МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Учреждение образования
«Витебский государственный технологический университет»

РАЗРАБОТКА МУЛЬТИМЕДИЙНЫХ ПРИЛОЖЕНИЙ. ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ C#

Методические указания
по выполнению лабораторных работ для студентов специальностей
1-19 01 01-06 «Дизайн виртуальной среды»,
6-05-0211-05 «Графический дизайн и мультимедиадизайн»

Витебск
2024

УДК 004.9
ББК 32.97
Р17

Составители:

Е. Б. Дунина, А. С. Соколова

Одобрено кафедрой «Информационные системы и технологии»
УО «ВГТУ», протокол № 12 от 18.04.2024.

Рекомендовано к изданию редакционно-издательским
советом УО «ВГТУ», протокол № 8 от 29.04.2024.

Разработка мультимедийных приложений. Основы программирования на языке C# : методические указания по выполнению лабораторных работ / УО «ВГТУ» ; сост.: Е. Б. Дунина, А. С. Соколова – Витебск: УО «ВГТУ», 2024. – 58 с.

В методических указаниях изложены теоретические сведения и задания к лабораторным работам по дисциплине «Разработка мультимедийных приложений». Издание предназначено для студентов специальностей 1-19 01 01-06 «Дизайн виртуальной среды», 6-05-0211-05 «Графический дизайн и мультимедиадизайн (профилизация: Мультимедиадизайн)»

УДК 004.9
ББК 32.97

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА 1. C#: ОСНОВЫ РАЗРАБОТКИ, ИЗУЧЕНИЕ БАЗОВЫХ АЛГОРИТМОВ

Цель работы: изучить принципы графического интерфейса приложений в Visual Studio.Net.

Краткие теоретические сведения

Средством взаимодействия с программой является графический пользовательский интерфейс (Graphical User Interface, GUI).

Windows Forms – это часть каркаса .NET Framework, которая поддерживает создание приложений со стандартным GUI на платформе Windows.

Форма – это экранный объект, обеспечивающий функциональность программы.

Процесс создания Windows-приложения состоит из двух основных этапов:

1. Визуальное проектирование, то есть задание внешнего облика приложения. Оно заключается в помещении на форму компонентов (элементов управления – controls) и задании их свойств и свойств самой формы.

2. Определение поведения приложения путем написания методов обработки событий.

Структура проекта:

1. Файл «Program.cs» (рис. 1.1).

Класс Program, метод Main – точка входа, запуск приложения: Application.Run(new Form1());

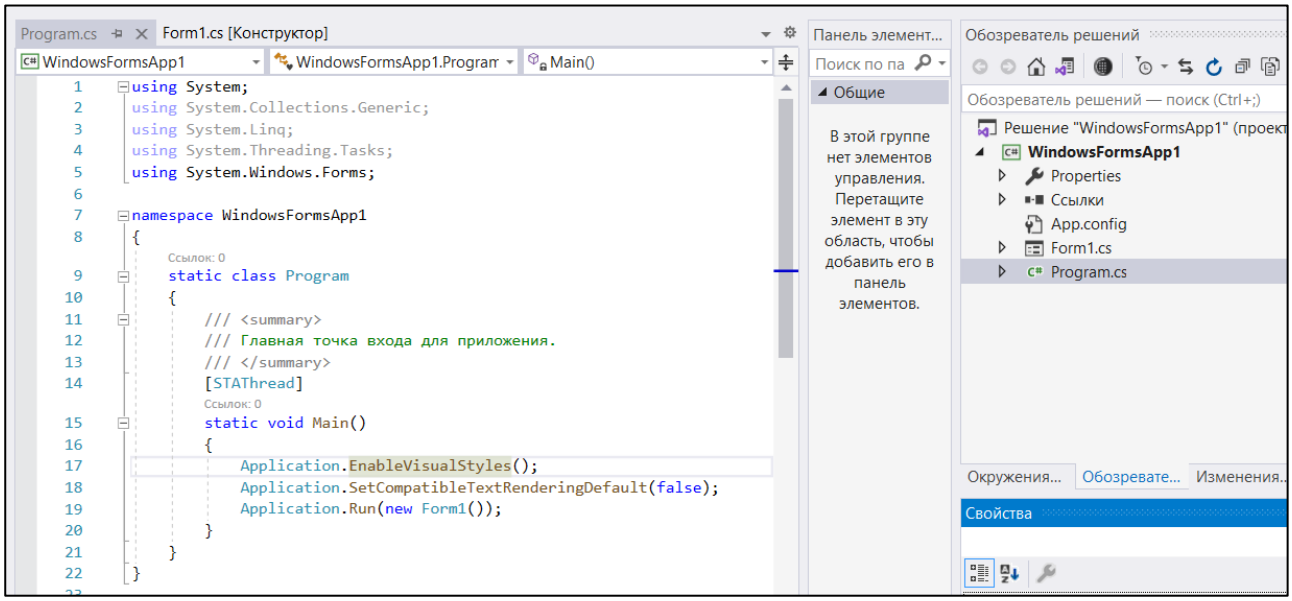

Рисунок 1.1 – Файл «Program.cs»

2. Файл «Form1.cs» (рис. 1.2).

В файле «Form1.cs» размещается часть класса Form1:

– конструктор с вызовом метода InitializeComponent();

– обработчики событий.

