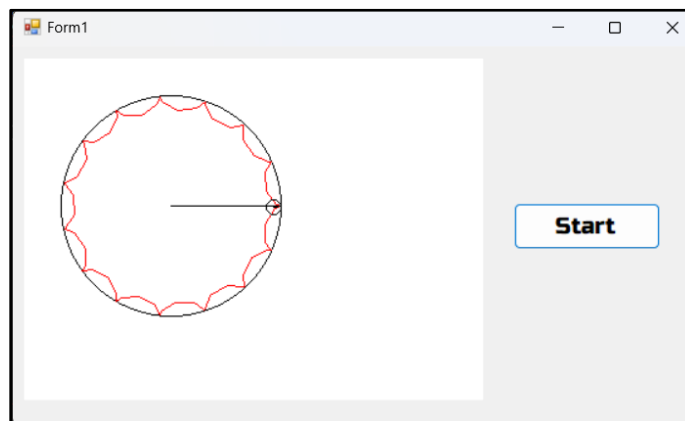
```

Ссылка 1
private void button1_Click(object sender, EventArgs e)
{
    double InitT = 0, LastT = 6.3; // оборот в 360 градусов (6,28 радиан)
    double Step = 0.1, angle = InitT;
    double x, y, x1, y1;
    int cX = 120, cY = 120; // центр большой окружности
    int R2 = 90; // радиус большой окружности
    int k = 15; // число областей на траектории
    int i = 0; // количество точек прорисовки
    int R1 = Convert.ToInt32(R2 / k); // радиус меньшей (движущейся) окружности
    Point[] p = new Point[64]; // точки для прорисовки (LastT/Step)
    while (angle <= LastT)
    {
        x = R1 * (k - 1) * (Math.Cos(angle) + Math.Cos((k - 1) * angle) / (k - 1));
        y = R1 * (k - 1) * (Math.Sin(angle) - Math.Sin((k - 1) * angle) / (k - 1));
        p[i] = new Point(cX + Convert.ToInt32(x), cY + Convert.ToInt32(y)); // расчет очередной точки траектории
        Paint_Graphic(cX, cY, R2, Convert.ToInt32(x), Convert.ToInt32(y), p);
        x1 = (R2 - R1) * Math.Sin(angle + 1.57);
        y1 = (R2 - R1) * Math.Cos(angle + 1.57);
        Paint_Circle(cX, cY, Convert.ToInt32(x1), Convert.ToInt32(y1), R1, Convert.ToInt32(x), Convert.ToInt32(y));
        angle += Step;

        Thread.Sleep(40); // время приостановки прорисовки
        ++i;
    }
}

```

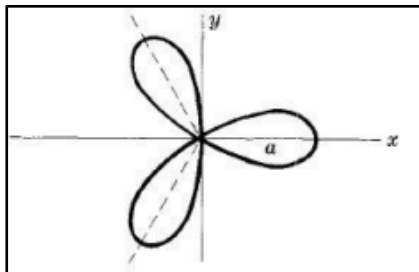
Результат после конца анимации:



2. Первое задание – Анимация движения объекта.

2.1. Формулировка задания.

Реализовать анимацию движения окружности радиуса $\frac{\alpha}{2}$ по траектории графика функции трехлепестковой розы:



У окружности должна быть реализована изменяемая 50% -я заливка и цвет контура. Более того, нужно добавить множество изменяемых параметров (тип линии, цвета заливки, цвета контура, пульсация радиуса, скорость перемещения, направление движения).

2.2. Математическая постановка.

Дано уравнение трехлепестковой розы в полярных координатах:

$$r = \alpha \sin(3\theta)$$

Требуется вычислить N точек для отображения траектории движения объекта по следующим соотношениям:

$$\begin{cases} x = r \cos(\theta), \\ y = r \sin(\theta). \end{cases}$$

Дополнительно реализовать пульсацию радиуса движимой окружности. Для этого будем использовать формулу зависимости радиуса от угла:

$$r(\varphi) = r_1 + (r_2 - r_1) * |\sin(\varphi)|$$

, где r_1 и r_2 – левая и правая границы изменяемого радиуса.

Таким образом, период пульсации составляет 180 градусов.

2.3. Описание используемых элементов интерфейса.

Элемент UI	Описание
PictureBox	Отображает математические функции

Button	Представляет интерактивную кнопку
MenuStrip	Настраиваемое меню сверху и сбоку формы
ToolStripMenuItem	Элементы меню предыдущего объекта
Label	Элемент с текстом. Используется для стилизации
TableLayoutPanel	Контейнер для элементов, хранящий их в невидимой таблице
TextBox	Элемент для ввода текста
GroupBox	Контейнер для переключателей типа «radio»
RadioButton	Переключатель типа «radio»
ColorDialog	Элемент для возможности пользовательского выбора цвета

2.4. Описание настроек элементов интерфейса.

Ниже приведено описание использованных свойств.

Свойство	Описание
BackColor	Задний фон элемента
Font	Задаёт стиль, размер и цвет шрифта
Text	Текст внутри элемента
Anchor	Определяет стороны, к которым привязан объект при масштабировании
AutoSize	автоматическое изменение размера
Dock	Определяет, как выравнивается элемент относительно родительского объекта
Location	Координаты элемента
Margin	Отступы снаружи элемента
Padding	Отступы внутри элемента
Size	Размер элемента
FlowDirection	Определяет, в каком направлении располагаются элементы (используется для элемента «MenuStrip»)
AutoScroll	Автоматическое появление полос прокрутки
(name)	Имя (текстовый индекс) элемента

2.5. Описание используемых графических примитивов.

Элемент	Использование
Линия	Составные части траектории

Эллипс	Объект, перемещающийся по траектории
--------	--------------------------------------

2.6. Текст программы.

ProjectSettings.cs
<pre> using System; using System.Collections.Generic; using System.Drawing; using System.Linq; using System.Text; using System.Threading.Tasks; namespace Task1_trajectory_movement { public class ProjectSettings { public enum MoveDirection { FORWARD, BACK } public Color BackgroundColor { get; set; } public Pen TrajectoryPen { get; set; } public Pen ObjectPen { get; set; } public Brush ObjectBrush { get; set; } public float[] ObjectPulsationValues { get; set; } public int ObjectPulsationsNumber { get; set; } public MoveDirection ObjectMoveDirection { get; set; } public float ObjectMoveSpeed { get; set; } public ProjectSettings() { TrajectoryPen = new Pen(Color.Red); ObjectPen = new Pen(Color.Black); ObjectBrush = new System.Drawing.SolidBrush(Color.Transparent); BackgroundColor = Color.White; ObjectPulsationValues = new float[2] { 1.0f, 1.0f }; ObjectPulsationsNumber = 5; ObjectMoveDirection = MoveDirection.FORWARD; ObjectMoveSpeed = 1.0f; } public ProjectSettings(ProjectSettings other) { TrajectoryPen = other.TrajectoryPen; ObjectPen = other.ObjectPen; ObjectBrush = other.ObjectBrush; BackgroundColor = other.BackgroundColor; ObjectPulsationValues = other.ObjectPulsationValues; ObjectPulsationsNumber = other.ObjectPulsationsNumber; ObjectMoveDirection = other.ObjectMoveDirection; ObjectMoveSpeed = other.ObjectMoveSpeed; } } } </pre>

Form1.cs
<pre> using System; using System.Collections.Generic; using System.ComponentModel; using System.Data; using System.Drawing; using System.Linq; using System.Reflection; using System.Text; using System.Threading; using System.Threading.Tasks; using System.Windows.Forms; namespace Task1_trajectory_movement { public partial class Form1 : Form { </pre>

```

public static ProjectSettings mainProjectSettings;

public Form1()
{
    InitializeComponent();
}

private void Form1_Load(object sender, EventArgs e)
{
    this.Text = "Движение объекта";

    mainProjectSettings = new ProjectSettings(); // set default project settings
}

// caculates "r" from input equation
private double Calculate_r(double alpha, double phi)
{
    return alpha * Math.Sin(3 * phi);
}

// caculates radius of the object based on pulsation bounds and pulsation power
private float Calculate_pulsatingRadius(float angle, float leftBound, float rightBound)
{
    return (float)(leftBound + (rightBound - leftBound) * Math.Abs(Math.Sin(angle)));
}

private Point[] CalculateTrajectoryPoints(int N, int alpha)
{
    double step = (1 * Math.PI / (N - 1));
    var variable = Pens.Black;

    // calculate degrees in radians and radiuses (polar coordinate system)
    double[] degrees = new double[N];
    double[] radiuses = new double[N];
    degrees[0] = 0.0;
    radiuses[0] = Calculate_r(alpha, 0);
    for (int i = 1; i < N; ++i)
    {
        degrees[i] = degrees[i - 1] + step;
        radiuses[i] = Calculate_r(alpha, degrees[i]);
    }

    // calculate result points (cartesian coordinate system)
    Point[] points = new Point[N];
    for (int i = 0; i < N; ++i)
    {
        points[i] = new Point(
            Convert.ToInt32(radiuses[i] * Math.Cos(degrees[i])),
            Convert.ToInt32(radiuses[i] * Math.Sin(degrees[i]))
        );
    }

    // convert points to be on the center of the picture
    int window_width = pictureBox1.Width;
    int window_height = pictureBox1.Height;
    for (int i = 0; i < N; ++i)
    {
        points[i] = new Point(
            window_width / 2 + points[i].X,
            window_height / 2 - points[i].Y;
        )
    }

    return points;
}

// Paints graphics updating it every frame
private void Update_Graphic(float ellipse_radius, Point[] points)
{
    // initialize graphic
    Graphics Графика = pictureBox1.CreateGraphics();
    Графика.Clear(mainProjectSettings.BackgroundColor);

    // coordinates of ellipse center
    int x0 = points[points.Length - 1].X;
    int y0 = points[points.Length - 1].Y;

    // add new piece (new frame) trajectory
    Графика.DrawLines(mainProjectSettings.TrajectoryPen, points);
}

