**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по учебной практике**

(технологическая (проектно-технологическая) практика)

|  |  |  |
| --- | --- | --- |
| Студент гр. 1302 |  | Новиков Г.В. |
| Руководитель |  | Калмычков В.А. |

Санкт-Петербург

2023

**ЗАДАНИЕ**

**на Учебную (технологическую (проектно-технологическую)) практику**

|  |  |  |
| --- | --- | --- |
| Студент Новиков Г.В. | | |
| Группа 1302 | | |
| Тема практики: учебная (технологическая (проектно-технологическая)) практика | | |
| Задание на практику:   1. Создание ознакомительных программ, содержащих winforms с различными элементами. 2. Первое индивидуальное задание. Создание приложения, в котором реализуется движение объекта по траектории. 3. Второе индивидуальное задание. Создание приложения, в котором реализуется графическое представление фрактала, где каждому уровню построения соответствует уровень дерева. | | |
| Сроки прохождения практики: 30.06.2023 – 13.07.2023 | | |
| Дата сдачи отчета: 12.07.2023 | | |
| Дата защиты отчета: 12.07.2023 | | |
|  | | |
| Студент |  | Новиков Г.В. |
| Руководитель |  | Калмычков В.А. |

**Аннотация**

Во время выполнения практики должно быть выполнено 3 задания. Первое задание общее, в нем необходимо изучить теоретический материал и выполнить описанные в нем задания для закрепления полученных знаний. Второе задание выполняется индивидуально, в нем необходимо разработать приложение, которое симулирует движение объекта по траектории. Третье задание также выполняется индивидуально, в нем необходимо разработать приложение, которое изображает фрактал и дерево, в котором фрактал должен храниться. Во всех заданиях необходимо использовать пользовательский графический интерфейс.

**Summary**

During the practice, 3 tasks must be completed. The first task is general, it is necessary to study the theoretical material and complete the tasks described in it to consolidate the knowledge gained. The second task is performed individually, in which it is necessary to develop an application that simulates the movement of an object along a trajectory. The third task is also carried out individually, in which it is necessary to develop an application that depicts a fractal and a tree in which the fractal should be stored. All jobs must use a graphical user interface.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 5 |
| 1. | Ознакомительные практические занятия | 6 |
| 1.1. | Занятие 1. Элементы button, textbox и label | 6 |
| 1.2.  1.3.  1.4.  1.5.  1.6.  1.7. | Занятие 2. Элемент MessageBox, Подсказка ToolTip  Занятие 3. Изменение шрифта текста и цвета формы и элементов  Занятие 4. Элемент MenuStrip и свойство Anchor, Открытие и запись текстового файла  Занятие 5. Рисование линий, треугольника, эллипса и окружности в PictureBox  Занятие 6. Событие MauseHover  Занятие 7. Формирование траектории для движения простого геометрического объекта | 6  7  9  9  11  12 |
| 2. | Движение геометрического объекта по траектории | 13 |
| 3. | Формирование графического изображения на основе математического описания фрактала |  |
|  | Заключение |  |
|  | Список использованных источников |  |

**введение**

Цель учебной практики состоит в изучении и освоении базовых понятий, методов и приемов использования инструментальных средств и технологий программирования при решении практических задач с выбором различных структур данных и организацией программного графического интерфейса пользователя, закрепление и приобретение новых знаний и практических навыков программирования.

Задачи учебной практики:

- формирование базовых практических понятий, лежащих в основе процесса разработки программного графического интерфейса пользователя,

- получение навыков применения средств визуализации при решении практических задач и использовании различных структур данных,

- освоение способов реализации программ на выбранном рабочем языке программирования С++ (C#) с учётом особенностей реализации в конкретной системе программирования.

**1. Ознакомительные практические занятия**

**Задание:**

Предлагается выполнить последовательность действий (файлы work++ или work#) по использованию элементов оформления пользовательского интерфейса. Каждое действие может быть оформлено отдельным приложением или все действия сосредоточены в одном приложении. Дополнительно надо реализовать пример действий по формированию движения по траектории (файл DynGO) и выявить какие характеристики отвечают за объект на траектории. Наполнение соответствующих элементов может быть произвольным.

**1.1. Занятие № 1. Элементы button, textBox и label**

Суть программы в том, что вы пишите в текстовом поле какое-то предложение, нажимаете на копку и на форме появляется надпись – тоже самое предложение.

Пример работы программы:

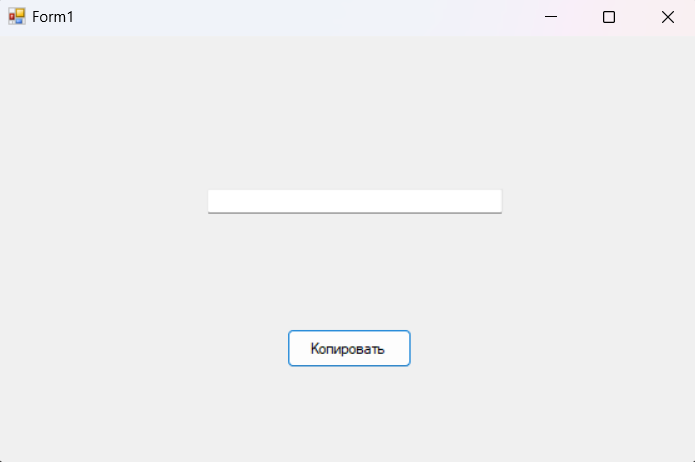
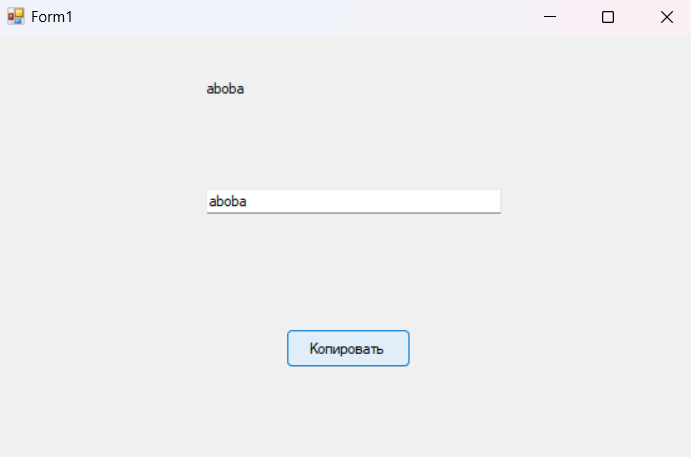
 

Рис. 1.1.1-1.1.2. Перемещение текста из TextBox в label

**1.2. Занятие № 2. Элемент MessageBox, Подсказка ToolTip**

Суть программы в том, что при нажатии на кнопку будет “выскакивать” небольшое окошко, сообщающее нам о том, какой текст был записан в текстовом поле. Для этого понадобятся всего лишь два элемента - "button" и "textBox". Подсказка "TollTip": всплывающая подсказка при наведении курсором на элемент.

Пример работы программы:

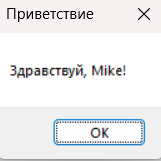
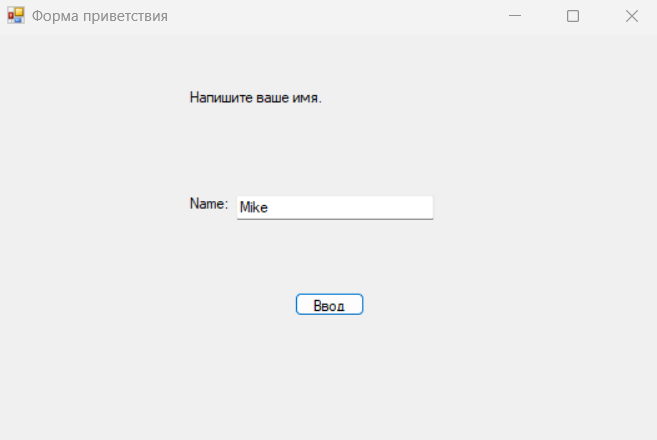


Рис. 1.2.1 и 1.2.2. Ввод имени и вывод в отдельном окне

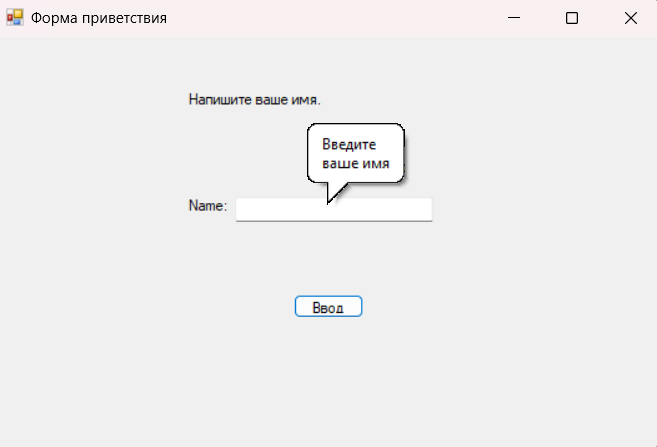


Рис. 1.2.3. Всплывающее окно при наведении

**1.3. Занятие № 3. Изменение шрифта текста и цвета формы и элементов**

**1.3.1**

Проект, в котором будем записывать на фоне изображения текст, задав изображения для заднего фона ("BackGroundImage") формы и изменив цвет кнопки. Для этого нам понадобятся "textBox", "label", кнопка "button" и изображение.