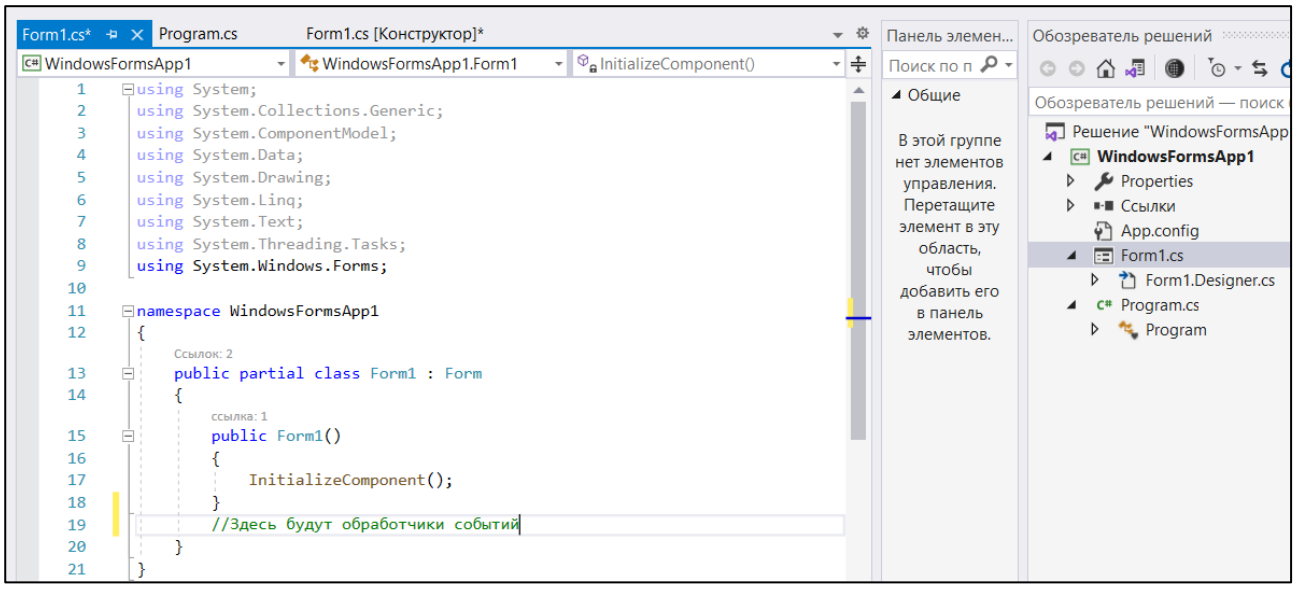

Рисунок 1.2 – Файл Form1.cs.

3. Файл «Form1.Designer.cs» (рисунок 1.3).

В области #region … #endregion размещается код метода InitializeComponent(), автоматически созданный конструктором форм Windows.

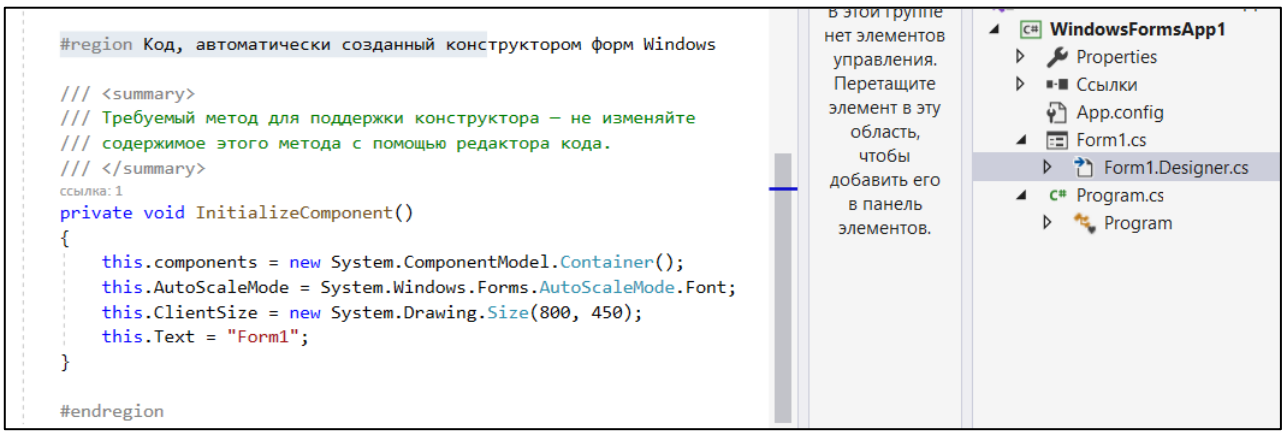

Рисунок 1.3 – Файл Form1.Designer.cs.

Размещение компонентов на форме:

– требуемый компонент перетаскивается с Панели элементов. При размещении компонента на форме автоматически создается экземпляр соответствующего класса и программный код;

– настраиваются свойства экземпляра в Окне свойств (имя, внешний вид, расположение, поведение).

Рассмотрим примеры приложений, демонстрирующих базовые алгоритмы разработки Windows-приложений.

Пример 1. Создание приложения, которое выводит на форму текст и изображение

Запустим MS Visual Studio.

Выбираем «Создать проект».

В появившемся окне выбераем язык C# и тип приложения «Windows Forms (.Net Framework)».

Далее в поле ввода «Расположение» указываем рабочую папку, в которой будут храниться файлы нашего проекта.

Введем имя проекта и решения «lab01». Откроется окно конструктора формы «Form1.cs [Конструктор]». Вызовем панель элементов «Toolbox» (меню «Вид» → «Панель элементов»). Размещаем на форме: две надписи «Label», поле ввода текста «TextBox» и кнопку «Button».

В окне свойств «Properties» изменим задаваемые по умолчанию свойства «text» для элементов: для формы – на «Моя первая программа», для первой надписи – на «Как Вас зовут?», для кнопки – на «GO». Для второй надписи значение свойства «BackColor» установим цвет, отличный от цвета формы (рис. 1.4).

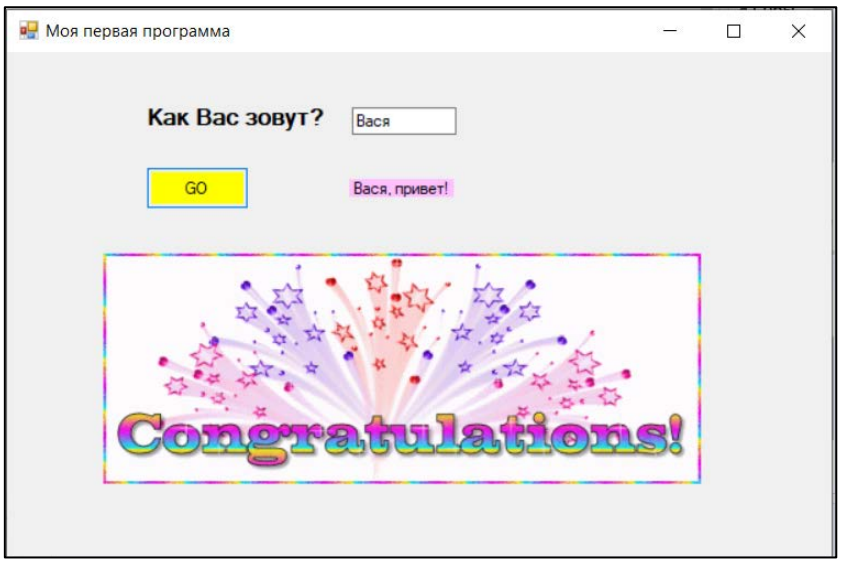

Рисунок 1.4 – Форма «Моя первая программа»

Создадим обработчик события «Click» кнопки. Для этого выполним двойной щелчок по кнопке (или выберем на вкладке «События» окна свойств «Click»). Откроется окно нового модуля «Form1.cs», в котором создаются обработчики событий компонентов формы Form1. Автоматически будет создан шаблон для ввода кода (рис. 1.5).

```
private void button1_Click(object sender, EventArgs e)
{

}
```

Рисунок 1.5 – Обработчик события Click кнопки

Введем в созданный шаблон код:

```
label2.Text = textBox1.Text + ", привет!";
```

Теперь протестируем программу (нажав F5).