```

```

        // print moving Ellipse
        // TODO: find out the idea of why the ellipse radius is divided by 2
        Графика.FillEllipse(mainProjectSettings.ObjectBrush, x0 - ellipse_radius / 2, y0 -
ellipse_radius / 2, ellipse_radius, ellipse_radius);
        Графика.DrawEllipse(mainProjectSettings.ObjectPen, x0 - ellipse_radius / 2, y0 -
ellipse_radius / 2, ellipse_radius, ellipse_radius);

    }

    private void button1_Click(object sender, EventArgs e)
    {
        int N = 100; // number of points in trajectory (also number of frames in animation)
        int alpha = 200; // hyper parameter in equation
        int time_delay = 15; // delay between frames (in ms)

        Point[] points = CalculateTrajectoryPoints(N, alpha); // points for trajectory
drawing
        if (mainProjectSettings.ObjectMoveDirection == ProjectSettings.MoveDirection.BACK)
        {
            points = points.Reverse().ToArray();
        }

        float pulsatingRadius;
        float normal_raduis = alpha / 2;
        float pulsating_angle_step = (float)(Math.PI *
mainProjectSettings.ObjectPulsationsNumber / (N - 1));

        // first frame (i == 0)
        Update_Graphic(0, new Point[] { points[0], points[1] });
        Thread.Sleep(time_delay);

        // start animation
        for (int i = 1; i < N; ++i)
        {
            pulsatingRadius = Calculate_pulsatingRadius(
                pulsating_angle_step * i,
                normal_raduis * mainProjectSettings.ObjectPulsationValues[0],
                normal_raduis * mainProjectSettings.ObjectPulsationValues[1]);

            Update_Graphic(pulsatingRadius, points.Take(i + 1).ToArray());
            Thread.Sleep(Convert.ToInt32(time_delay /
mainProjectSettings.ObjectMoveSpeed));
        }
    }

    private void параметрыToolStripMenuItem_Click(object sender, EventArgs e)
    {
        SettingsForm settingsForm = new SettingsForm();
        settingsForm.ShowDialog();
    }

    private void выходToolStripMenuItem_Click(object sender, EventArgs e)
    {
        this.Close();
    }
}
}

```

SettingsForm.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Windows.Forms.VisualStyles;
using static System.Windows.Forms.VisualStyles.VisualStyleElement.Button;

namespace Task1_trajectory_movement
{

```

```

public partial class SettingsForm : Form
{
    private ProjectSettings tempProjectSettings;

    // Checks if the values of settings are correct
    private bool UpdateSettings()
    {
        try
        {
            tempProjectSettings.ObjectPen.Width =
Convert.ToInt32(textBox_объект_ТолщинаКонтура.Text);
            tempProjectSettings.ObjectPulsationValues[0] =
Convert.ToInt32(textBox_объект_ПульсацияРадиусаОт.Text) / 100.0f;
            tempProjectSettings.ObjectPulsationValues[1] =
Convert.ToInt32(textBox_объект_ПульсацияРадиусаДо.Text) / 100.0f;
            if (tempProjectSettings.ObjectPulsationValues[0] >
tempProjectSettings.ObjectPulsationValues[1])
            {
                return false;
            }
            tempProjectSettings.ObjectPulsationsNumber =
Convert.ToInt32(textBox_объект_ЧислоПульсаций.Text);
            tempProjectSettings.ObjectMoveSpeed =
Convert.ToInt32(textBox_объект_Скорость.Text) / 100.0f;

            tempProjectSettings.TrajectoryPen.Width =
Convert.ToInt32(textBox_траектория_ТолщинаКонтура.Text);
        }
        catch (Exception)
        {
            return false;
        }

        return true;
    }

    private void SetLineTypeChecked(Pen pen, System.Windows.Forms.GroupBox lineTypeBox)
    {
        foreach (Control control in lineTypeBox.Controls)
        {
            if (control is System.Windows.Forms.RadioButton radioButton)
            {
                if ((radioButton.Text == "Сплошной" && pen.DashStyle ==
System.Drawing.Drawing2D.DashStyle.Solid) ||
                    (radioButton.Text == "Пунктирный" && pen.DashStyle ==
System.Drawing.Drawing2D.DashStyle.Dash))
                {
                    radioButton.Checked = true;
                }
                else
                {
                    radioButton.Checked = false;
                }
            }
        }
    }

    private void SetMoveDirectionChecked(ProjectSettings.MoveDirection direction,
System.Windows.Forms.GroupBox lineTypeBox)
    {
        foreach (Control control in lineTypeBox.Controls)
        {
            if (control is System.Windows.Forms.RadioButton radioButton)
            {
                if ((radioButton.Text == "Стандартное" && direction ==
ProjectSettings.MoveDirection.FORWARD) ||
                    (radioButton.Text == "Обратное" && direction ==
ProjectSettings.MoveDirection.BACK))
                {
                    radioButton.Checked = true;
                }
                else
                {
                    radioButton.Checked = false;
                }
            }
        }
    }
}

```

```

// =====
// ===== ОСНОВНЫЕ ЭЛЕМЕНТЫ ===== //
// =====

public SettingsForm()
{
    InitializeComponent();
}

private void SettingsForm_Load(object sender, EventArgs e)
{
    // set default checkboxes and text values for object
    tempProjectSettings = new ProjectSettings(Form1.mainProjectSettings);
    textBox_объект_ТолщинаКонтура.Text =
tempProjectSettings.ObjectPen.Width.ToString();
    SetLineTypeChecked(tempProjectSettings.ObjectPen, groupBox_объект_ТипКонтура);
    textBox_объект_ПульсацияРадиусаОт.Text =
(tempProjectSettings.ObjectPulsationValues[0] * 100).ToString();
    textBox_объект_ПульсацияРадиусаДо.Text =
(tempProjectSettings.ObjectPulsationValues[1] * 100).ToString();
    textBox_объект_ЧислоПульсаций.Text =
tempProjectSettings.ObjectPulsationsNumber.ToString();
    SetMoveDirectionChecked(tempProjectSettings.ObjectMoveDirection,
groupBox_объект_Направление);
    textBox_объект_ЧислоПульсаций.Text =
tempProjectSettings.ObjectPulsationsNumber.ToString();
    textBox_объект_Скорость.Text = (tempProjectSettings.ObjectMoveSpeed *
100).ToString();

    // set default checkboxes and text values for trajectory
    textBox_траектория_ТолщинаКонтура.Text =
tempProjectSettings.TrajectoryPen.Width.ToString();
    SetLineTypeChecked(tempProjectSettings.TrajectoryPen,
groupBox_траектория_ТипЛинии);

    // default settings
    this.Text = "Параметры";
    flowLayoutPanel_объект.Visible = false;
    flowLayoutPanel_траектория.Visible = false;
    button_Применить.Enabled = false;
}

private void toolStripMenuItem1_Click(object sender, EventArgs e)
{
    flowLayoutPanel_объект.Visible = true;
    flowLayoutPanel_траектория.Visible = false;
}

private void ToolStripMenuItem_траектория_Click(object sender, EventArgs e)
{
    flowLayoutPanel_траектория.Visible = true;
    flowLayoutPanel_объект.Visible = false;
}

private void button_Применить_Click(object sender, EventArgs e)
{
    if (UpdateSettings())
    {
        var result = MessageBox.Show("Сохранить изменения?", "Подтвердите изменения",
MessageBoxButtons.OKCancel);

        if (result == System.Windows.Forms.DialogResult.OK)
        {
            Form1.mainProjectSettings = new ProjectSettings(tempProjectSettings);
            button_Применить.Enabled = false;
        }
        else
        {
            MessageBox.Show("Есть некорректные значения", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        }
    }
}

private void button_Выход_Click(object sender, EventArgs e)
{
    if (button_Применить.Enabled)
    {

```