Пример работы программы:



Рис. 1.3.1. Изменение фона, цвета и шрифта

**1.3.2**

Загрузка изображения в PictureBox при помощи ComboBox.

Суть программы следующая – есть четыре картинки. Они находятся на диске или на флешке. На форме есть "comboBox", в котором находится некоторый список из четырѐх слов. При выборе одного из слов в списке должна появляться картинка, а в "label" ее название.

Пример работы программы:

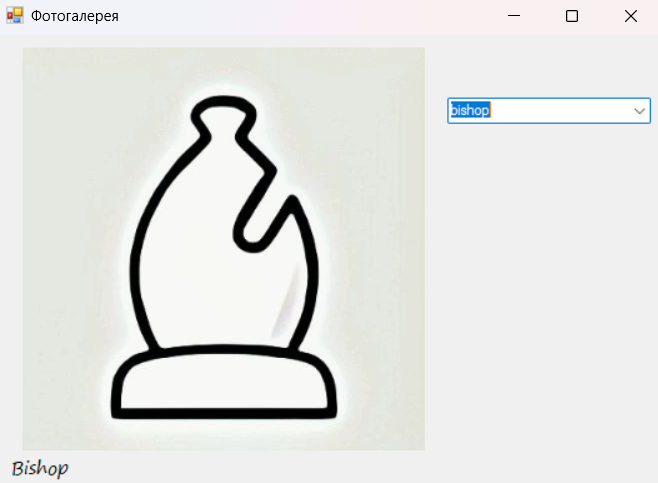
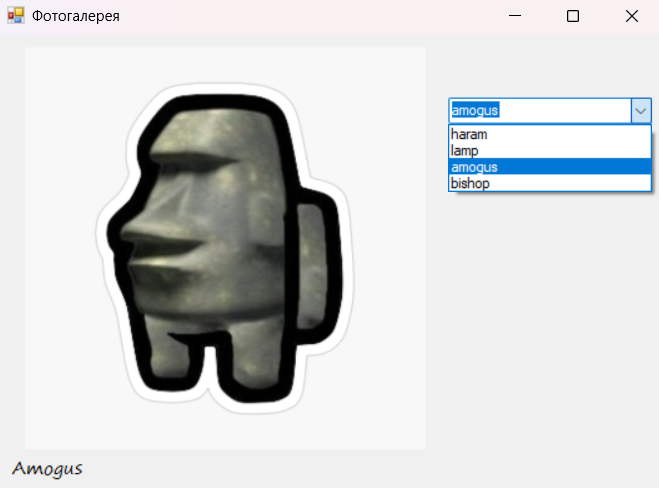


Рис. 1.3.2-1.3.3. Выбор картинки и вывод в PictureBox

**1.3.3**

Рисование текста на PictureBox.

Нужно написать в коде, чтобы текст, написанный в поле элемента "textBox" появился на поле "PictureBox" определенного размера, определенного цвета и в определенном месте.

Пример работы программы:

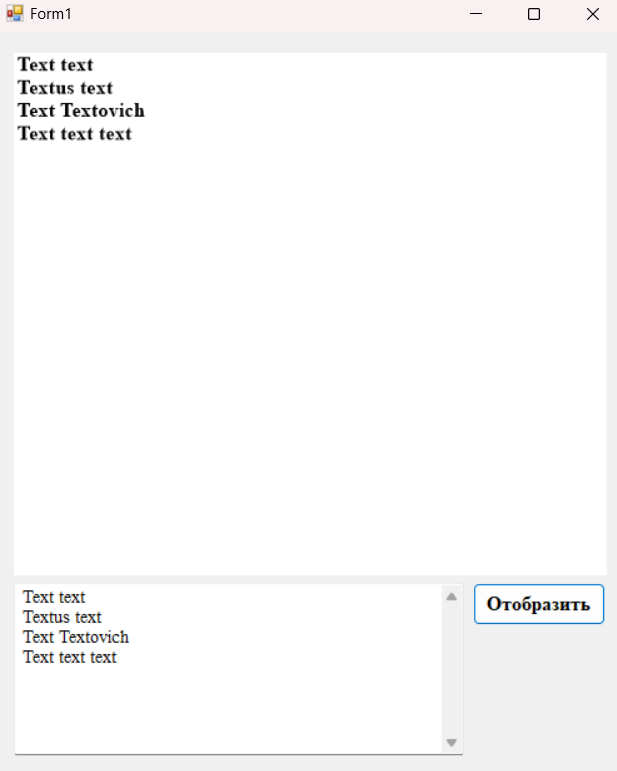


Рис. 1.3.4. Изменение цвета и шрифта

**1.4. Занятие № 4. Элемент MenuStrip и свойство Anchor, Открытие и запись текстового файла**

Рассматривается элемент "MenuStrip", который вы можете встретить: в текстовом редакторе, выбирая "Файл -> Сохранить как", в графическом редакторе "Файл -> Вставить" и во многих других программах. "MenuStrip" – это, по сути, выпадающее меню, в котором есть определѐнные пункты. Так же будет рассмотрено свойство "Anchor" – это свойство, при котором определяется к каким сторонам формы будет привязан элемент, если к правой, то при увеличении или уменьшении размера формы – правая сторона "textBox", так же увеличиваться или уменьшаться. Если поставлена кнопка "button", то она будет перемещаться за стороной формы, к которой привязана.

Пример работы программы:

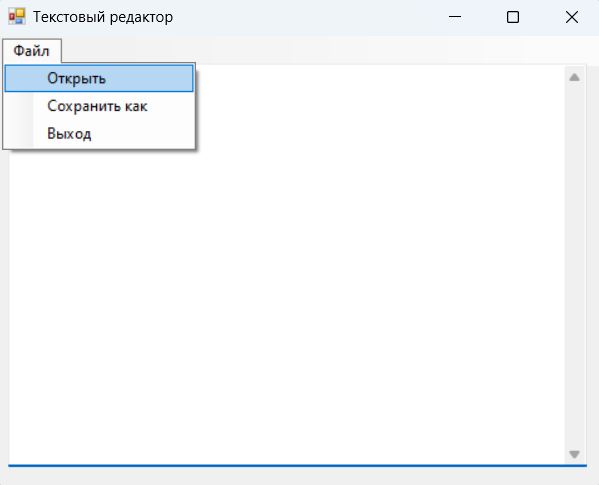


Рис. 1.4.1-1.4.2 Открытие, сохранение текстового файла

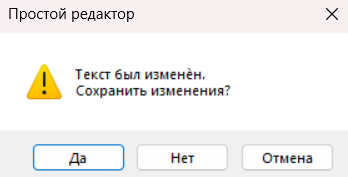
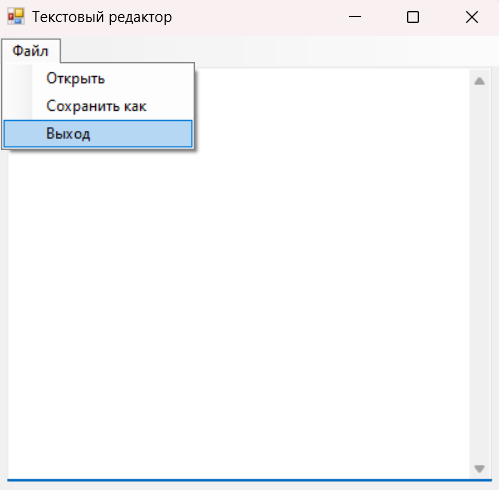


Рис. 1.4.3-1.4.4 Выход из редактора, предупреждение о несохранении нового текста при закрытии

**1.5. Занятие № 5. Рисование линий, треугольника, эллипса и окружности в PictureBox**

**1.5.1**

Изучение базовых приемов рисования в "PictureBox". Рассмотрим, как нарисовать линию заданной длины, цвета и в заданном месте поля "PictureBox". В “textBox”ы мы будем записывать начальное и конечное значение координат поля рисования: две координаты – одна точка, еще две координаты – еще одна точка, которая соединяется с предыдущей, образуя отрезок, заданного в коде программы цвета.

Пример работы программы:

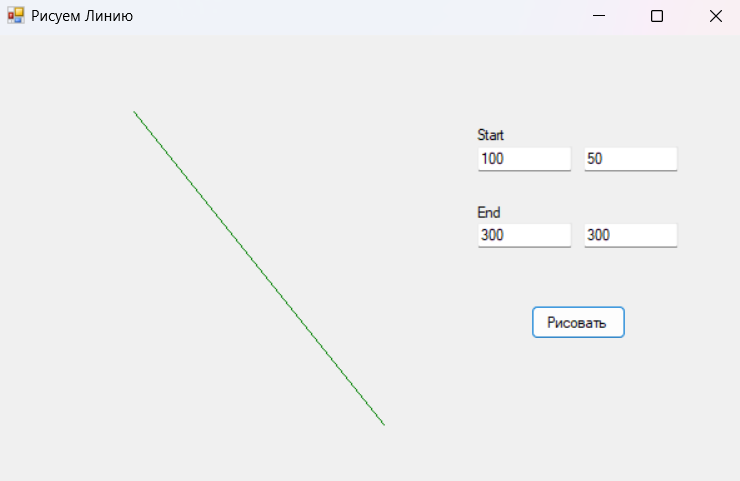


Рис. 1.5.1. Рисование линии

**1.5.2**

После того, как вы узнали, как рисовать линию, можно уже будет нарисовать какую-нибудь простую фигуру – например треугольник. Для этого нужно нарисовать всего лишь три точки, а не шесть. Сначала может показать, что нужно рисовать три отрезка по отдельности и пытаться задать такие координаты, чтобы получился треугольник – это излишнее и неэффективное нагромождение программы. Поэтому нужно сделать так, чтобы конец одного отрезка являлся началом другого – что, собственно, мы и сделаем. Нужно задавать массив из шести элементов для создания трех точек.