Модифицируем проект. Разместим на форме элемент pictureBox (рис. 1.4). Импортируем изображение (свойство «Image») из файла «congratulations.gif». Подберем размеры и установим для свойства «SizeMode» значение «StretchImage», для «Visible» – «false». В обработчик события добавим строку:

```
pictureBox1.Visible = true;
```

Задание: самостоятельно модифицировать проект, чтобы по щелчку по картинке выполнялся выход из программы (закрыть программу – `this.Close();`).

Пример 2. Создание простейшего теста с использованием radioButton

Создадим новый проект «lab02».

Разместим на форме надпись «Label» с вопросом «Изображение формируется как матрица пикселей в», а также разместим элементы «radioButton» с ответами: «векторной графике», «растровой графике», «фрактальной графике», кнопку «Button» с надписью «Ответ» и элемент «TextBox» для вывода результата (рис. 1.6).

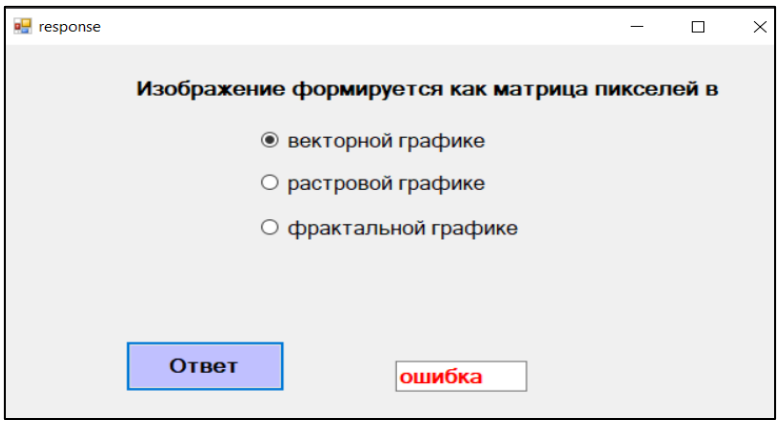

Рисунок 1.6 – Пример использования «radioButton»

Свойству «Checked» элемента «radioButton1» зададим значение «True» (по умолчанию будет выделен первый переключатель).

Создадим обработчик события «Click» кнопки и введем в него код:

```
textBox1.Text = (radioButton2.Checked) ? "верно" : "ошибка";
```

Пример 3. Создание приложения с использованием ComboBox и оператора выбора

Создадим новый проект «lab03».

Разместите на форме: надпись «Label», поле со списком «ComboBox» и кнопку «Button». В окне свойств «Properties» изменим задаваемые по умолчанию свойства «text»: для формы – на «Первая пара», для кнопки – на «Вывести» (рис. 1.7).

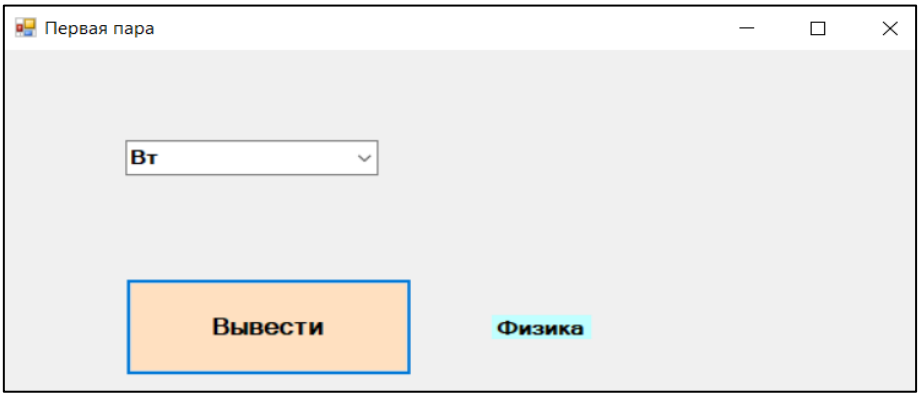

Рисунок 1.7 – Пример использования ComboBox

В поле «Item» компонента «ComboBox» введем названия дней недели: Пн, Вт, … Вс. Создадим обработчик события «Click» кнопки и введем в него код , представленный в листинге 1.1.

Листинг 1.1 – Пример работы оператора выбора

```
string day; //переменная строкового типа
day = comboBox1.SelectedItem.ToString(); //выбранный элемент преобразуем в
строку
switch (day)
{
    case "Пн": label1.Text = "Иностранный язык"; break;
    case "Вт": label1.Text = "Физика"; break;
    case "Ср": label1.Text = "Нет первой пары"; break;
    case "Чт": label1.Text = "Системы компьютерной графики"; break;
    case "Пт": label1.Text = "Выходной"; break;
    default: label1.Text = "Выходной"; break;
}
```

Перейдем в конструктор формы, выделим форму, выберем на вкладке «События» окна свойств событие «Load». В обработчике события напишем код:

```
comboBox1.SelectedIndex = 0;
```

Протестируем программу (нажав F5).

Пример 4. Расчет площади круга. Использование элемента trackBar

Создадим проект «lab04».

Разместим на форме надписи: «R = », «S = », два текстовых поля и элемент «trackBar». Установим свойству «Maximum» значение 80, «Minimum» – значение 10.