```

        var result = MessageBox.Show("У вас есть несохраненные изменения.\nВы точно  
хотите выйти?",
        "Подтвердите выход", MessageBoxButtons.OKCancel);
        if (result == DialogResult.OK)
        {
            this.Close();
        }
    }
    else
    {
        this.Close();
    }
}

// =====
// ===== ОБЪЕКТ =====
// =====

private void button_объект_ВыбратьЦветЗаливки_Click(object sender, EventArgs e)
{
    colorDialog1 = new ColorDialog();
    colorDialog1.Color = ((SolidBrush)tempProjectSettings.ObjectBrush).Color;
    if (colorDialog1.ShowDialog() == DialogResult.OK)
    {
        tempProjectSettings.ObjectBrush = new System.Drawing.SolidBrush(Color.FromArgb(
            128, colorDialog1.Color.R, colorDialog1.Color.G, colorDialog1.Color.B));
        button_Применить.Enabled = true;
    }
}

private void button_объект_ВыбратьЦветКонтура_Click(object sender, EventArgs e)
{
    colorDialog1 = new ColorDialog();
    colorDialog1.Color = tempProjectSettings.ObjectPen.Color;
    if (colorDialog1.ShowDialog() == DialogResult.OK)
    {
        tempProjectSettings.ObjectPen = new Pen(colorDialog1.Color,
Form1.mainProjectSettings.ObjectPen.Width);
        button_Применить.Enabled = true;
    }
}

private void textBox_объект_ТолщинаКонтура_TextChanged(object sender, EventArgs e)
{
    button_Применить.Enabled = true;
}

private void radioButton_объект_ТипЛинииСплошная_CheckedChanged(object sender,
EventArgs e)
{
    tempProjectSettings.ObjectPen.DashStyle = System.Drawing.Drawing2D.DashStyle.Solid;
    button_Применить.Enabled = true;
}

private void radioButton_объект_ТипЛинииПунктирная_CheckedChanged(object sender,
EventArgs e)
{
    tempProjectSettings.ObjectPen.DashStyle = System.Drawing.Drawing2D.DashStyle.Dash;
    button_Применить.Enabled = true;
}

private void textBox_объект_ПульсацияРадиусаОт_TextChanged(object sender, EventArgs e)
{
    button_Применить.Enabled = true;
}

private void textBox_объект_ПульсацияРадиусаДо_TextChanged(object sender, EventArgs e)
{
    button_Применить.Enabled = true;
}

private void textBox_объект_ЧислоПульсаций_TextChanged(object sender, EventArgs e)
{
    button_Применить.Enabled = true;
}

private void textBox_объект_Скорость_TextChanged(object sender, EventArgs e)

```

```

        {
            button_Применить.Enabled = true;
        }

        private void radioButton_объект_НаправлениеСтандартное_CheckedChanged(object sender,
EventArgs e)
        {
            tempProjectSettings.ObjectMoveDirection = ProjectSettings.MoveDirection.FORWARD;
            button_Применить.Enabled = true;
        }

        private void radioButton_объект_НаправлениеОбратное_CheckedChanged(object sender,
EventArgs e)
        {
            tempProjectSettings.ObjectMoveDirection = ProjectSettings.MoveDirection.BACK;
            button_Применить.Enabled = true;
        }

        // =====
        // ===== ТРАЕКТОРИЯ ===== //
        // =====

        private void button_траектория_ВыбратьЦвет_Click(object sender, EventArgs e)
        {
            colorDialog1 = new ColorDialog();
            if (colorDialog1.ShowDialog() == DialogResult.OK)
            {
                tempProjectSettings.TrajectoryPen = new Pen(colorDialog1.Color,
Form1.mainProjectSettings.TrajectoryPen.Width);
                button_Применить.Enabled = true;
            }
        }

        private void textBox_траектория_ТолщинаКонттура_TextChanged(object sender, EventArgs e)
        {
            button_Применить.Enabled = true;
        }

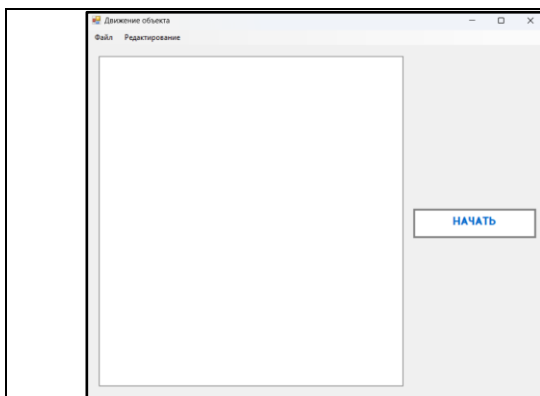
        private void radioButton_траектория_ТипЛинииСплошной_CheckedChanged(object sender,
EventArgs e)
        {
            tempProjectSettings.TrajectoryPen.DashStyle =
System.Drawing.Drawing2D.DashStyle.Solid;
            button_Применить.Enabled = true;
        }

        private void radioButton_траектория_ТипЛинииПунктирный_CheckedChanged(object sender,
EventArgs e)
        {
            tempProjectSettings.TrajectoryPen.DashStyle =
System.Drawing.Drawing2D.DashStyle.Dash;
            button_Применить.Enabled = true;
        }
    }
}