Пример работы программы:

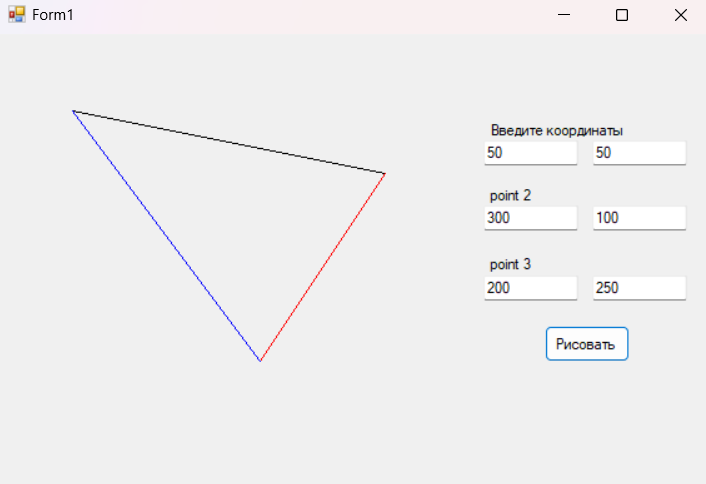


Рис. 1.5.2. Рисование треугольника

**1.5.3**

Пример создания "Пера" или "Карандаша", которые будет задавать цвет контура фигуры. Чтобы нарисовать эллипс нужно указать координаты верхнего левого угла условного прямоугольника, в котором находится рисуемая фигура, и радиусы. Если горизонтальный и вертикальный радиусы равны, то получится окружность, а если нет, то получится эллипс.

Пример работы программы:



Рис. 1.5.3. Рисование эллипса

**1.5.4**

Что бы фигура стала закрашенной, нужно создать "заливку" типа "Brush", указав цвет заливки. Создадим проект, в котором с помощью "comboBox" будем выбирать цвет заливки нарисованного прямоугольника.

Пример работы программы:

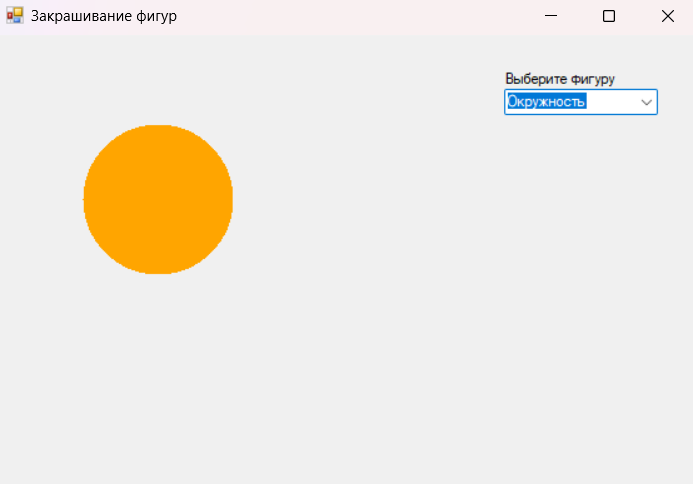


Рис. 1.5.4. Рисование закрашенной фигуры на выбор

**1.6. Занятие № 6. Событие MouseHover**

Знакомство с “событием” "MousHover". Идея вполне простая – если вы наводите на один из элементов, на которой описано данное событие, то элемент как-либо изменяется - как напишите, так и изменится. В данном примере, при наведении курсора мыши на надпись "label" после чего изменяется текст и цвет текста, а также “выскакивает \” “MessageBox”, с каким-нибудь текстом.

Пример работы программы:

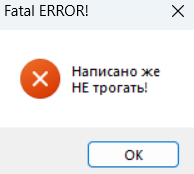
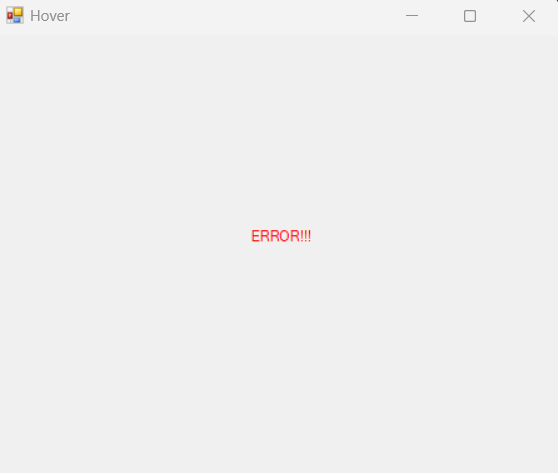
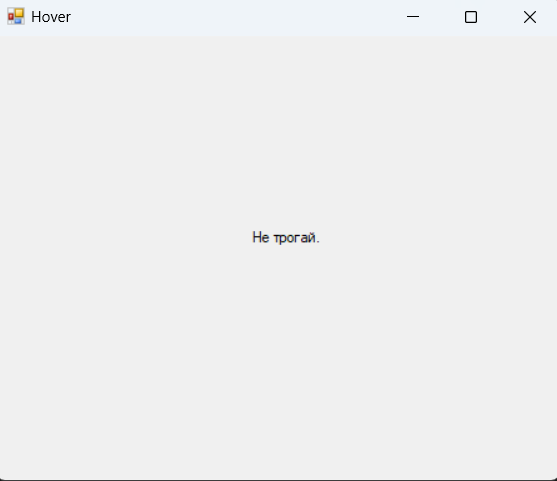
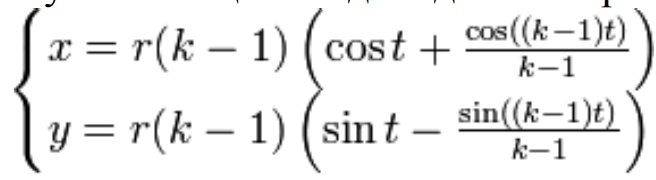
****

Рис. 1.6.1-1.6.3. MessageBox и событие MouseHover

**1.7. Формирование траектории для движения простого геометрического объекта**

Пусть гипоциклоида задается параметрическим уравнением:



Обеспечить формирование изображения траектории и анимацию движения катящейся окружности.

Пример работы программы:

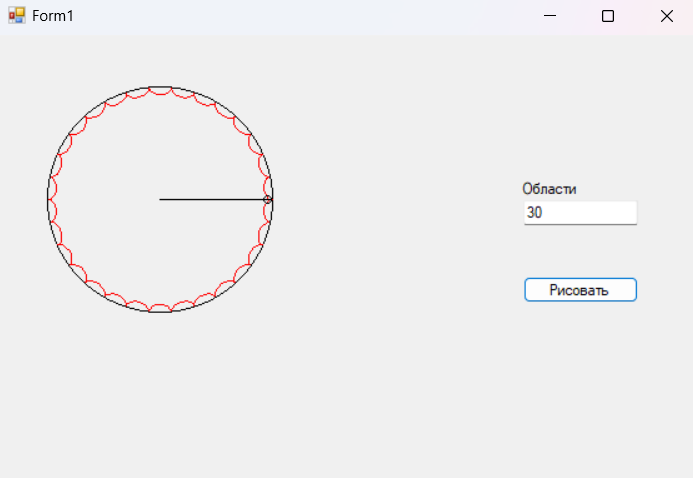
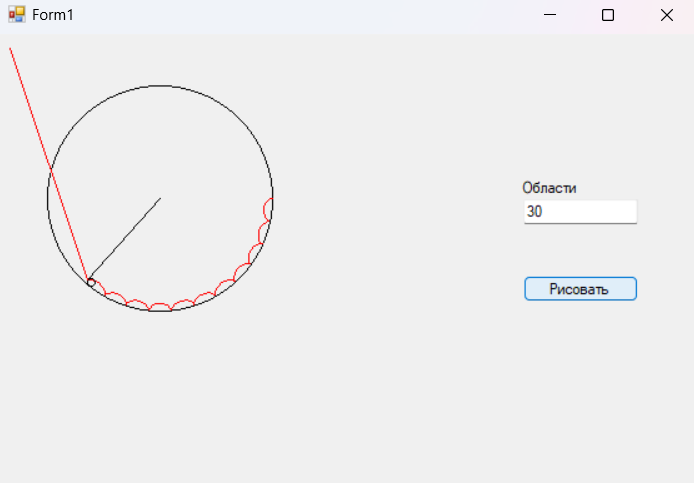


Рис. 1.7.1 и 1.7.2. Рисование траектории

**2. Движение геометрического объекта по траектории**

**2.1. Формулировка задания**

Реализовать движение геометрического объекта по заданной траектории.

Вариант задания: 36-11-3

Заданная траектория:

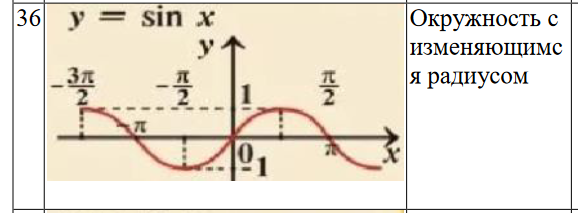


Рис.2.1. Задание

Объект: шестиугольник

Наличие у объекта цвета: контур заданного цвета без заливки

**2.2. Математическая постановка**

Траектория считается через данное уравнение:

y = sinx

За «х» берутся значения в диапазоне [-2π; 2π].