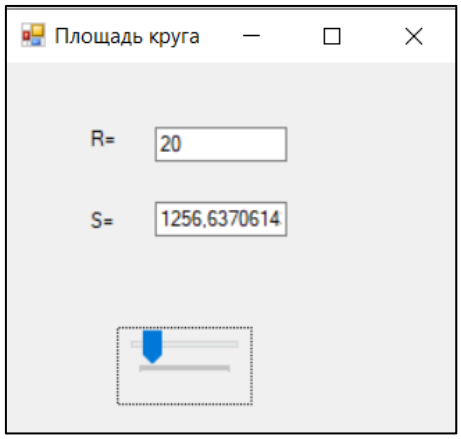

Рисунок 1.8 – Пример использования «trackBar»

Создадим обработчик события «Scroll» (листинг 1.2).

Листинг 1.2 – Обработчик события Scroll

```
private void trackBar1_Scroll(object sender, EventArgs e)
{
    int R = trackBar1.Value;
    textBox1.Text = Convert.ToString(R);
    textBox2.Text = Convert.ToString(Math.PI * R * R);
}
```

Пример 5. Вычисление суммы, разности и произведения двух чисел

Создадим проект «lab05».

Разместим на форме надписи: «a = », «b = », «сумма двух чисел», «разность двух чисел», «произведение двух чисел», два текстовых поля ввода чисел. Вывод будем осуществлять в три надписи по нажатию кнопки (рис. 1.9).

Создадим обработчик события «Click» кнопки и вводим в него код (листинг 1.3).

Листинг 1.3 – Обработчик события «button1»

```
private void button1_Click(object sender, EventArgs e)
{
    double a = Convert.ToDouble(textBox1.Text);
    double b = Convert.ToDouble(textBox2.Text);
    label6.Text = Convert.ToString(a + b);
    label7.Text = Convert.ToString(a - b);
    label8.Text = Convert.ToString(a * b);
}
```

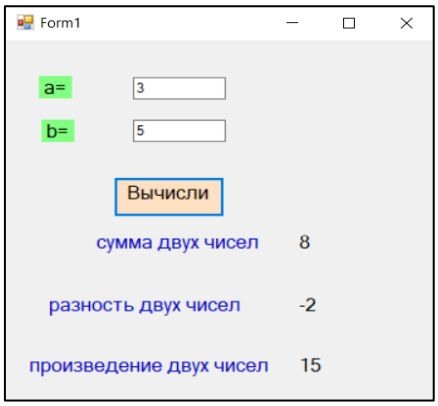

Рисунок 1.9 – Пример выполнения арифметических операций

Порядок выполнения работы

1. Реализовать примеры 1–5, описанные в краткой теории.
2. Создать windows-приложение так, чтобы в нём разместились поля для задания (радиус основания и высота цилиндра) и отображения характеристик геометрического тела – цилиндра (площадь основания, объем цилиндра, площадь полной поверхности).
3. Создать windows-приложение так, чтобы в нём разместились поля для задания сторон прямоугольного треугольника (a, b – катеты, c – гипотенуза; должна выполняться теорема Пифагора) и отображения площади и периметра треугольника.
4. Создать windows-приложение так, чтобы в нём разместились поля для задания радиуса двух концентрических окружностей R1 и R2 и отображения площади кольца.
5. Создать windows-приложение для расчета площади квадрата с использованием элемента trackBar.
6. Создать windows-приложение для расчета скалярного произведения двух векторов. Вводятся координаты двух векторов. Скалярное произведение вычисляется по формуле: $\vec{a}\vec{b} = x1 \cdot x2 + y1 \cdot y2 + z1 \cdot z2$.
7. Создать windows-приложение так, чтобы в нём разместились поля для задания радиуса шара и отображения площади его поверхности.
8. Создать windows-приложение так, чтобы в нём разместились поля для задания радиуса шара и отображения его объема.
9. Создать windows-приложение для расчета угла треугольника по известным двум углам.
10. Создать windows-приложение так, чтобы в нём разместились поля для задания двух сторон основания и высоты прямоугольного параллелепипеда и отображения его объема.

Содержание отчета

Предоставить программы, написанные в Visual Studio на языке C# и результаты их работы. При реализации заданий в одном решении можно размещать несколько проектов.

Контрольные вопросы

1. Напишите код, который считывает введенное расстояние в км и выводит ответ в метрах.
2. Как и для чего нужна конструкция using?
3. Какие модификаторы доступа в C# Вы знаете.
4. Какой компонент предназначен для отображения текстовой информации?
5. Какой компонент предназначен для ввода данных с клавиатуры?
6. В каком случае используют компонент RadioButton?
7. Назовите основные структурные элементы проекта.

ЛАБОРАТОРНАЯ РАБОТА 2. КАЛЬКУЛЯТОР НА C#

Цель работы: изучить модель обработки событий в языке C#.

Краткие теоретические сведения

Функциональность windows-приложения реализуется на основе объектно-событийной модели. Определение поведения объектов начинается с принятия решений, какие действия должны выполняться при щелчке на кнопках, вводе текста, выборе пунктов меню и т. д. (по каким событиям будут выполняться действия, реализующие функциональность программы).

Для каждого класса определен свой набор событий, на которые он может реагировать. Требуемое событие для выбранного объекта регистрируется в методе InitializeComponent() (файл Form1.Designer.cs) или непосредственно в конструкторе формы (файл Form1.cs)

Регистрацию события удобно выполнять в окне свойств на вкладке Events двойным щелчком на поле, расположенном справа от имени соответствующего события (рис. 2.1).