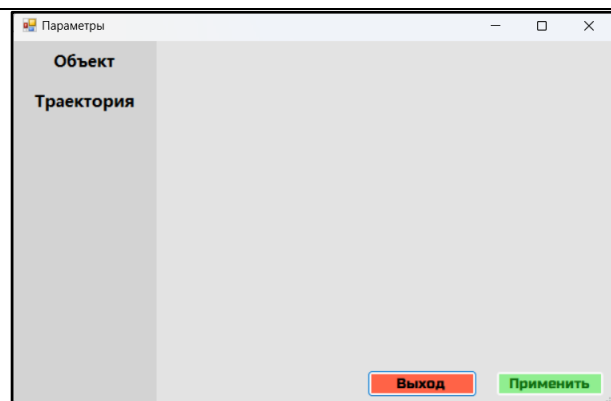
```

2.7. Примеры работы программы.

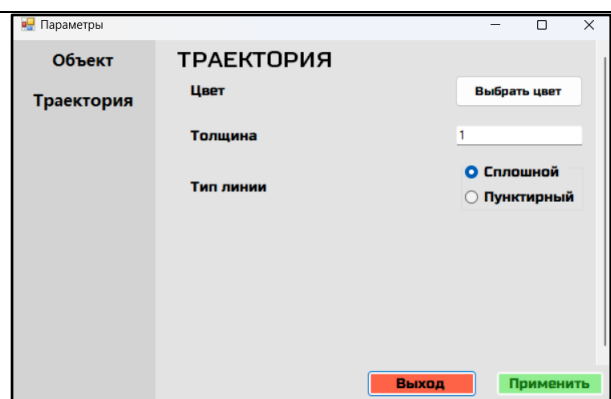
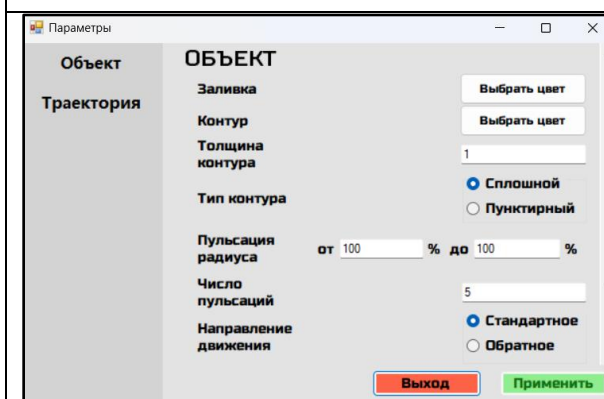
Внешний вид программы	Панель настроек
-----------------------	-----------------



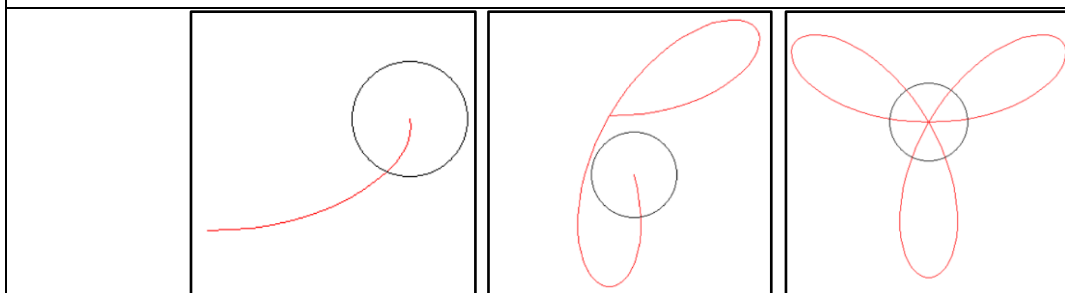
Панель настроек объекта



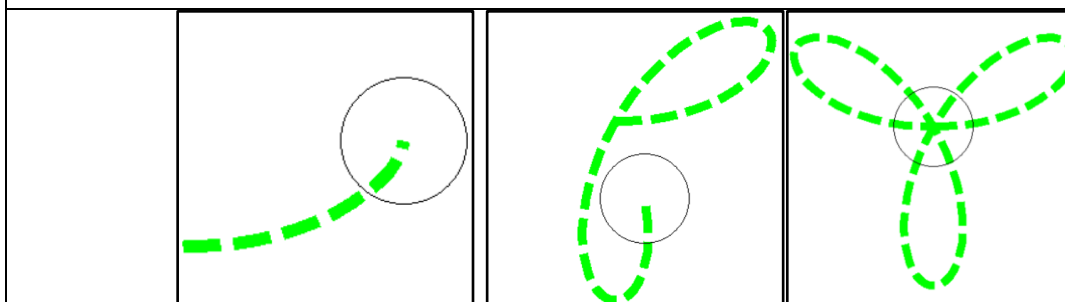
Панель настроек траектории



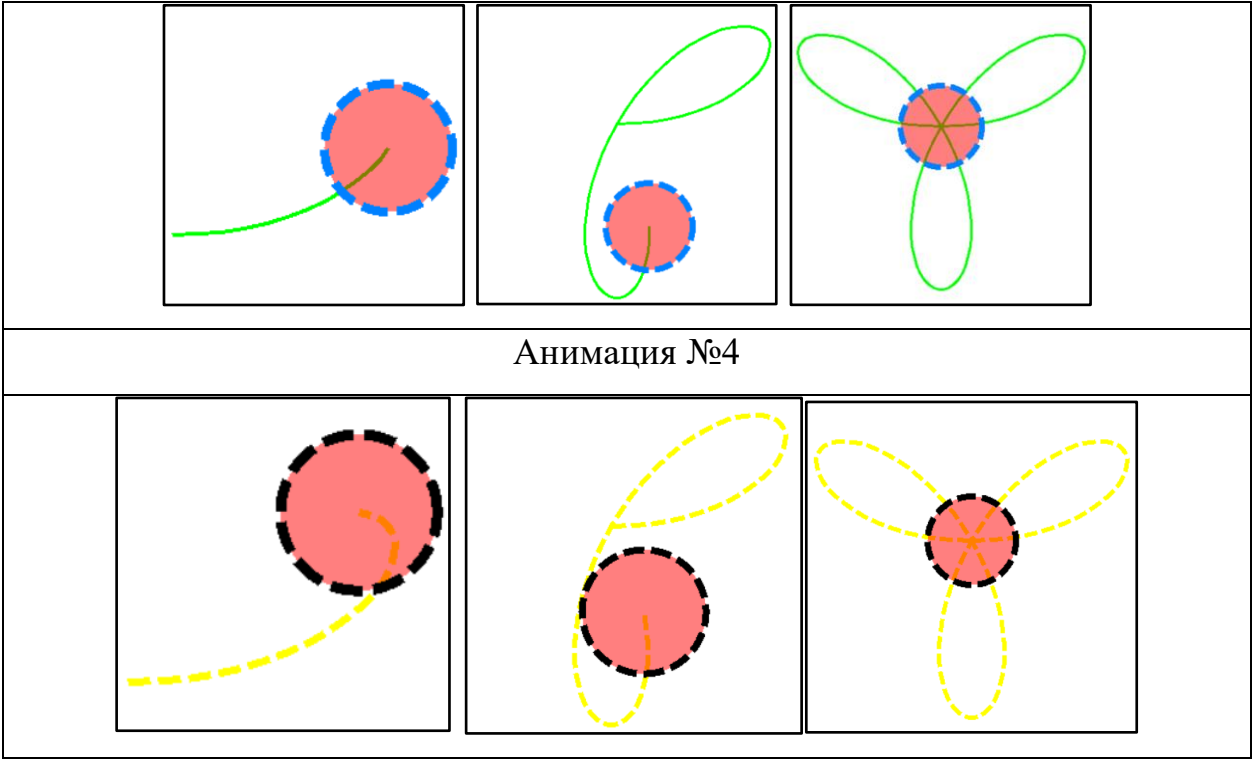
Анимация №1



Анимация №2



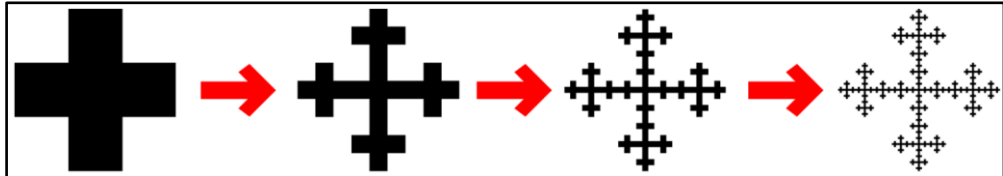
Анимация №3



3. Задание №2 – Построение фрактала и фрактального дерева.

3.1. Формулировка задания.

Реализовать хранение построение фрактала «Крест». Каждый крест составлен из пяти одинаковых квадратов. Пример деления такого фрактала приведен ниже:



Пользователь должен иметь возможность выбирать число шагов для построения фрактала. На экран при этом должно выводиться как полученное на указанном шаге дерево, так и пошаговый процесс построения фрактала.

Каждый уровень дерева должен иметь уникальный цвет, соотносящийся с цветом фрактала на каждом новом шаге.

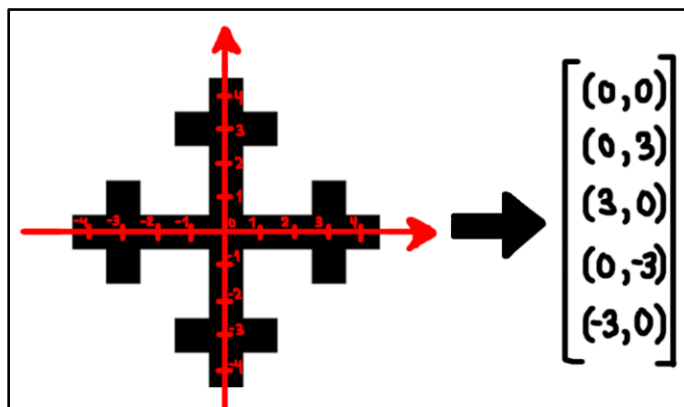
3.2. Математическая постановка.

Реализовать генерацию N точек для построения фрактала на i -м шаге. Число N при этом вычисляется следующим образом:

$$N = 5^i$$

, где $i = 0, 1, 2, \dots$

Каждая точка представляет собой центр крестов, образующих фрактал. Сторона квадрата при этом внутри дерева считается равной 1 (сторона может быть умножена на соответствующий коэффициент при выведении фрактала на экран). Пример центров для фрактала на 1-м шаге:



3.3. Описание используемых элементов интерфейса.

Элемент UI	Описание
PictureBox	Отображает математические функции
Button	Представляет интерактивную кнопку
MenuStrip	Настраиваемое меню сверху и сбоку формы
ToolStripMenuItem	Элементы меню предыдущего объекта
Label	Элемент с текстом. Используется для стилизации
TableLayoutPanel	Контейнер для элементов, хранящий их в невидимой таблице
NumericalUpDown	Элемент для ввода чисел
FlowLayoutPanel	Элемент для последовательного хранения других элементов интерфейса

3.4. Описание настроек элементов интерфейса.

Ниже приведено описание использованных свойств.

Свойство	Описание
BackColor	Задний фон элемента
Font	Задаёт стиль, размер и цвет шрифта
Text	Текст внутри элемента
Anchor	Определяет стороны, к которым привязан объект при масштабировании
AutoSize	автоматическое изменение размера
Location	Координаты элемента
Margin	Отступы снаружи элемента
Padding	Отступы внутри элемента
Size	Размер элемента
FlowDirection	Определяет, в каком направлении располагаются элементы (используется для элемента «MenuStrip»)
AutoScroll	Автоматическое появление полос прокрутки
(name)	Имя (текстовый индекс) элемента

3.5. Описание используемых графических примитивов.

Элемент	Использование
---------	---------------

Линия	Соединение узлов дерева, прорисовка граничных участков изображения
Окружность	Узлы дерева
Прямоугольник	Составная часть крестов, образующих фрактал

3.6. Текст программы.

ProjectSettings.cs
<pre> using System; using System.Collections.Generic; using System.Diagnostics; using System.Drawing; using System.Linq; using System.Net.Http.Headers; using System.Text; using System.Threading.Tasks; using System.Windows.Forms; namespace Task2_FractalBuilding { public class ProjectSettings { public int StepsNumber { get; set; } public List<Color> StepsColors { get; set; } public float[] PictureSizeCoeffs { get; set; } public float[] TreeBoundsCoeffs { get; set; } public float[] GridBoundsCoeffs { get; set; } public int GridRowsNumber { get; set; } public int GridColsNumber { get; set; } public ProjectSettings() { StepsNumber = 4; // 0 step + 4 dividing steps (5 steps total) StepsColors = new List<Color>() { Color.Black, Color.Red, Color.Orange, Color.Green, Color.Blue, Color.Purple, Color.Yellow, Color.Red, Color.Orange, Color.Green }; PictureSizeCoeffs = new float[2] { 1.0f, 1.0f }; TreeBoundsCoeffs = new float[4] { 0.0f, 1.0f, 0.5f, 0.0f }; GridBoundsCoeffs = new float[4] { 0.5f, 1.0f, 1.0f, 0.0f }; GridRowsNumber = 1; GridColsNumber = 5; } public ProjectSettings(ProjectSettings other) { StepsNumber = other.StepsNumber; StepsColors = other.StepsColors; PictureSizeCoeffs = other.PictureSizeCoeffs; TreeBoundsCoeffs = other.TreeBoundsCoeffs; GridBoundsCoeffs = other.GridBoundsCoeffs; GridRowsNumber = other.GridRowsNumber; GridColsNumber = other.GridColsNumber; } } } </pre>

PictureGrid.cs
<pre> using System; using System.Collections.Generic; using System.Drawing; using System.Linq; using System.Text; using System.Threading.Tasks; using System.Windows.Forms; </pre>