Для построения ромба берутся точки на окружности с расстоянием между ними и заданный пользователем радиус (в форме - size). Координаты точки можно получить, вычислив косинус и синус выбранного угла и умножив на радиус.

Для отображения точек на экран используются преобразования с помощью переменных масштаба и сдвига (сдвиг нужен для того, чтобы точка (0; 0) была в центре экрана)

**2.3. Описание интерфейса**

«Скелет» приложения:

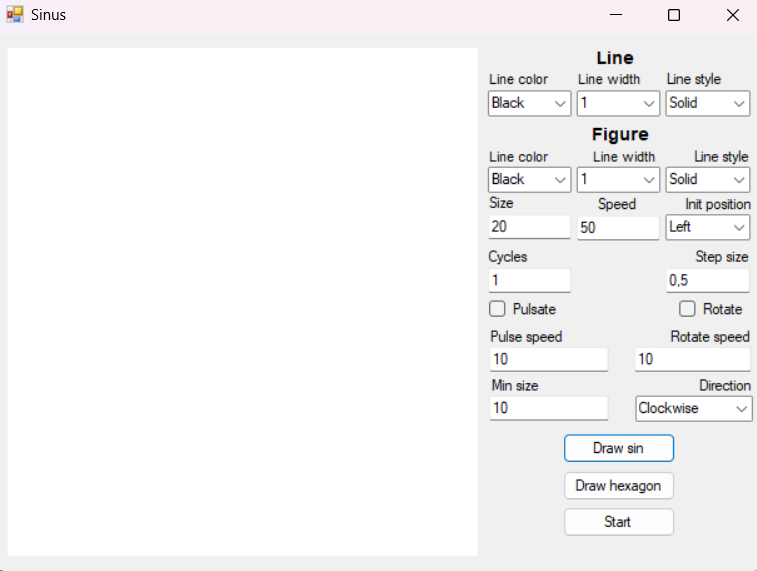


Рис. 2.3.1 Форма

Форма с введенными данными и выводом задания:

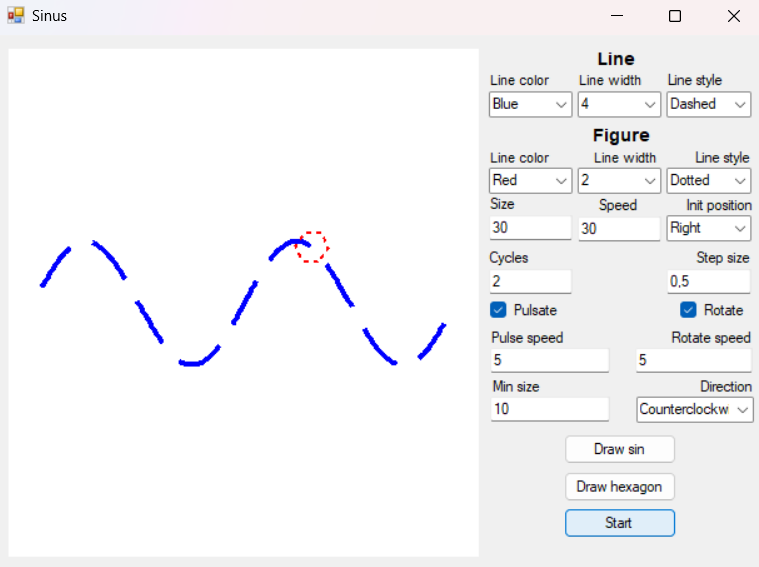
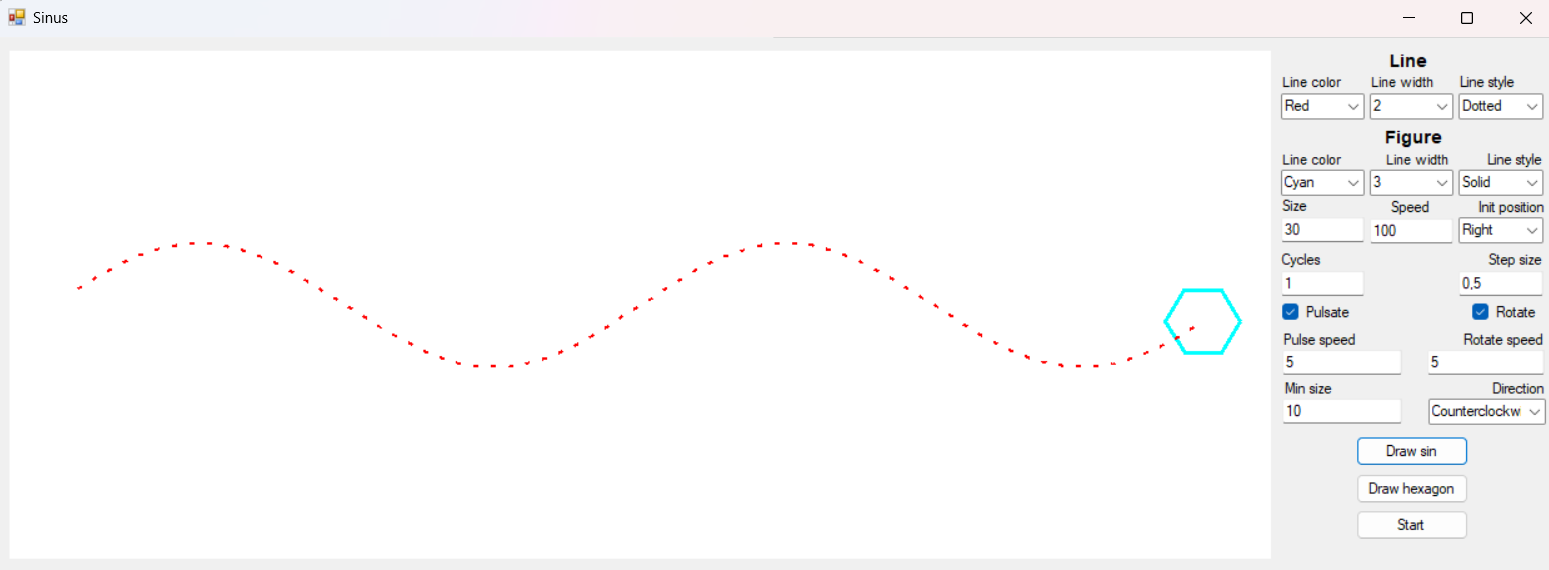


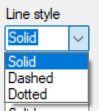
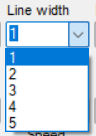
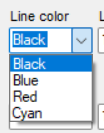
Рис.2.3.2. Форма с выводом

Форма при масштабировании:

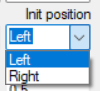


В форме вводятся данные для траектории, шестиугольника и его движения по траектории. Для траектории вводятся цвет, ширина и стиль линии, для шестиугольника - цвет, ширина и стиль линии, размер (радиус), скорость движения по траектории, начальное положение (слева или справа), количество циклов, размер шага, должен ли объект пульсировать и вращаться, скорость пульсации и вращения, минимальный размер при пульсации и направление вращения (по часовой стрелке или против). Кнопки «Draw sin» и «Draw hexagon» отвечают за рисование объектов, «Start» начинает движение шестиугольника по траектории.

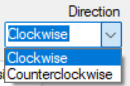
Выбор цвета, ширины и стиля линии:



Выбор начального положения:



Выбор направления поворота:



**2.4. Описание реализации**

Работа программы основана на нажатии кнопок, после нажатия происходят все вычисления. Кнопка №1 отвечает за построение траектории. Если траектория нарисована, она удаляется, после чего выводится новая:

private void button1\_Click(object sender, EventArgs e)

{

DeleteSin();

DrawSin(Line\_color, Line\_width, Line\_style);

Sin\_activated = true;

}

private void DrawSin(Color line\_color, int line\_width, float[] line\_style)

{

ResetCoefficients();

int i = 0;

for (double x = Left\_limit; x <= Right\_limit; x += Step)

{

Points[i] = new Point((int)(x \* Scale\_x) + Shift\_x, (int)(-F(x) \* Scale\_y) + Shift\_y);

i++;

}

Points[Points.Length - 1] = Points[Points.Length - 2];

Pen pen = new Pen(line\_color, line\_width);

if (line\_style != null) pen.DashPattern = line\_style;

Graphics gr = pictureBox1.CreateGraphics();

gr.DrawLines(pen, Points);

Current\_line\_color = line\_color;

Current\_line\_width = line\_width;

Current\_line\_style = line\_style;

}

private void DeleteSin()

{

int i = 0;

for (double x = Left\_limit; x <= Right\_limit; x += Step)

{

Points[i] = new Point((int)(x \* Scale\_x) + Shift\_x, (int)(-F(x) \* Scale\_y) + Shift\_y);

i++;

}

Points[Points.Length - 1] = Points[Points.Length - 2];

Pen pen = new Pen(pictureBox1.BackColor, Current\_line\_width);

Graphics gr = pictureBox1.CreateGraphics();

if (Current\_line\_style != null) pen.DashPattern = Current\_line\_style;

gr.DrawLines(pen, Points);

if (Hex\_activated) DrawHexagon(Hex\_line\_color, Hex\_line\_width, Hex\_line\_style);