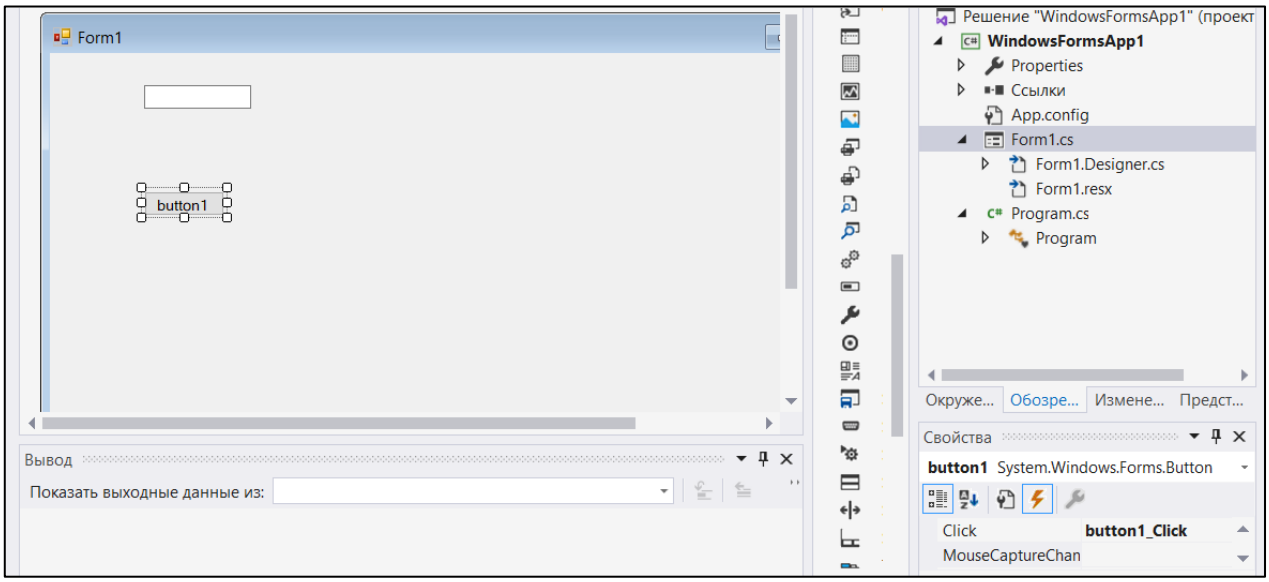

Рисунок 2.1 – Регистрация события нажатия кнопки

При этом в методе InitializeComponent() (файл Form1.Designer.cs) появляется строка

```
this.button1.Click += new System.EventHandler(this.button1_Click);
```

В файле кода формы Form1.cs автоматически создается шаблон соответствующего метода – обработчика

```
private void button1_Click(object sender, EventArgs e) {}
```

Чтобы управлять внешним отображением кнопки, можно использовать свойство FlatStyle. Оно может принимать следующие значения:

– Flat – кнопка имеет плоский вид;

– Popup – кнопка приобретает объемный вид при наведении на нее указателя, в иных случаях она имеет плоский вид;

– Standard – кнопка имеет объемный вид (используется по умолчанию);

– System – вид кнопки зависит от операционной системы.

Порядок выполнения работы

1. Создадим новый проект. Пользовательский интерфейс представлен на рисунке 2.2.

Основные компоненты программы: текстовое поле и 25 кнопок:

button1 – button11 – кнопки для ввода цифр 1, 2…9, 0 и 00;

button12 – button13 – кнопки «C» и «<–»;

button14 – button17 – кнопки для арифметических действий (сложения, вычитания, умножения и деления);

button18 – button19 – кнопки «=» и «,»;

button20 – кнопка для возведения в квадрат;

button21 – button24 – кнопки для вычисления тригонометрических функций (sin, cos, tan, ctan);

button25 – кнопка для извлечения квадратного корня.

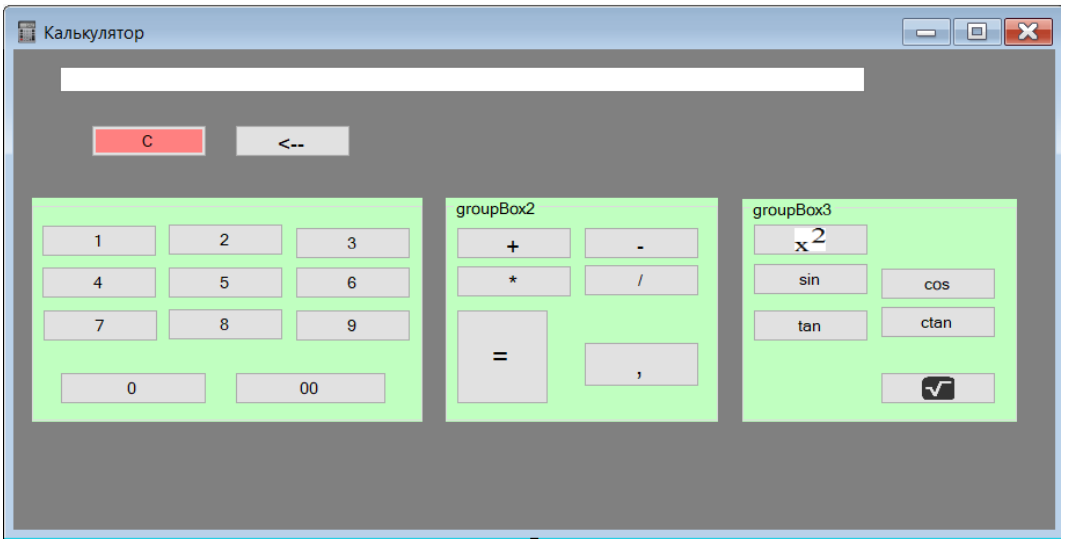

Рисунок 2.2 – Интерфейс калькулятора

Для оформления используем «groupBox». Что бы вставить иконку квадратного корня с расширением ICO, необходимо воспользоваться свойством «BackgroundImage», а затем «BackgroundImageLayout» → «Zoom».

Для каждой кнопки «button1» в свойстве «Tag» пропишем «1», для «button2» – «2» и т. д. Для кнопки «button10» пропишем «0».

Выделяем все кнопки и добавляем событие «Click». Появится функция:

```
        private void button10_Click_1(object sender, EventArgs e)
    {
        int position = Convert.ToInt32(((Button)sender).Tag);
        textBox1.Text += position;
    }
```

Информация, какая кнопка нажата, содержится в object sender.

2. Проанализировать код и создать соответствующие обработчики событий (листинг 2.1).