```

namespace Task2_FractalBuilding
{
    public class PictureGrid
    {
        public int Nrows { get; private set; }
        public int Ncols { get; private set; }
        private int[] bounds { get; set; }
        public Point[,] Centers { get; private set; }
        public int[, ,] Bounds { get; private set; }
        private Pen pen { get; set; }
        private System.Drawing.Drawing2D.DashStyle dashstyle { get; set; }

        public PictureGrid(int nrows, int ncols, int[] bounds, Color color,
            System.Drawing.Drawing2D.DashStyle dashstyle)
        {
            this.Nrows = nrows;
            this.Ncols = ncols;
            this.pen = new Pen(color);
            pen.DashStyle = dashstyle;
            this.dashstyle = dashstyle;
            this.bounds = bounds;

            Centers = new Point[nrows, ncols];
            Bounds = new int[nrows, ncols, 4];
        }

        public void Initialize()
        {
            // [0] - top bound
            // [1] - right bound
            // [2] - down bound
            // [3] - left bound

            int colsStep = (bounds[1] - bounds[3]) / Ncols;
            int rowsStep = (bounds[2] - bounds[0]) / Nrows;

            for (int i = 0; i < Nrows; ++i)
            {
                for (int j = 0; j < Ncols; ++j)
                {
                    Bounds[i, j, 0] = bounds[0] + rowsStep * i;
                    Bounds[i, j, 1] = bounds[3] + colsStep * (j + 1);
                    Bounds[i, j, 2] = bounds[0] + rowsStep * (i + 1);
                    Bounds[i, j, 3] = bounds[3] + colsStep * j;

                    Centers[i, j] = new Point(
                        (Bounds[i, j, 1] + Bounds[i, j, 3]) / 2,
                        (Bounds[i, j, 0] + Bounds[i, j, 2]) / 2);
                }
            }
        }

        public void Print(Graphics g)
        {
            int horizontalStep = (bounds[1] - bounds[3]) / Ncols;
            int verticalStep = (bounds[2] - bounds[0]) / Nrows;

            for (int i = 0; i <= Ncols; ++i)
            {
                g.DrawLine(pen,
                    new Point(bounds[3] + horizontalStep * i, bounds[0]),
                    new Point(bounds[3] + horizontalStep * i, bounds[2]));
            }

            for (int i = 0; i <= Nrows; ++i)
            {
                g.DrawLine(pen,
                    new Point(bounds[3], bounds[0] + verticalStep * i),
                    new Point(bounds[1], bounds[0] + verticalStep * i));
            }
        }
    }
}

```

FractalTree.cs

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.Linq;
using System.Reflection.Emit;
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Xml.Linq;

namespace Task2_FractalBuilding
{
    public class FractalNode
    {
        public Point Center { get; set; }
        public List<FractalNode> Children { get; set; }

        public FractalNode(Point Center)
        {
            this.Center = Center;
            Children = new List<FractalNode>();
        }

        public FractalNode(Point Center, List<FractalNode> Children)
        {
            this.Center = Center;
            this.Children = Children;
        }

        public FractalNode(FractalNode other)
        {
            this.Center = other.Center;
            this.Children = other.Children;
        }
    }

    // Representation of a fractal "Cross" where each cross contains 5 squares
    public class FractalTree
    {
        private readonly FractalNode root;
        public int Height { get; private set; }

        public FractalTree(int depth)
        {
            Height = depth;
            root = CreateFractalNode(new FractalNode(new Point(0, 0)), 0);
        }

        // Level of the tree starts from 0
        public int NumberOfNodesOnLevel(int level)
        {
            return (int)Math.Pow(5, level);
        }

        // Returns diameter of fractal (number of squares in diameter).
        // (level starts from 0).
        public int DiameterOnLevel(int level)
        {
            int diameter = 3;
            for (int i = 0; i < level; ++i)
            {
                diameter *= 3;
            }
            return diameter;
        }

        private FractalNode CreateFractalNode(FractalNode node, int currentLevel)
        {
            if (currentLevel == Height)
            {
                return node;
            }
        }
    }
}
```

```

        int childrenOffset = (int)Math.Pow(3, currentLevel + 1);

        node.Children.Add(
            CreateFractalNode(
                new FractalNode(new Point(node.Center.X, node.Center.Y)), currentLevel + 1
            ));
        node.Children.Add(
            CreateFractalNode(
                new FractalNode(new Point(node.Center.X, node.Center.Y + childrenOffset)),
currentLevel + 1
            ));
        node.Children.Add(
            CreateFractalNode(
                new FractalNode(new Point(node.Center.X + childrenOffset, node.Center.Y)),
currentLevel + 1
            ));
        node.Children.Add(
            CreateFractalNode(
                new FractalNode(new Point(node.Center.X, node.Center.Y - childrenOffset)),
currentLevel + 1
            ));
        node.Children.Add(
            CreateFractalNode(
                new FractalNode(new Point(node.Center.X - childrenOffset, node.Center.Y)),
currentLevel + 1
            ));

        return node;
    }

    public void Print(Graphics g, int[] pictuteBounds)
    {
        // pictureBounds
        // [0] - top bound
        // [1] - right bound
        // [2] - down bound
        // [3] - left bound

        // assume we print tree in a table where
        // every node takes his own cell
        float cellHeight = (pictuteBounds[2] - pictuteBounds[0]) / (float)(Height + 1);
        float cellWidth;

        // offset for drawing circles within a cells
        float verticalOffset = 0.05f;

        float minCircleRadius = cellHeight * 0.05f;

        List<Point> prevLevelNodesCenters = new List<Point>();
        float prevLevelCicrcleRadius = 0;

        for (int i = 0; i < (Height + 1); ++i)
        {
            int nodesNumber = NumberOfNodesOnLevel(i);
            cellWidth = (pictuteBounds[1] - pictuteBounds[3]) / (float)(nodesNumber);
            float circleRadius = (
                Math.Min(cellWidth, cellHeight) -
                2 * verticalOffset * Math.Min(cellWidth, cellHeight)
            ) / 2;
            circleRadius = Math.Max(circleRadius, minCircleRadius);

            // Drawing
            for (int j = 0; j < nodesNumber; ++j)
            {
                Point circleCenter = new Point(
                    pictuteBounds[3] + (int)(cellWidth * j + cellWidth / 2),
                    pictuteBounds[0] + (int)(cellHeight * i + cellHeight / 2));

                // also if i > 0
                if (prevLevelNodesCenters.Count > 0)
                {
                    g.DrawLine(Pens.Black,
                        new Point(circleCenter.X, (int)(circleCenter.Y - circleRadius)),
                        new Point(prevLevelNodesCenters.First().X,
(int) (prevLevelNodesCenters.First().Y + prevLevelCicrcleRadius))
                    );
                    if ((j + 1) % 5 == 0)
                    {
                        prevLevelNodesCenters.RemoveAt(0);