}

Построение шестиугольника происходит подобным образом при нажатии кнопки №2, но при этом учитывается угол поворота и состояние до задания параметров и нажатия на кнопку:

private void DrawHexagon(Color line\_color, int line\_width, float[] line\_style)

{

ResetCoefficients();

Point[] points = new Point[7];

for (int i = 0; i < 6; i++)

points[i] = new Point(Hex\_center.X + (int)(Hex\_radius \* (Math.Cos(Math.PI \* i / 3 + Angle))), Hex\_center.Y + (int)(Hex\_radius \* (Math.Sin(Math.PI \* i / 3 + Angle))));

points[6] = points[0];

Pen pen = new Pen(line\_color, line\_width);

Graphics gr = pictureBox1.CreateGraphics();

if (line\_style != null) pen.DashPattern = line\_style;

gr.DrawLines(pen, points);

Current\_hex\_line\_width = line\_width;

Current\_hex\_line\_style = line\_style;

Current\_hex\_radius = Hex\_radius;

Current\_hex\_center = Hex\_center;

Current\_angle = Angle;

}

private void DeleteHexagon()

{

Point[] points = new Point[7];

for (int i = 0; i < 6; i++)

points[i] = new Point(Current\_hex\_center.X + (int)(Current\_hex\_radius \* (Math.Cos(Math.PI \* i / 3 + Current\_angle))), Current\_hex\_center.Y + (int)(Current\_hex\_radius \* (Math.Sin(Math.PI \* i / 3 + Current\_angle))));

points[6] = points[0];

Pen pen = new Pen(pictureBox1.BackColor, Current\_hex\_line\_width);

Graphics gr = pictureBox1.CreateGraphics();

if (Current\_hex\_line\_style != null) pen.DashPattern = Current\_hex\_line\_style;

gr.DrawLines(pen, points);

if (Sin\_activated) DrawSin(Current\_line\_color, Current\_line\_width, Current\_line\_style);

}

private void button2\_Click(object sender, EventArgs e)

{

DeleteHexagon();

Angle = 0;

switch (comboBox7.SelectedIndex)

{

case 0:

Hex\_center = new Point(Left\_limit \* Scale\_x + Shift\_x, (int)(-F(Left\_limit) \* Scale\_y + Shift\_y));

break;

case 1:

Hex\_center = new Point(Right\_limit \* Scale\_x + Shift\_x, (int)(-F(Right\_limit) \* Scale\_y + Shift\_y));

break;

}

try

{

Hex\_radius = int.Parse(textBox1.Text);

if (Hex\_radius < 0) throw new FormatException();

}

catch (FormatException)

{

MessageBox.Show("Error: Invalid size\nCannot draw hexagon", "Error");

return;

}

DrawHexagon(Hex\_line\_color, Hex\_line\_width, Hex\_line\_style);

Hex\_activated = true;

}

Кнопка №3 отвечает за движение шестиугольника по траектории. После нажатия на нее считываются параметры, связанные с движением, после чего запускается цикл, в котором объект проходит заданное число циклов, при этом, если это задано, объект вращается и пульсирует с заданными параметрами:

private void button3\_Click(object sender, EventArgs e)

{

if (!Hex\_activated || !Sin\_activated) return;

bool OK = InitMovement();

if (!OK) return;

int radius = Hex\_radius;

bool grows = false;

int delay\_time = (int)(10000 / Speed \* Move\_step);

double base\_x = (Hex\_center.X - Shift\_x) / Scale\_x;

int init\_direction = (base\_x <= 0) ? 1 : -1;

int direction = init\_direction;

int right = Right\_limit \* Scale\_x + Shift\_x;

int left = Left\_limit \* Scale\_x + Shift\_x;

int rotation\_direction = Counterclockwise ? 1 : -1;

if (!Is\_rotating) rotation\_direction = 0;

int cycle\_count = 0;

while (cycle\_count < Cycles)

{

while (direction == 1 && Hex\_center.X < right || direction == -1 && Hex\_center.X > left)

{

DeleteHexagon();

if (grows)

{

Hex\_radius += (int)(Move\_step \* Pulsation\_speed);

if (Hex\_radius >= radius)

{

Hex\_radius = radius;

grows = false;

}

}

else

{

Hex\_radius -= (int)(Move\_step \* Pulsation\_speed);

if (Hex\_radius <= Min\_size)

{

Hex\_radius = Min\_size;

grows = true;

}

}

base\_x += Move\_step \* direction;

Hex\_center.X = (int)(base\_x \* Scale\_x) + Shift\_x;

Hex\_center.Y = (int)(-F(base\_x) \* Scale\_y) + Shift\_y;

Angle += Move\_step \* Rotation\_speed \* rotation\_direction;

DrawHexagon(Hex\_line\_color, Hex\_line\_width, Hex\_line\_style);

System.Threading.Thread.Sleep(delay\_time);

}

direction = -direction;

if (direction == init\_direction) cycle\_count++;