Листинг 2.1 – Class Form1

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    public double a, b;
    public char c;

    private void button11_Click(object sender, EventArgs e)
    {
        textBox1.Text += "00";
    }

    private void button14_Click(object sender, EventArgs e)
    {
        try
        {
            a = System.Convert.ToDouble(textBox1.Text);
            c = '+';
            textBox1.Text = "";
        }
        catch (Exception)
        {
            MessageBox.Show("Invalid input");
        }
    }

    private void button15_Click(object sender, EventArgs e)
    {
        try
        {
            a = System.Convert.ToDouble(textBox1.Text);
            c = '-';
            textBox1.Text = "";
```

```
    }
    catch (Exception)
    {
        MessageBox.Show("Invalid input");
    }
}

private void button16_Click(object sender, EventArgs e)
{
    try
    {
        a = System.Convert.ToDouble(textBox1.Text);
        c = '*';
        textBox1.Text = "";
    }
    catch (Exception)
    {
        MessageBox.Show("Invalid input");
    }
}

private void button17_Click(object sender, EventArgs e)
{
    try
    {
        a = System.Convert.ToDouble(textBox1.Text);
        c = '/';
        textBox1.Text = "";
    }
    catch (Exception)
    {
        MessageBox.Show("Invalid input");
    }
}

private void button18_Click(object sender, EventArgs e)
{
    b = Convert.ToDouble(textBox1.Text);
    textBox1.Text = "";
    switch (c)
    {
        case '+':
            textBox1.Text = Convert.ToString(a + b); break;
        case '-':
            textBox1.Text = Convert.ToString(a - b); break;
        case '*':
            textBox1.Text = Convert.ToString(a * b); break;
        case '/':
            textBox1.Text = Convert.ToString(a / b); break;
    }
}

private void button19_Click(object sender, EventArgs e)
{
    if (!textBox1.Text.Contains(","))
        textBox1.Text += ",";
```

```
    }

    private void button20_Click(object sender, EventArgs e)
    {
        a = Convert.ToDouble(textBox1.Text);
        textBox1.Text = Convert.ToString(Math.Pow(a, 2));
    }

    private void button21_Click(object sender, EventArgs e)
    {
        a = Convert.ToDouble(textBox1.Text);
        textBox1.Text = Convert.ToString(Math.Sin((a * Math.PI) / 180));
    }

    private void button22_Click(object sender, EventArgs e)
    {
        a = Convert.ToDouble(textBox1.Text);
        textBox1.Text = Convert.ToString(Math.Cos((a * Math.PI) / 180));
    }

    private void button23_Click(object sender, EventArgs e)
    {
        a = Convert.ToDouble(textBox1.Text);
        textBox1.Text = Convert.ToString(Math.Tan((a * Math.PI) / 180));
    }

    private void button24_Click(object sender, EventArgs e)
    {
        a = Convert.ToDouble(textBox1.Text);
        textBox1.Text = Convert.ToString(Math.Cos((a * Math.PI) / 180) /
(Math.Sin((a * Math.PI) / 180)));
    }

    private void button12_Click(object sender, EventArgs e)
    {
        textBox1.Text = "";
        a = 0; b = 0;
    }

    private void button13_Click(object sender, EventArgs e)
    {
        string s = textBox1.Text;
        if (s.Length != 0)
        {
            s = s.Substring(0, s.Length - 1);
            textBox1.Text = s;
        }
    }

    private void button25_Click(object sender, EventArgs e)
    {
        a = Convert.ToDouble(textBox1.Text);
        textBox1.Text = Convert.ToString(Math.Sqrt(a));
    }

    private void button10_Click_1(object sender, EventArgs e)
```

```
    {
        int position = Convert.ToInt32(((Button)sender).Tag);
        textBox1.Text += position;
    }
}
```

3. Протестировать программу.

Содержание отчета

Предоставить приложение «Калькулятор», написанное в Visual Studio на языке C# и шаблонах Windows Forms. Продемонстрировать работу приложения.

Контрольные вопросы

1. Как определить номер кнопки, используя свойство Tag?
2. Для чего используется панель элементов?
3. Какие стандартные элементы управления Вы знаете?
4. Как создать обработчик события нажатия кнопки?
5. Разместите на форме четыре кнопки (Button). Сделайте на кнопках следующие надписи: красный, зеленый, синий, желтый. Создайте четыре обработчика события нажатия на данные кнопки, которые буду менять цвет формы в соответствии с текстом на кнопках.

ЛАБОРАТОРНАЯ РАБОТА 3. СЕКУНДОМЕР

Цель работы: научиться использовать таймеры в приложениях на C#.

Краткие теоретические сведения

Компонент «Timer» генерирует последовательность событий «Tick». Обычно он используется для активизации с заданным периодом некоторых действий. Компонент является невизуальным (во время работы программы в окне не отображается). Свойства компонента приведены в таблице 3.1.

Таблица 3.1 – Свойства компонента «Timer»

Свойство	Описание
Interval	Период генерации события «Tick». Задается в миллисекундах
Enabled	Разрешение работы. Разрешает (значение «True») или запрещает (значение «false») генерацию события «Tick»

Мы можем управлять программами с помощью таймера в миллисекундах, секундах, минутах и даже в часах. Элемент управления таймером позволяет нам установить свойство интервала в миллисекундах. То есть одна секунда равна 1000 миллисекундам. Например, если мы хотим установить интервал в 1 минуту, мы устанавливаем значение свойства Interval равным 60000 $(60 \cdot 1000 = 60000)$.

По умолчанию свойство управления таймером имеет значение False. Поэтому перед запуском программы мы должны установить свойство Enabled равным True, тогда только управление таймером начнет свою функцию. Для чего правой кнопкой мыши щелкаем на элементе и во вкладке «Свойства» выбираем нужные пункты (рис. 3.1).

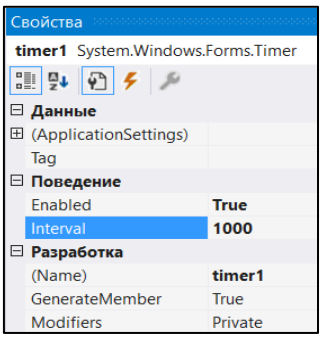

Рисунок 3.1 – Свойства Timer

Пример. Отобразим текущее время в элементе lable. Для этого на форму перетащим элемент Timer и lable. Двойной щелчок по элементу таймера создаст метод обработчик, внутри которого мы и будем присваивать изменяющееся значения таймера метке на lable