```

```

        }
    }

    g.FillEllipse(new
SolidBrush(MainForm.mainProjectSettings.StepsColors.ElementAt(i)),
        circleCenter.X - circleRadius,
        circleCenter.Y - circleRadius,
        circleRadius * 2, circleRadius * 2);

    prevLevelNodesCenters.Add(circleCenter);
}

prevLevelCircleRadius = circleRadius;
}
}

public List<FractalNode> BreadthFirstSearch()
{
    List<FractalNode> visited = new List<FractalNode>();
    Queue<FractalNode> queue = new Queue<FractalNode>();

    if (root != null)
    {
        queue.Enqueue(root);
    }

    while (queue.Count > 0)
    {
        FractalNode current = queue.Dequeue();
        visited.Add(current);

        foreach (FractalNode child in current.Children)
        {
            queue.Enqueue(child);
        }
    }

    return visited;
}
}
}

```

ProjectSettingsForm.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Task2_FractalBuilding
{
    public partial class ProjectSettingsForm : Form
    {
        private ProjectSettings tempProjectSettings;

        private void SettingsChanged(object sender, EventArgs e)
        {
            button_ПараметрыПрименить.Enabled = true;
        }

        private void UpdateFields(ProjectSettings settings)
        {
            numericUpDown_ЧислоШагов.Value = tempProjectSettings.StepsNumber + 1;

            numericUpDown_ШиринаИзображения.Value =
Convert.ToInt32(tempProjectSettings.PictureSizeCoeffs[0] * 100);
            numericUpDown_ВысотаИзображения.Value =
Convert.ToInt32(tempProjectSettings.PictureSizeCoeffs[1] * 100);

            numericUpDown_ДеревоВерхняяГраница.Value =
Convert.ToInt32(tempProjectSettings.TreeBoundsCoeffs[0] * 100);

```



```

        numericUpDown_ДеревоНижняяГраница.Value =
Convert.ToInt32(tempProjectSettings.TreeBoundsCoeffs[2] * 100);
        numericUpDown_ДеревоЛеваяГраница.Value =
Convert.ToInt32(tempProjectSettings.TreeBoundsCoeffs[3] * 100);
        numericUpDown_ДеревоПраваяГраница.Value =
Convert.ToInt32(tempProjectSettings.TreeBoundsCoeffs[1] * 100);

        numericUpDown_СеткаВерхняяГраница.Value =
Convert.ToInt32(tempProjectSettings.GridBoundsCoeffs[0] * 100);
        numericUpDown_СеткаНижняяГраница.Value =
Convert.ToInt32(tempProjectSettings.GridBoundsCoeffs[2] * 100);
        numericUpDown_СеткаЛеваяГраница.Value =
Convert.ToInt32(tempProjectSettings.GridBoundsCoeffs[3] * 100);
        numericUpDown_СеткаПраваяГраница.Value =
Convert.ToInt32(tempProjectSettings.GridBoundsCoeffs[1] * 100);

        numericUpDown_СеткаЧислоСтрок.Value =
Convert.ToInt32(tempProjectSettings.GridRowsNumber);
        numericUpDown_СеткаЧислоСтолбцов.Value =
Convert.ToInt32(tempProjectSettings.GridColsNumber);
    }

    private bool CorrectSettings(ProjectSettings settings)
    {
        if (settings.GridRowsNumber * settings.GridColsNumber < (settings.StepsNumber + 1))
        {
            return false;
        }
        return true;
    }

    public ProjectSettingsForm()
    {
        InitializeComponent();

        foreach (Control control in this.Controls)
        {
            if (control is FlowLayoutPanel)
            {
                foreach (Control control2 in control.Controls)
                {
                    if (control2 is TableLayoutPanel)
                    {
                        foreach (Control control3 in control2.Controls)
                        {
                            if (control3 is NumericUpDown)
                            {
                                ((NumericUpDown)control3).ValueChanged += SettingsChanged;
                            }
                        }
                    }
                }
            }
        }

        private void ProjectSettingsForm_Load(object sender, EventArgs e)
        {
            tempProjectSettings = new ProjectSettings(MainForm.mainProjectSettings);
            UpdateFields(tempProjectSettings);

            button_ПараметрыПрименить.Enabled = false;
        }

        private void button_ПараметрыПрименить_Click(object sender, EventArgs e)
        {
            if (button_ПараметрыПрименить.Enabled)
            {
                if (CorrectSettings(tempProjectSettings))
                {
                    var result = MessageBox.Show("Сохранить изменения?", "Подтвердите
изменения", MessageBoxButtons.OKCancel);

                    if (result == DialogResult.OK)
                    {
                        MainForm.mainProjectSettings = new
ProjectSettings(tempProjectSettings);
                        button_ПараметрыПрименить.Enabled = false;
                    }
                }
            }
        }
    }

```

```

        }
        else
        {
            MessageBox.Show("Число ячеек в сетке не должно быть меньше числа шагов",
"Ошибка", MessageBoxButtons.OK);
        }
    }

    private void button_ПараметрыВыход_Click(object sender, EventArgs e)
    {
        if (button_ПараметрыПрименить.Enabled)
        {
            var result = MessageBox.Show("Сохранить измененбья?", "У вас есть несохраненные
изменения", MessageBoxButtons.OKCancel);

            if (result == DialogResult.OK)
            {
                MainForm.mainProjectSettings = new ProjectSettings(tempProjectSettings);
            }
            this.Close();
        }
    }

    private void numericUpDown_ЧислоШагов_ValueChanged(object sender, EventArgs e)
    {
        tempProjectSettings.StepsNumber = (int)numericUpDown_ЧислоШагов.Value - 1;
    }

    private void numericUpDown_ШиринаИзображения_ValueChanged(object sender, EventArgs e)
    {
        tempProjectSettings.PictureSizeCoeffs[0] =
(float) (numericUpDown_ШиринаИзображения.Value / 100);
    }

    private void numericUpDown_ВысотаИзображения_ValueChanged(object sender, EventArgs e)
    {
        tempProjectSettings.PictureSizeCoeffs[1] =
(float) (numericUpDown_ВысотаИзображения.Value / 100);
    }

    private void numericUpDown_ДеревоВерхняяГраница_ValueChanged(object sender, EventArgs
e)
    {
        tempProjectSettings.TreeBoundsCoeffs[0] =
(float) (numericUpDown_ДеревоВерхняяГраница.Value / 100);
    }

    private void numericUpDown_ДеревоНижняяГраница_ValueChanged(object sender, EventArgs e)
    {
        tempProjectSettings.TreeBoundsCoeffs[2] =
(float) (numericUpDown_ДеревоНижняяГраница.Value / 100);
    }

    private void numericUpDown_ДеревоЛеваяГраница_ValueChanged(object sender, EventArgs e)
    {
        tempProjectSettings.TreeBoundsCoeffs[3] =
(float) (numericUpDown_ДеревоЛеваяГраница.Value / 100);
    }

    private void numericUpDown_ДеревоПраваяГраница_ValueChanged(object sender, EventArgs e)
    {
        tempProjectSettings.TreeBoundsCoeffs[1] =
(float) (numericUpDown_ДеревоПраваяГраница.Value / 100);
    }

    private void numericUpDown_СеткаВерхняяГраница_ValueChanged(object sender, EventArgs e)
    {
        tempProjectSettings.GridBoundsCoeffs[0] =
(float) (numericUpDown_СеткаВерхняяГраница.Value / 100);
    }

    private void numericUpDown_СеткаНижняяГраница_ValueChanged(object sender, EventArgs e)
    {
        tempProjectSettings.GridBoundsCoeffs[2] =
(float) (numericUpDown_СеткаНижняяГраница.Value / 100);
    }

```