}

}

**2.5. Текст программы**

|  |  |
| --- | --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.IO;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;    namespace work1  {  public partial class Form1 : Form  {  private int Shift\_x;  private int Shift\_y;  private int Scale\_x;  private int Scale\_y;    // line  private Color Line\_color;  private Color Current\_line\_color;  private int Line\_width;  private int Current\_line\_width;  private float[] Line\_style;  private float[] Current\_line\_style;    private int Left\_limit;  private int Right\_limit;  private double Step;  private Point[] Points;    private bool Sin\_activated = false;  private bool Hex\_activated = false;    // hexagon  private Point Hex\_center;  private Point Current\_hex\_center;  private int Hex\_radius;  private int Current\_hex\_radius;  private Color Hex\_line\_color;  private int Hex\_line\_width;  private int Current\_hex\_line\_width;  private float[] Hex\_line\_style;  private float[] Current\_hex\_line\_style;    private int Speed;  private double Move\_step;  private int Cycles;  private bool Is\_pulsating = false;  private int Pulsation\_speed;  private int Min\_size;  private bool Is\_rotating = false;  private int Rotation\_speed;  private bool Counterclockwise = false;  private double Angle;  private double Current\_angle;    public Form1()  {  InitializeComponent();  }    private void Form1\_Load(object sender, EventArgs e)  {  // default values  // sin  Line\_color = Color.Black;  Line\_width = 1;  Line\_style = null;    Left\_limit = (int)(-2 \* Math.PI);  Right\_limit = (int)(2 \* Math.PI);  Step = 0.05;  Points = new Point[(int)((Right\_limit - Left\_limit) / Step + 1)];    ResetCoefficients();  Scale\_y = 50;    // hexagon  Hex\_line\_color = Color.Black;  Hex\_line\_width = 1;    Hex\_radius = 20;  Hex\_center = new Point(Left\_limit \* Scale\_x + Shift\_x, (int)(-F(Left\_limit) \* Scale\_y + Shift\_y));  Speed = 50;  Move\_step = 0.5;  Cycles = 1;  Pulsation\_speed = 10;  Min\_size = (Hex\_radius / 2);  Rotation\_speed = 10;  Angle = 0;    // form  Text = "Sinus";  pictureBox1.BackColor = Color.White;    comboBox1.Text = "Black";  comboBox1.Items.AddRange(new string[] { "Black", "Blue", "Red", "Cyan" });    comboBox2.Text = "1";  comboBox2.Items.AddRange(new string[] { "1", "2", "3", "4", "5" });    comboBox3.Text = "Solid";  comboBox3.Items.AddRange(new string[] { "Solid", "Dashed", "Dotted" });    comboBox4.Text = "Black";  comboBox4.Items.AddRange(new string[] { "Black", "Blue", "Red", "Cyan" });    comboBox5.Text = "1";  comboBox5.Items.AddRange(new string[] { "1", "2", "3" });    comboBox6.Text = "Solid";  comboBox6.Items.AddRange(new string[] { "Solid", "Dashed", "Dotted" });    comboBox7.Text = "Left";  comboBox7.Items.AddRange(new string[] { "Left", "Right" });    comboBox8.Text = "Clockwise";  comboBox8.Items.AddRange(new string[] { "Clockwise", "Counterclockwise" });    textBox1.Text = Hex\_radius.ToString();  textBox2.Text = Pulsation\_speed.ToString();  textBox3.Text = Min\_size.ToString();  textBox4.Text = Speed.ToString();  textBox5.Text = Move\_step.ToString();  textBox6.Text = Cycles.ToString();  textBox7.Text = Rotation\_speed.ToString();  }    private double F(double x)  {  return Math.Sin(x);  }    private void ResetCoefficients()  {  Shift\_x = pictureBox1.Width / 2;  Shift\_y = pictureBox1.Height / 2;  Scale\_x = pictureBox1.Width / (Right\_limit - Left\_limit) \* 90 / 100;  }    private void DrawSin(Color line\_color, int line\_width, float[] line\_style)  {  ResetCoefficients();    int i = 0;  for (double x = Left\_limit; x <= Right\_limit; x += Step)  {  Points[i] = new Point((int)(x \* Scale\_x) + Shift\_x, (int)(-F(x) \* Scale\_y) + Shift\_y);  i++;  }    Points[Points.Length - 1] = Points[Points.Length - 2];    Pen pen = new Pen(line\_color, line\_width);  if (line\_style != null) pen.DashPattern = line\_style;  Graphics gr = pictureBox1.CreateGraphics();  gr.DrawLines(pen, Points);    Current\_line\_color = line\_color;  Current\_line\_width = line\_width;  Current\_line\_style = line\_style;  }    private void DeleteSin()  {  int i = 0;  for (double x = Left\_limit; x <= Right\_limit; x += Step)  {  Points[i] = new Point((int)(x \* Scale\_x) + Shift\_x, (int)(-F(x) \* Scale\_y) + Shift\_y);  i++;  }    Points[Points.Length - 1] = Points[Points.Length - 2];    Pen pen = new Pen(pictureBox1.BackColor, Current\_line\_width);  Graphics gr = pictureBox1.CreateGraphics();  if (Current\_line\_style != null) pen.DashPattern = Current\_line\_style;  gr.DrawLines(pen, Points);  if (Hex\_activated) DrawHexagon(Hex\_line\_color, Hex\_line\_width, Hex\_line\_style);  }    private void DrawHexagon(Color line\_color, int line\_width, float[] line\_style)  {  ResetCoefficients();    Point[] points = new Point[7];  for (int i = 0; i < 6; i++)  points[i] = new Point(Hex\_center.X + (int)(Hex\_radius \* (Math.Cos(Math.PI \* i / 3 + Angle))), Hex\_center.Y + (int)(Hex\_radius \* (Math.Sin(Math.PI \* i / 3 + Angle))));  points[6] = points[0];    Pen pen = new Pen(line\_color, line\_width);  Graphics gr = pictureBox1.CreateGraphics();  if (line\_style != null) pen.DashPattern = line\_style;  gr.DrawLines(pen, points);    Current\_hex\_line\_width = line\_width;  Current\_hex\_line\_style = line\_style;  Current\_hex\_radius = Hex\_radius;  Current\_hex\_center = Hex\_center;  Current\_angle = Angle;  }    private void DeleteHexagon()  {  Point[] points = new Point[7];  for (int i = 0; i < 6; i++)  points[i] = new Point(Current\_hex\_center.X + (int)(Current\_hex\_radius \* (Math.Cos(Math.PI \* i / 3 + Current\_angle))), Current\_hex\_center.Y + (int)(Current\_hex\_radius \* (Math.Sin(Math.PI \* i / 3 + Current\_angle))));  points[6] = points[0];    Pen pen = new Pen(pictureBox1.BackColor, Current\_hex\_line\_width);  Graphics gr = pictureBox1.CreateGraphics();  if (Current\_hex\_line\_style != null) pen.DashPattern = Current\_hex\_line\_style;  gr.DrawLines(pen, points);  if (Sin\_activated) DrawSin(Current\_line\_color, Current\_line\_width, Current\_line\_style);  }    private void button1\_Click(object sender, EventArgs e)  {  DeleteSin();  DrawSin(Line\_color, Line\_width, Line\_style);  Sin\_activated = true;  }    private void button2\_Click(object sender, EventArgs e)  {  DeleteHexagon();  Angle = 0;  switch (comboBox7.SelectedIndex)  {  case 0:  Hex\_center = new Point(Left\_limit \* Scale\_x + Shift\_x, (int)(-F(Left\_limit) \* Scale\_y + Shift\_y));  break;  case 1:  Hex\_center = new Point(Right\_limit \* Scale\_x + Shift\_x, (int)(-F(Right\_limit) \* Scale\_y + Shift\_y));  break;  }    try  {  Hex\_radius = int.Parse(textBox1.Text);  if (Hex\_radius < 0) throw new FormatException();  }  catch (FormatException)  {  MessageBox.Show("Error: Invalid size\nCannot draw hexagon", "Error");  return;  } | DrawHexagon(Hex\_line\_color, Hex\_line\_width, Hex\_line\_style);  Hex\_activated = true;  }    private void button3\_Click(object sender, EventArgs e)  {  if (!Hex\_activated || !Sin\_activated) return;  bool OK = InitMovement();  if (!OK) return;    int radius = Hex\_radius;  bool grows = false;  int delay\_time = (int)(10000 / Speed \* Move\_step);  double base\_x = (Hex\_center.X - Shift\_x) / Scale\_x;    int init\_direction = (base\_x <= 0) ? 1 : -1;  int direction = init\_direction;  int right = Right\_limit \* Scale\_x + Shift\_x;  int left = Left\_limit \* Scale\_x + Shift\_x;    int rotation\_direction = Counterclockwise ? 1 : -1;  if (!Is\_rotating) rotation\_direction = 0;    int cycle\_count = 0;  while (cycle\_count < Cycles)  {  while (direction == 1 && Hex\_center.X < right || direction == -1 && Hex\_center.X > left)  {  DeleteHexagon();  if (grows)  {  Hex\_radius += (int)(Move\_step \* Pulsation\_speed);  if (Hex\_radius >= radius)  {  Hex\_radius = radius;  grows = false;  }  }  else  {  Hex\_radius -= (int)(Move\_step \* Pulsation\_speed);  if (Hex\_radius <= Min\_size)  {  Hex\_radius = Min\_size;  grows = true;  }  }    base\_x += Move\_step \* direction;  Hex\_center.X = (int)(base\_x \* Scale\_x) + Shift\_x;  Hex\_center.Y = (int)(-F(base\_x) \* Scale\_y) + Shift\_y;    Angle += Move\_step \* Rotation\_speed \* rotation\_direction;    DrawHexagon(Hex\_line\_color, Hex\_line\_width, Hex\_line\_style);  System.Threading.Thread.Sleep(delay\_time);  }  direction = -direction;  if (direction == init\_direction) cycle\_count++;  }  }  private bool InitMovement()  {  try  {  Speed = int.Parse(textBox4.Text);  if (Speed < 0) throw new FormatException();  }  catch (FormatException)  {  MessageBox.Show("Error: Invalid speed\nSpeed must be a positive integer", "Error");  return false;  }    try  {  Move\_step = double.Parse(textBox5.Text);  if (Move\_step < 0) throw new FormatException();  }  catch (FormatException)  {  MessageBox.Show("Error: Invalid step\nStep must be a positive float number (with comma)", "Error");  return false;  }    try  {  Cycles = int.Parse(textBox6.Text);  if (Cycles < 0) throw new FormatException();  }  catch (FormatException)  {  MessageBox.Show("Error: Invalid cycle number\nCycle number must be a positive integer", "Error");  return false;  }    if (Is\_pulsating)  {  try  {  Pulsation\_speed = int.Parse(textBox2.Text);  if (Pulsation\_speed < 0) throw new FormatException();  }  catch (FormatException)  {  MessageBox.Show("Error: Invalid speed\nPulsation speed must be a positive integer", "Error");  return false;  }    try  {  Min\_size = int.Parse(textBox3.Text);  if (Min\_size > Hex\_radius) throw new InvalidDataException("Min size cannot be bigger than size");  if (Min\_size < 0) throw new FormatException();  }  catch (FormatException)  {  MessageBox.Show("Error: Invalid min size\nMin size must be a positive integer", "Error");  return false;  }  catch (InvalidDataException ex)  {  MessageBox.Show($"Error: {ex.Message}", "Error");  return false;  }  }  else Pulsation\_speed = 0;    if (Is\_rotating)  {  try  {  Rotation\_speed = int.Parse(textBox7.Text);  if (Rotation\_speed < 0) throw new FormatException();  }  catch (FormatException)  {  MessageBox.Show("Error: Invalid speed\nRotation speed must be a positive integer", "Error");  return false;  }  }    return true;  }    private void Form1\_ResizeEnd(object sender, EventArgs e)  {  int old\_shift\_x = Shift\_x;  int old\_shift\_y = Shift\_y;  int old\_scale\_x = Scale\_x;  ResetCoefficients();    double base\_x = (double)(Hex\_center.X - old\_shift\_x) / old\_scale\_x;  double base\_y = Hex\_center.Y - old\_shift\_y;  Hex\_center = new Point((int)(base\_x \* Scale\_x + Shift\_x), (int)(base\_y + Shift\_y));    if (!Sin\_activated && !Hex\_activated) return;  Graphics gr = pictureBox1.CreateGraphics();  gr.Clear(pictureBox1.BackColor);    if (Hex\_activated) DrawHexagon(Hex\_line\_color, Hex\_line\_width, Hex\_line\_style);  if (Sin\_activated) DrawSin(Line\_color, Line\_width, Line\_style);    }    private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)  {  switch (comboBox1.SelectedIndex)  {  case 0:  Line\_color = Color.Black;  return;  case 1:  Line\_color = Color.Blue;  return;  case 2:  Line\_color = Color.Red;  return;  case 3:  Line\_color = Color.Cyan;  return;  }  }    private void comboBox2\_SelectedIndexChanged(object sender, EventArgs e)  {  Line\_width = comboBox2.SelectedIndex + 1;  }    private void comboBox3\_SelectedIndexChanged(object sender, EventArgs e)  {  switch (comboBox3.SelectedIndex)  {  case 0:  Line\_style = null;  return;  case 1:  Line\_style = new float[] { 10, 5 };  return;  case 2:  Line\_style = new float[] { 2, 5 };  return;  }  }    private void comboBox4\_SelectedIndexChanged(object sender, EventArgs e)  {  switch (comboBox4.SelectedIndex)  {  case 0:  Hex\_line\_color = Color.Black;  return;  case 1:  Hex\_line\_color = Color.Blue;  return;  case 2:  Hex\_line\_color = Color.Red;  return;  case 3:  Hex\_line\_color = Color.Cyan;  return;  }  }    private void comboBox5\_SelectedIndexChanged(object sender, EventArgs e)  {  Hex\_line\_width = comboBox5.SelectedIndex + 1;  }    private void comboBox6\_SelectedIndexChanged(object sender, EventArgs e)  {  switch (comboBox6.SelectedIndex)  {  case 0:  Hex\_line\_style = null;  return;  case 1:  Hex\_line\_style = new float[] { 5, 2 };  return;  case 2:  Hex\_line\_style = new float[] { 2, 2 };  return;  }  }    private void checkBox1\_CheckedChanged(object sender, EventArgs e)  {  if (checkBox1.Checked) Is\_pulsating = true;  else Is\_pulsating = false;  }    private void checkBox2\_CheckedChanged(object sender, EventArgs e)  {  if (checkBox2.Checked) Is\_rotating = true;  else Is\_rotating = false;  }    private void comboBox8\_SelectedIndexChanged(object sender, EventArgs e)  {  switch (comboBox8.SelectedIndex)  {  case 0:  Counterclockwise = false;  return;  case 1:  Counterclockwise = true;  return;  }  }  }  } |

**3. Формирование последовательного изображения Фракталов (G шагов генерации) с отображением формирующихся уровней на дереве**

**3.1. Формулировка задания**

.