```

        private void numericUpDown_СеткаЛеваяГраница_ValueChanged(object sender, EventArgs e)
        {
            tempProjectSettings.GridBoundsCoeffs[3] =
(float) (numericUpDown_СеткаЛеваяГраница.Value / 100);
        }

        private void numericUpDown_СеткаПраваяГраница_ValueChanged(object sender, EventArgs e)
        {
            tempProjectSettings.GridBoundsCoeffs[1] =
(float) (numericUpDown_СеткаПраваяГраница.Value / 100);
        }

        private void numericUpDown_СеткаЧислоСтрок_ValueChanged(object sender, EventArgs e)
        {
            tempProjectSettings.GridRowsNumber = (int)numericUpDown_СеткаЧислоСтрок.Value;
        }

        private void numericUpDown_СеткаЧислоСтолбцов_ValueChanged(object sender, EventArgs e)
        {
            tempProjectSettings.GridColsNumber = (int)numericUpDown_СеткаЧислоСтолбцов.Value;
        }
    }
}

```

MainForm.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Linq.Expressions;
using System.Net.Security;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Task2_FractalBuilding
{
    public partial class MainForm : Form
    {
        public static ProjectSettings mainProjectSettings;

        public MainForm()
        {
            InitializeComponent();

            mainProjectSettings = new ProjectSettings();
        }

        private void UpdateSettings()
        {
            // - 0.05f due to troubles with pictureBox sizing
            mainPictureBox.Size = new Size(
                (int) (flowLayoutPanel_main.Width * (mainProjectSettings.PictureSizeCoeffs[0] -
0.05f)),
                (int) (flowLayoutPanel_main.Height * (mainProjectSettings.PictureSizeCoeffs[1] -
0.05f)));
        }

        private void DrawDividingLines(Graphics g)
        {
            g.DrawRectangle(Pens.Black,
                (int) (mainPictureBox.Width * mainProjectSettings.TreeBoundsCoeffs[3]),
                (int) (mainPictureBox.Height * mainProjectSettings.TreeBoundsCoeffs[0]),
                (int) (mainPictureBox.Width * mainProjectSettings.TreeBoundsCoeffs[1] -
mainPictureBox.Width * mainProjectSettings.TreeBoundsCoeffs[3]),
                (int) (mainPictureBox.Height * mainProjectSettings.TreeBoundsCoeffs[2] -
mainPictureBox.Height * mainProjectSettings.TreeBoundsCoeffs[0]));

            g.DrawRectangle(Pens.Black,
                (int) (mainPictureBox.Width * mainProjectSettings.GridBoundsCoeffs[3]),
                (int) (mainPictureBox.Height * mainProjectSettings.GridBoundsCoeffs[0]),

```

```

        (int)(mainPictureBox.Width * mainProjectSettings.GridBoundsCoeffs[1] -
mainPictureBox.Width * mainProjectSettings.GridBoundsCoeffs[3]),
        (int)(mainPictureBox.Height * mainProjectSettings.GridBoundsCoeffs[2] -
mainPictureBox.Height * mainProjectSettings.GridBoundsCoeffs[0]));
    }

    private void MainForm_Load(object sender, EventArgs e)
    {
        UpdateSettings();
    }

    private void ToolStripMenuItem_Параметры_Click(object sender, EventArgs e)
    {
        ProjectSettingsForm settingsForm = new ProjectSettingsForm();
        settingsForm.Show();
    }

    private void ToolStripMenuItem_Выход_Click(object sender, EventArgs e)
    {
        this.Close();
    }

    private void ToolStripMenuItem_Запуск_Click(object sender, EventArgs e)
    {
        UpdateSettings();

        Image cachedImage = new Bitmap(mainPictureBox.Width, mainPictureBox.Height);
        Graphics g = Graphics.FromImage(cachedImage);
        g.Clear(Color.White);

        DrawDividingLines(g);

        FractalTree tree = new FractalTree(mainProjectSettings.StepsNumber);
        tree.Print(g, new int[4] {
            (int)(mainPictureBox.Height * mainProjectSettings.TreeBoundsCoeffs[0]),
            (int)(mainPictureBox.Width * mainProjectSettings.TreeBoundsCoeffs[1]),
            (int)(mainPictureBox.Height * mainProjectSettings.TreeBoundsCoeffs[2]),
            (int)(mainPictureBox.Width * mainProjectSettings.TreeBoundsCoeffs[3]) });

        PictureGrid grid = new PictureGrid(
            mainProjectSettings.GridRowsNumber,
            mainProjectSettings.GridColsNumber, new int[4] {
                (int)(mainPictureBox.Height * mainProjectSettings.GridBoundsCoeffs[0]),
                (int)(mainPictureBox.Width * mainProjectSettings.GridBoundsCoeffs[1]),
                (int)(mainPictureBox.Height * mainProjectSettings.GridBoundsCoeffs[2]),
                (int)(mainPictureBox.Width * mainProjectSettings.GridBoundsCoeffs[3]) },
            Color.Black, System.Drawing.Drawing2D.DashStyle.Solid);
        grid.Initialize();
        grid.Print(g);

        PrintFractalSteps(g, tree, grid);

        mainPictureBox.Image = cachedImage;
    }

    private void PrintFractalSteps(Graphics g, FractalTree tree, PictureGrid grid)
    {
        // pictureBounds
        // [0] - top bound
        // [1] - right bound
        // [2] - down bound
        // [3] - left bound

        List<FractalNode> nodesBFS = tree.BreadthFirstSearch();
        int row = 0, col = 0;

        for (int level = 0; level <= tree.Height; ++level)
        {
            List<FractalNode> currentNodes = new List<FractalNode>();
            for (int i = 0; i < tree.NumberOfNodesOnLevel(level); ++i)
            {
                currentNodes.Add(nodesBFS.First());
                nodesBFS.RemoveAt(0);
            }

            PrintFractalLevel(g, grid, row, col, currentNodes, level,
tree.DiameterOnLevel(level));

            ++col;
        }
    }

```

```

        if (col == grid.Ncols)
        {
            ++row;
            col = 0;
        }
    }

    private void PrintFractalLevel(Graphics g, PictureGrid grid, int row, int col,
    List<FractalNode> nodes, int level, int levelDiameter)
    {
        float scaleCoeff = Math.Min(
            (grid.Bounds[row, col, 1] - grid.Bounds[row, col, 3]) / (float)(levelDiameter),
            (grid.Bounds[row, col, 2] - grid.Bounds[row, col, 0]) /
            (float)(levelDiameter));

        float squareRadius = scaleCoeff / 2;

        if (squareRadius >= 1)
        {
            while (nodes.Count > 0)
            {
                FractalNode currentNode = nodes.First();

                PrintCross(g, mainProjectSettings.StepsColors.ElementAt(level),
                new Point(
                    (int)(grid.Centers[row, col].X + currentNode.Center.X * scaleCoeff),
                    (int)(grid.Centers[row, col].Y - currentNode.Center.Y * scaleCoeff)),
                (int)(squareRadius));

                nodes.RemoveAt(0);
            }
        }
        else
        {
            while (nodes.Count > 0)
            {
                FractalNode currentNode = nodes.First();

                // minimum possible scaleCoeff is 2
                Point newPoint = new Point(
                    (int)(grid.Centers[row, col].X + currentNode.Center.X * 2),
                    (int)(grid.Centers[row, col].Y - currentNode.Center.Y * 2));

                // print if node places within a bounds
                if (newPoint.X > grid.Bounds[row, col, 3] &&
                    newPoint.X < grid.Bounds[row, col, 1] &&
                    newPoint.Y > grid.Bounds[row, col, 0] &&
                    newPoint.Y < grid.Bounds[row, col, 2])
                {
                    // minimum possible circleRadius is 1
                    PrintCross(g, mainProjectSettings.StepsColors.ElementAt(level),
                    newPoint, 1);
                }

                nodes.RemoveAt(0);
            }
        }
    }

    private void PrintCross(Graphics g, Color color, Point center, float squareRadius)
    {
        // center part
        g.FillRectangle(new SolidBrush(color),
            center.X - squareRadius, center.Y - squareRadius,
            squareRadius * 2, squareRadius * 2);

        // top part
        g.FillRectangle(new SolidBrush(color),
            center.X - squareRadius, center.Y - 3 * squareRadius,
            squareRadius * 2, squareRadius * 2);

        // right part
        g.FillRectangle(new SolidBrush(color),
            center.X + squareRadius, center.Y - squareRadius,
            squareRadius * 2, squareRadius * 2);

        // down part
        g.FillRectangle(new SolidBrush(color),