Вариант задания: 36-11-3

Заданная траектория:

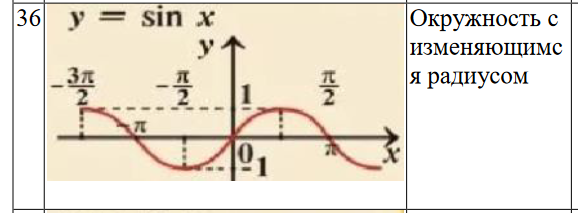


Рис.2.1. Задание

Объект: шестиугольник

Наличие у объекта цвета: контур заданного цвета без заливки

**2.2. Математическая постановка**

Траектория считается через данное уравнение:

y = sinx

За «х» берутся значения в диапазоне [-2π; 2π].

Для построения ромба берутся точки на окружности с расстоянием между ними и заданный пользователем радиус (в форме - size). Координаты точки можно получить, вычислив косинус и синус выбранного угла и умножив на радиус.

Для отображения точек на экран используются преобразования с помощью переменных масштаба и сдвига (сдвиг нужен для того, чтобы точка (0; 0) была в центре экрана)

**2.3. Описание интерфейса**

«Скелет» приложения:

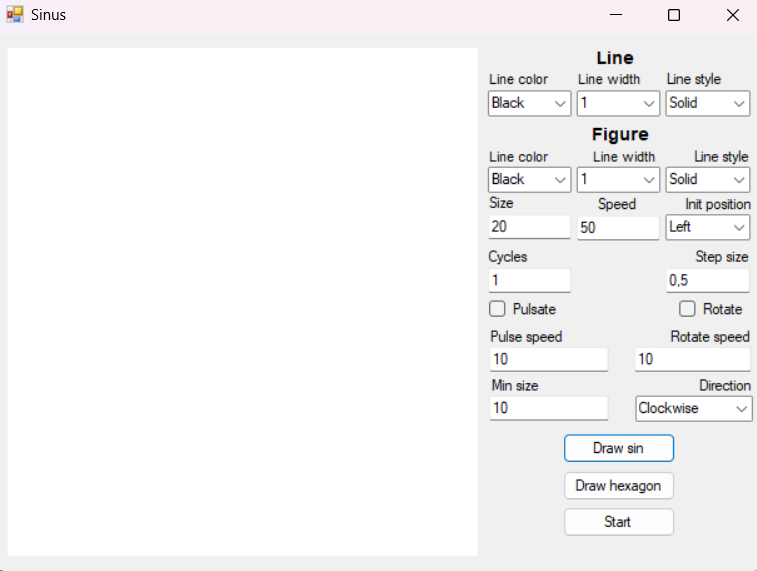


Рис. 2.3.1 Форма

Форма с введенными данными и выводом задания:

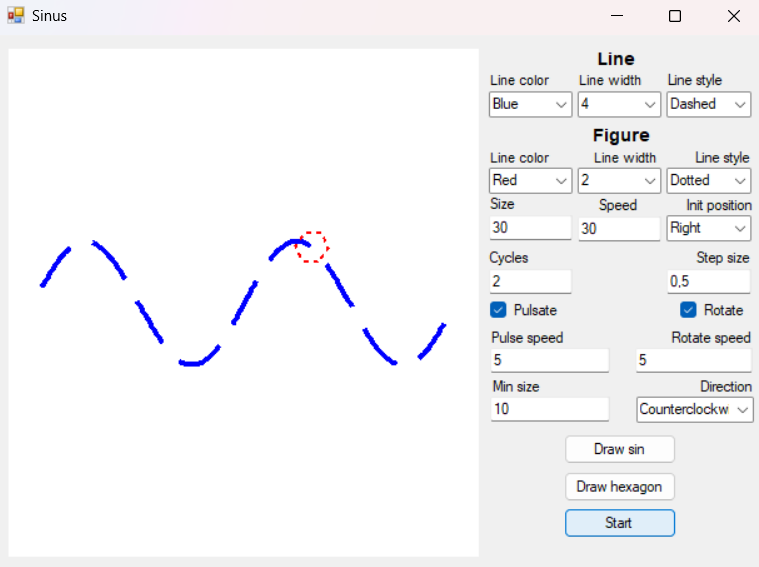
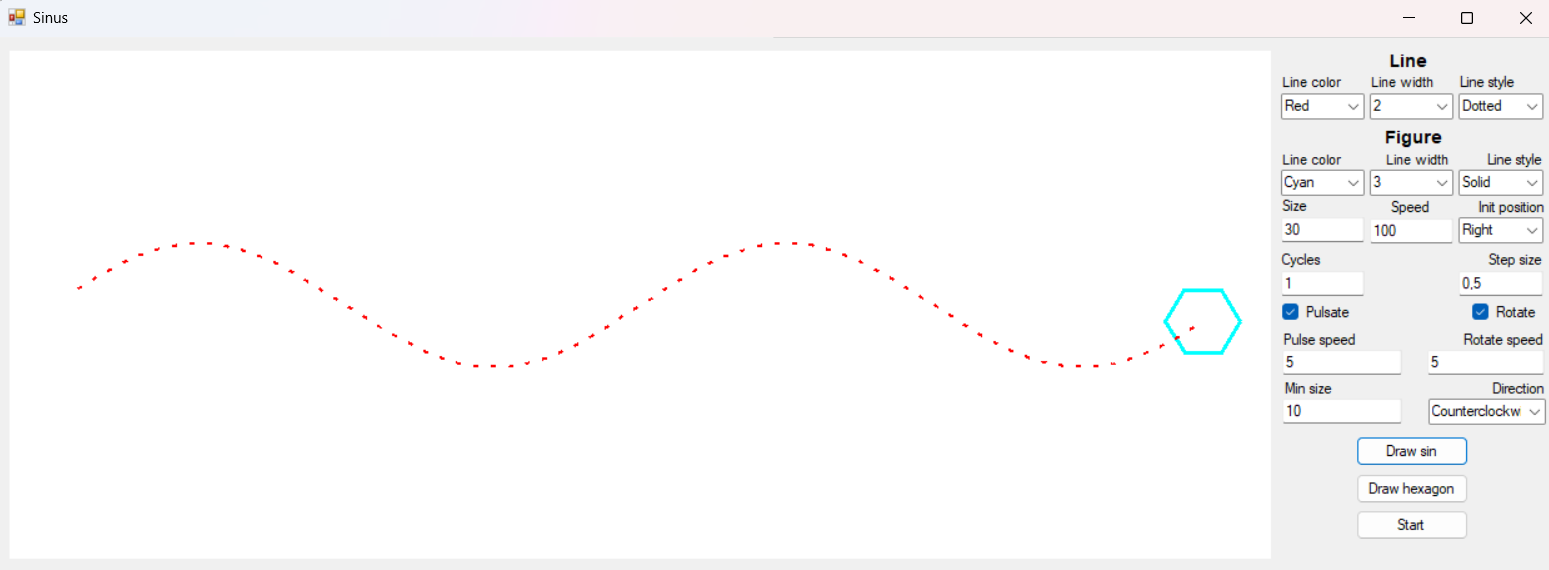


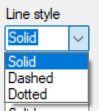
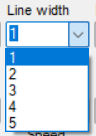
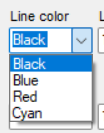
Рис.2.3.2. Форма с выводом

Форма при масштабировании:

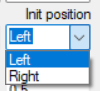


В форме вводятся данные для траектории, шестиугольника и его движения по траектории. Для траектории вводятся цвет, ширина и стиль линии, для шестиугольника - цвет, ширина и стиль линии, размер (радиус), скорость движения по траектории, начальное положение (слева или справа), количество циклов, размер шага, должен ли объект пульсировать и вращаться, скорость пульсации и вращения, минимальный размер при пульсации и направление вращения (по часовой стрелке или против). Кнопки «Draw sin» и «Draw hexagon» отвечают за рисование объектов, «Start» начинает движение шестиугольника по траектории.