```

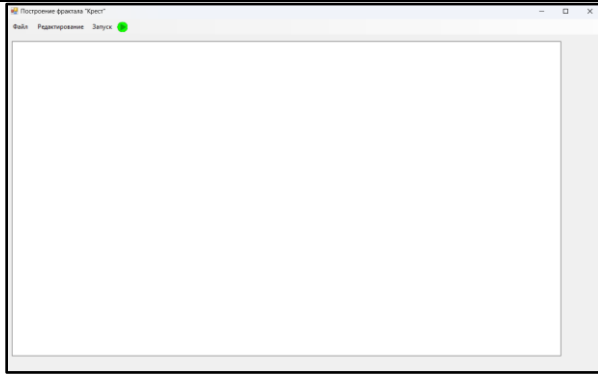
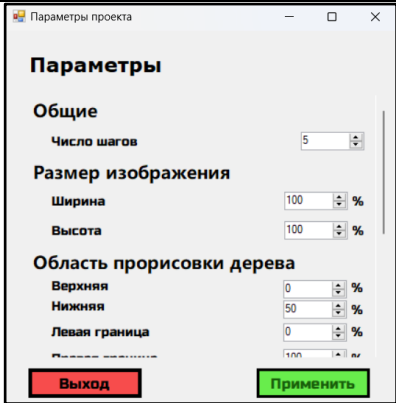
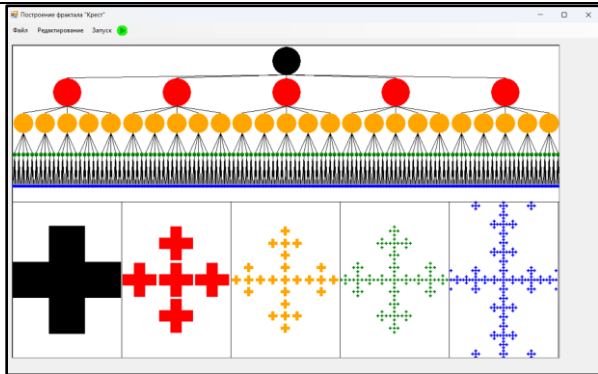
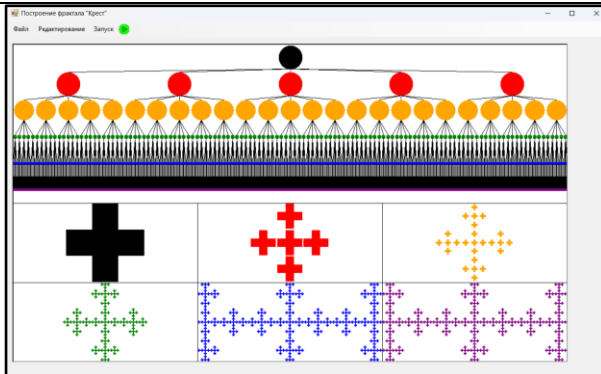
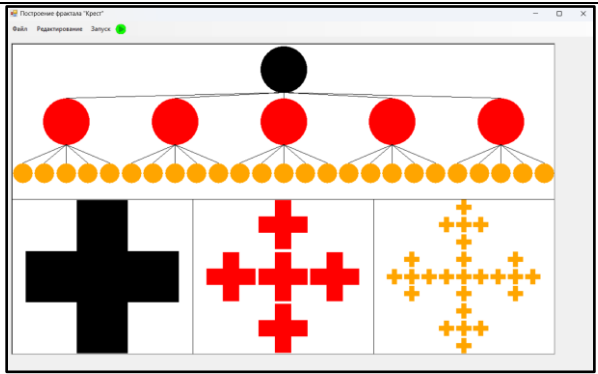
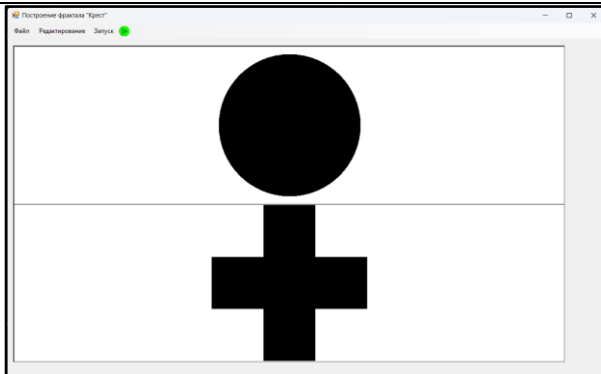
```

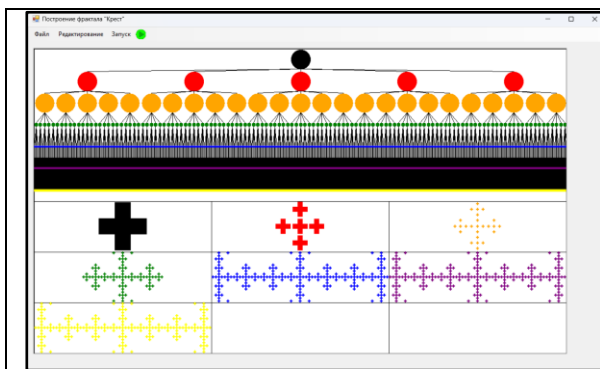
        center.X - squareRadius, center.Y + squareRadius,
        squareRadius * 2, squareRadius * 2);

// left part
g.FillRectangle(new SolidBrush(color),
    center.X - 3 * squareRadius, center.Y - squareRadius,
    squareRadius * 2, squareRadius * 2);
    }
}
}

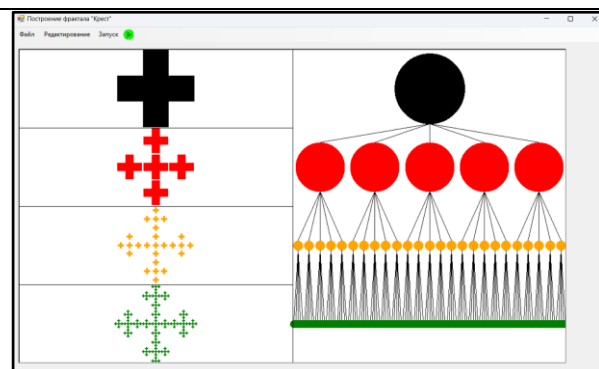
```

3.7. Примеры работы программы.

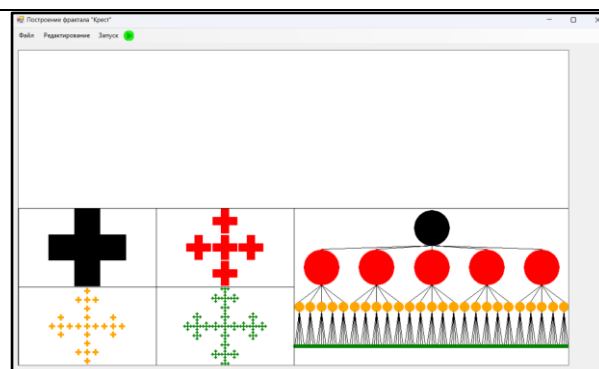
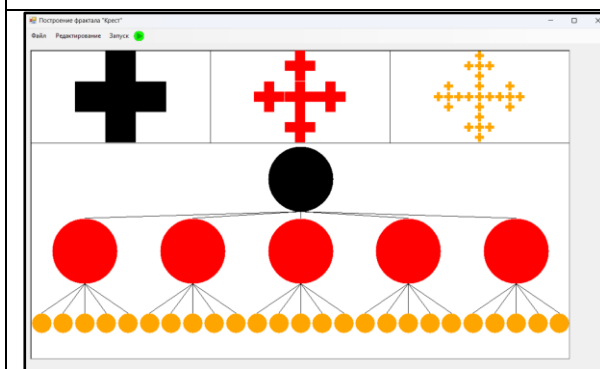
Внешний вид программы	Панель настроек
	
Запуск №1	Запуск №2
	
Запуск №3	Запуск №4
	
Запуск №5	Запуск №6



Запуск №7



Запуск №8



ЗАКЛЮЧЕНИЕ

В результате учебной практики мы познакомились с набором инструментов Windows Forms. С помощью ознакомительных заданий были изучены базовые элементы интерфейса и способы их программирования. Далее полученные знания были применены на практике.

В результате первого задания было разработано оконное приложение с анимацией движения по заданной траектории. Более того, для анимации были добавлены различные параметры, такие как цвет, тип и ширина линии, цвет заливки объекта, направление движения, сила пульсации и т.п. Как итог, были

использованы базовые элементы интерфейса и добавлен код для их функционирования.

В результате второго задания было разработано оконное приложение, которое реализует построение фрактала «Крест» с заданным числом шагов (глубиной фрактала). На изображении отображается получившееся дерево, хранящее фрактал, а также пошаговый процесс построения фрактала. Дополнительно, добавлена настройка размера изображения и положения в нем дерева и шагов построения фрактала. Как итог, были использованы базовые элементы интерфейса и добавлен код для их функционирования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Microsoft Learn – сеть разработчиков Microsoft. URL: <https://learn.microsoft.com/ru-ru/dotnet/csharp/> (дата обращения: 15.07.2024)