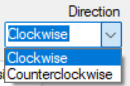
Выбор цвета, ширины и стиля линии:



Выбор начального положения:



Выбор направления поворота:



**2.4. Описание реализации**

Работа программы основана на нажатии кнопок, после нажатия происходят все вычисления. Кнопка №1 отвечает за построение траектории. Если траектория нарисована, она удаляется, после чего выводится новая:

private void button1\_Click(object sender, EventArgs e)

{

DeleteSin();

DrawSin(Line\_color, Line\_width, Line\_style);

Sin\_activated = true;

}

private void DrawSin(Color line\_color, int line\_width, float[] line\_style)

{

ResetCoefficients();

int i = 0;

for (double x = Left\_limit; x <= Right\_limit; x += Step)

{

Points[i] = new Point((int)(x \* Scale\_x) + Shift\_x, (int)(-F(x) \* Scale\_y) + Shift\_y);

i++;

}

Points[Points.Length - 1] = Points[Points.Length - 2];

Pen pen = new Pen(line\_color, line\_width);

if (line\_style != null) pen.DashPattern = line\_style;

Graphics gr = pictureBox1.CreateGraphics();

gr.DrawLines(pen, Points);

Current\_line\_color = line\_color;

Current\_line\_width = line\_width;

Current\_line\_style = line\_style;

}

private void DeleteSin()

{

int i = 0;

for (double x = Left\_limit; x <= Right\_limit; x += Step)

{

Points[i] = new Point((int)(x \* Scale\_x) + Shift\_x, (int)(-F(x) \* Scale\_y) + Shift\_y);

i++;

}

Points[Points.Length - 1] = Points[Points.Length - 2];

Pen pen = new Pen(pictureBox1.BackColor, Current\_line\_width);

Graphics gr = pictureBox1.CreateGraphics();

if (Current\_line\_style != null) pen.DashPattern = Current\_line\_style;

gr.DrawLines(pen, Points);

if (Hex\_activated) DrawHexagon(Hex\_line\_color, Hex\_line\_width, Hex\_line\_style);

}

Построение шестиугольника происходит подобным образом при нажатии кнопки №2, но при этом учитывается угол поворота и состояние до задания параметров и нажатия на кнопку:

private void DrawHexagon(Color line\_color, int line\_width, float[] line\_style)

{

ResetCoefficients();

Point[] points = new Point[7];

for (int i = 0; i < 6; i++)

points[i] = new Point(Hex\_center.X + (int)(Hex\_radius \* (Math.Cos(Math.PI \* i / 3 + Angle))), Hex\_center.Y + (int)(Hex\_radius \* (Math.Sin(Math.PI \* i / 3 + Angle))));

points[6] = points[0];

Pen pen = new Pen(line\_color, line\_width);

Graphics gr = pictureBox1.CreateGraphics();

if (line\_style != null) pen.DashPattern = line\_style;

gr.DrawLines(pen, points);

Current\_hex\_line\_width = line\_width;

Current\_hex\_line\_style = line\_style;

Current\_hex\_radius = Hex\_radius;

Current\_hex\_center = Hex\_center;

Current\_angle = Angle;

}

private void DeleteHexagon()

{

Point[] points = new Point[7];

for (int i = 0; i < 6; i++)

points[i] = new Point(Current\_hex\_center.X + (int)(Current\_hex\_radius \* (Math.Cos(Math.PI \* i / 3 + Current\_angle))), Current\_hex\_center.Y + (int)(Current\_hex\_radius \* (Math.Sin(Math.PI \* i / 3 + Current\_angle))));

points[6] = points[0];

Pen pen = new Pen(pictureBox1.BackColor, Current\_hex\_line\_width);

Graphics gr = pictureBox1.CreateGraphics();

if (Current\_hex\_line\_style != null) pen.DashPattern = Current\_hex\_line\_style;

gr.DrawLines(pen, points);

if (Sin\_activated) DrawSin(Current\_line\_color, Current\_line\_width, Current\_line\_style);

}

private void button2\_Click(object sender, EventArgs e)

{

DeleteHexagon();

Angle = 0;

switch (comboBox7.SelectedIndex)

{

case 0:

Hex\_center = new Point(Left\_limit \* Scale\_x + Shift\_x, (int)(-F(Left\_limit) \* Scale\_y + Shift\_y));

break;

case 1:

Hex\_center = new Point(Right\_limit \* Scale\_x + Shift\_x, (int)(-F(Right\_limit) \* Scale\_y + Shift\_y));

break;

}

try

{

Hex\_radius = int.Parse(textBox1.Text);

if (Hex\_radius < 0) throw new FormatException();

}

catch (FormatException)

{

MessageBox.Show("Error: Invalid size\nCannot draw hexagon", "Error");

return;

}

DrawHexagon(Hex\_line\_color, Hex\_line\_width, Hex\_line\_style);

Hex\_activated = true;

}

Кнопка №3 отвечает за движение шестиугольника по траектории. После нажатия на нее считываются параметры, связанные с движением, после чего запускается цикл, в котором объект проходит заданное число циклов, при этом, если это задано, объект вращается и пульсирует с заданными параметрами:

private void button3\_Click(object sender, EventArgs e)

{

if (!Hex\_activated || !Sin\_activated) return;

bool OK = InitMovement();

if (!OK) return;

int radius = Hex\_radius;

bool grows = false;

int delay\_time = (int)(10000 / Speed \* Move\_step);

double base\_x = (Hex\_center.X - Shift\_x) / Scale\_x;

int init\_direction = (base\_x <= 0) ? 1 : -1;

int direction = init\_direction;

int right = Right\_limit \* Scale\_x + Shift\_x;

int left = Left\_limit \* Scale\_x + Shift\_x;

int rotation\_direction = Counterclockwise ? 1 : -1;

if (!Is\_rotating) rotation\_direction = 0;

int cycle\_count = 0;

while (cycle\_count < Cycles)

{

while (direction == 1 && Hex\_center.X < right || direction == -1 && Hex\_center.X > left)

{

DeleteHexagon();

if (grows)

{

Hex\_radius += (int)(Move\_step \* Pulsation\_speed);

if (Hex\_radius >= radius)

{

Hex\_radius = radius;

grows = false;

}

}

else

{

Hex\_radius -= (int)(Move\_step \* Pulsation\_speed);

if (Hex\_radius <= Min\_size)

{

Hex\_radius = Min\_size;

grows = true;

}

}

base\_x += Move\_step \* direction;

Hex\_center.X = (int)(base\_x \* Scale\_x) + Shift\_x;

Hex\_center.Y = (int)(-F(base\_x) \* Scale\_y) + Shift\_y;

Angle += Move\_step \* Rotation\_speed \* rotation\_direction;

DrawHexagon(Hex\_line\_color, Hex\_line\_width, Hex\_line\_style);

System.Threading.Thread.Sleep(delay\_time);

}

direction = -direction;

if (direction == init\_direction) cycle\_count++;

}

}

**2.5. Текст программы**

**заключение**

Кратко подвести итоги, проанализировать соответствие поставленной цели и полученного результата.

**список использованных источников**

***Ниже представлены примеры библиографического описания, В качестве названия источника в примерах приводится вариант, в котором применяется то или иное библиографическое описание.***

1. Иванов И. И. Книга одного-трех авторов. М.: Издательство, 2010. 000 с.

2. Книга четырех авторов / И. И. Иванов, П. П. Петров, С. С. Сидоров, В. В. Васильев. СПб.: Издательство, 2010. 000 с.

3. Книга пяти и более авторов / И. И. Иванов, П. П. Петров, С. С. Сидоров и др.. СПб.: Издательство, 2010. 000 с.

4. Описание книги под редакцией / под ред. И.И. Иванова СПб., Издательство, 2010. 000 с.

5. Иванов И.И. Описание учебного пособия и текста лекций: учеб. пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2010. 000 с.

6. Описание методических указаний / сост.: И.И. Иванов, П.П. Петров. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2010. 000 с.

7. Иванов И.И. Описание статьи с одним-тремя авторами из журнала // Название журнала. 2010, вып. (№) 00. С. 000–000.

8. Описание статьи с четырьмя и более авторами из журнала / И. И. Иванов, П. П. Петров, С. С. Сидоров и др. // Название журнала. 2010, вып. (№) 00. С. 000–000.

9. Иванов И.И. Описание тезисов доклада с одним-тремя авторами / Название конференции: тез. докл. III международной науч.-техн. конф., СПб, 00–00 янв. 2000 г. / СПбГЭТУ «ЛЭТИ», СПБ, 2010, С. 000–000.

10. Описание тезисов доклада с четырьмя и более авторами / И. И. Иванов, П. П. Петров, С. С. Сидоров и др. // Название конференции: тез. докл. III международной науч.-техн. конф., СПб, 00–00 янв. 2000 г. / СПбГЭТУ «ЛЭТИ», СПБ, 2010, С. 000–000.

11. Описание электронного ресурса // Наименование сайта. URL: http://east-front.narod.ru/memo/latchford.htm (дата обращения: 00.00.2010).

12. ГОСТ 0.0–00. Описание стандартов. М.: Изд-во стандартов, 2010.

13. Пат. RU 00000000. Описание патентных документов / И. И. Иванов, П. П. Петров, С. С. Сидоров. Опубл. 00.00.2010. Бюл. № 00.

14. Иванов И.И. Описание авторефератов диссертаций: автореф. дисс. канд. техн. наук / СПбГЭТУ «ЛЭТИ», СПБ, 2010.

15. Описание федерального закона: Федер. закон [принят Гос. Думой 00.00.2010] // Собрание законодательств РФ. 2010. № 00. Ст. 00. С. 000–000.

16. Описание федерального постановления: постановление Правительства Рос. Федерации от 00.00.2010 № 00000 // Опубликовавшее издание. 2010. № 0. С. 000–000.

17. Описание указа: указ Президента РФ от 00.00.2010 № 00 // Опубликовавшее издание. 2010. № 0. С. 000–000.

**приложение А**

**Название приложения**