```
private void timer1_Tick(object sender, EventArgs e)
                        {
label1.Text = DateTime.Now.ToString();
                        }
```

Таким образом, передали метке lable изменяющееся значение элемента DateTime (рис. 3.2).

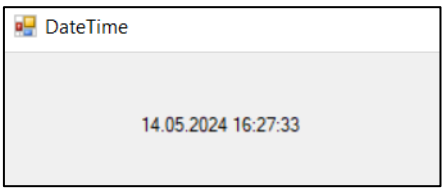

Рисунок 3.2 – Значение элемента DateTime

Порядок выполнения работы

1. Пользовательский интерфейс представлен на рисунке 3.3.

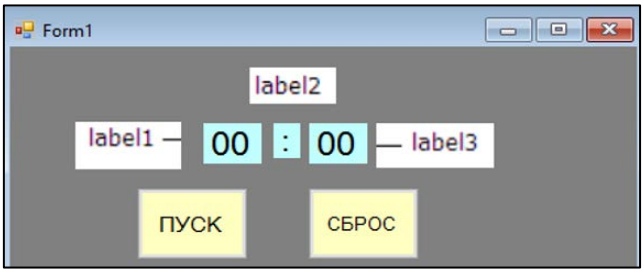

Рисунок 3.3 – Интерфейс секундомера

Во время создания формы свойству «Interval» таймера необходимо присвоить значение 500. Кнопка «button1» предназначена как для запуска секундомера, так и для его остановки. В начале работы программы значение свойства «Enabled» компонента «Timer» равно «False», поэтому таймер не работает (не генерирует события «Tick»). Процедура обработки события «Click» на кнопке «button1» проверяет состояние таймера и, если он не работает, присваивает значение «True» свойству «Enabled» компонента «Timer» и тем самым запускает таймер. Процедура обработки события «Tick» инвертирует значение свойства «Visible» компонента «label2» (в результате двоеточие мигает с периодом 0,5 секунд) и, если двоеточие скрыто, завершает свою работу. Если двоеточие отображается, то она увеличивает счетчик времени и выводит в поле компонентов «label1» и «label3» значения счетчиков минут и секунд (величину интервала с момента запуска секундомера). Если секундомер работает, то щелчок на кнопке «Стоп» («button1») останавливает секундомер.

2. Проанализировать код и создать соответствующие обработчики событий (листинг 3.1).

Листинг 3.1 – Код реализации секундомера

```
public partial class Form1 : Form
{
    int m, s;
    public Form1()
    {
        InitializeComponent();
        timer1.Interval = 500;
        // обнуление показаний
        m = 0; s = 0;
        label1.Text = "00";
        label3.Text = "00";
        label2.Visible = true;
    }

    private void button1_Click(object sender, EventArgs e)
    {
        if (timer1.Enabled)
        {
            // таймер работает, остановить таймер
            timer1.Enabled = false;
            // изменить текст на кнопке и сделать ее доступной
            button1.Text = "Пуск";
            button2.Enabled = true;
        }
        else
        {
            // таймер не работает, запускаем таймер
            timer1.Enabled = true;
            // изменить текст на кнопке и сделать ее недоступной
            button1.Text = "Стоп";
            button2.Enabled = false;
        }
```

```
    }

    private void button2_Click(object sender, EventArgs e)
    {
        m = 0; s = 0;
        label1.Text = "00";
        label3.Text = "00";
    }

    private void timer1_Tick_1(object sender, EventArgs e)
    {
        // двоеточие мигает с периодом 0.5 сек
        if (label2.Visible)
        {
            if (s < 59)
            {
                s++;
                if (s < 10)
                    label3.Text = "0" + s.ToString();
                else
                    label3.Text = s.ToString();
            }
            else
            {
                if (m < 59)
                {
                    m++;
                    if (m < 10)
                        label1.Text = "0" + m.ToString();
                    else
                        label1.Text = m.ToString();
                    s = 0;
                    label3.Text = "00";
                }
                else
                {
                    m = 0;
                    label1.Text = "00";
                }
            }
            label2.Visible = false;
        }
        else
            label2.Visible = true;
    }
}
```

3. Протестировать программу.

Содержание отчета

Предоставить приложение «Секундомер», написанное в Visual Studio на языке C# и шаблонах Windows Forms. Продемонстрировать работу приложения.

Контрольные вопросы

1. Что означает «компонент Timer является невизуальным»?
2. Для чего служит компонент Timer?
3. Какие свойства компонента Timer Вы знаете?
4. Какое событие можно создать для Timer?
5. Как отобразить на форме текущую дату и время.

ЛАБОРАТОРНАЯ РАБОТА 4. ОСНОВЫ РАБОТЫ С ГРАФИКОЙ

Цель работы: разработать приложения с графическим интерфейсом (GUI) в среде визуального программирования (Visual Studio).

Краткие теоретические сведения

Отображение графики обеспечивает компонент «PictureBox». Графическая поверхность «PictureBox» представляет собой объект «Graphics», методы которого обеспечивают вывод графики. Чтобы на поверхности компонента «PictureBox» появилась линия, прямоугольник, окружность или загруженная из файла иллюстрация, необходимо вызвать соответствующий метод объекта «Graphics». Доступ к графической поверхности объекта есть только у функции обработки события «Paint». Событие «Paint» возникает в начале работы программы, когда окно, а, следовательно, и все компоненты появляются на экране.

Чтобы создать графическую поверхность компонента «pictureBox» необходимо написать следующий код и добавить пространство имен «System.Drawing»:

```
Graphics g = pictureBox1.CreateGraphics();
```

Карандаш (объект «Pen») определяет вид линии, кисть («Brush») – вид закраски области.

Создание карандаша (пера):

```
Pen pn = new Pen(Color.Red,4); //перо красного цвета толщиной 4 пикселя
```

Если толщина не указана, создается перо толщиной в 1 пиксель.

Создание кисти:

```
Brush br = new SolidBrush(Color.Green); //сплошная кисть зеленого цвета
```

Существуют и другие виды кистей: штриховая («HatchBrush»), текстурная («TextureBrush»), градиентная («LinearGradientBrush»).

Цвета задаются с помощью структуры «System.Drawing.Color». Например:

```
Color clr = Color.Orange;
```

Цвет также можно задать в модели αRGB: альфа (прозрачность), красный, зеленый, синий (от 0 до 255), например:

```
Color clr=Color.FromArgb(255,0,255,0); //непрозрачный зеленый
```

Если необходимо задать полностью непрозрачный цвет, параметр альфа можно не указывать.

Для стирания нарисованных элементов удобно использовать системный цвет «SystemColors.Control» – цвет формы, например:

```
g.Clear(SystemColors.Control);
```

Для вычисления графических примитивов используются методы, приведенные в таблице 4.1.

Таблица 4.1 – Методы для вычисления графических примитивов

Метод	Действие
DrawLine(Pen, x1, y1, x2, y2) DrawLine(Pen, p1, p2)	Рисует линию. Параметр «Pen» определяет цвет, толщину и стиль линии; параметры x1, y1, x2, y2 или p1 и p2 – координаты точек начала и конца линии
DrawRectangle(Pen, x, y, w, h)	Рисует контур прямоугольника. Параметр «Pen» определяет цвет, толщину и стиль границы прямоугольника: параметры x, y – координаты левого верхнего угла; параметры w и h задают размер прямоугольника
FillRectangle(Brush, x, y, w, h)	Рисует закрашенный прямоугольник. Параметр Brush определяет цвет и стиль закраски прямоугольника; параметры x, y – координаты левого верхнего угла; параметры w и h задают размер прямоугольника
DrawEllipse(Pen, x, y, w, h)	Рисует эллипс (контур). Параметр Pen определяет цвет, толщину и стиль линии эллипса; параметры x, y, w, h – координаты левого верхнего угла и размер прямоугольника, внутри которого вычерчивается эллипс
FillEllipse(Brush, x, y, w, h)	Рисует закрашенный эллипс. Параметр Brush определяет цвет и стиль закраски внутренней области эллипса; параметры x, y, w, h – координаты левого верхнего угла и размер прямоугольника, внутри которого вычерчивается эллипс
DrawPolygon(Pen, P)	Рисует контур многоугольника. Параметр Pen определяет цвет, толщину и стиль линии границы многоугольника; параметр P (массив типа Point) – координаты углов многоугольника
FillPolygon(Brush, P)	Рисует закрашенный многоугольник. Параметр Brush определяет цвет и стиль закраски внутренней области многоугольника; параметр P (массив типа Point) – координаты углов многоугольника
DrawString(str, Font, Brush, x, y)	Выводит на графическую поверхность строку текста. Параметр Font определяет шрифт; Brush – цвет символов; x и y – точку, от которой будет выведен текст
DrawImage(Image, x, y)	Выводит на графическую поверхность иллюстрацию. Параметр Image определяет иллюстрацию; x и y – координату левого верхнего угла области вывода иллюстрации

В качестве параметров методов вычерчивания графических примитивов часто используется структура «Point». Ее поля X и Y определяют положение (координаты) точки графической поверхности. Например:

```
Point p1 = new Point(10,10);
Point p2 = new Point(100,10);
// рисуем линию из p1 в p2
g.Graphics.DrawLine(Pens.Green, p1, p2);
```

Порядок выполнения работы

1. Графические примитивы.

1.1 Создать новый проект. Разместить на форме поле со списком «ListBox». В пункте «Items» окна свойств ввести список графических примитивов: Line Rectangle, FillRectangle, Ellipse, FillEllipse, Pie, FillPie (вводятся столбцом без запятых).

Затем создать обработчик события выбора из списка (листинг 4.1).

Листинг 4.1 – Обработчик события выбора из списка

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    Graphics g = CreateGraphics();
    Pen pn = new Pen(Color.Orange, 5);
    Brush br = new SolidBrush(Color.Green);
    g.Clear(SystemColors.Control);
    switch (listBox1.SelectedIndex)
    {
        case 0: g.DrawLine(pn, 150, 40, 350, 180); break;
        case 1: g.DrawRectangle(pn, 150, 30, 250, 150); break;
        case 2: g.FillRectangle(br, 150, 30, 250, 150); break;
        case 3: g.DrawEllipse(pn, 150, 30, 250, 150); break;
        case 4: g.FillEllipse(br, 150, 30, 250, 150); break;
        case 5: g.DrawPie(pn, 150, 30, 200, 200, 180, 225); break;
        case 6: g.FillPie(br, 150, 30, 150, 150, 0, 45); break;
    }
}
```

Протестировать программу.

1.2 Модифицировать проект. Добавить на форму кнопку выбора цвета пера «Button» и компонент «ColorDialog». Создайте обработчик события клика по кнопке (листинг 4.2).

Листинг 4.2 – Обработчик события кнопки выбора цвета пера

```
private void button1_Click(object sender, EventArgs e)
{
    colorDialog1.ShowDialog();
}
```

Измените строку кода создания пера:

```
Pen pn = new Pen(colorDialog1.Color, 5);
```

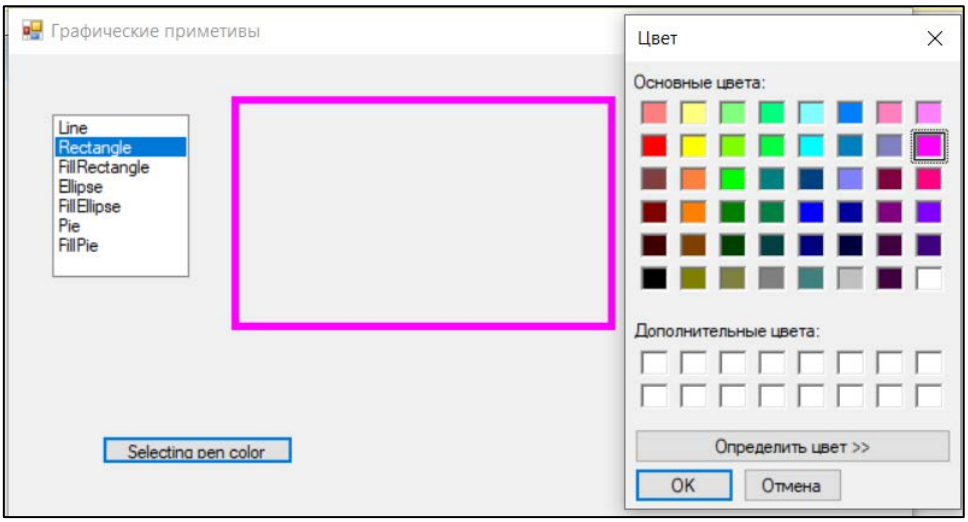

Рисунок 4.1 – Выбор цвета пера

1.3 Добавить обработчик события кнопки очистки формы (листинг 4.3).

Листинг 4.3 – Обработчик события кнопки очистки формы

```
private void button3_Click(object sender, EventArgs e)
{
    Graphics g = CreateGraphics();
    g.Clear(SystemColors.Control);
}
```

1.4 Самостоятельно модифицировать проект, добавив кнопки выбора цвета кисти.

2. Рисование мышью.

2.1 Создать новый проект. Разместить на форме «PictureBox». Свойство «BorderStyle» установить «Fixed3D». Добавить на форму кнопки «Стереть› и «Выбрать цвет».

2.2 Проанализировать код и создать соответствующие обработчики событий (листинг 4.4).

Листинг 4.4 – Код для создания рисования мышью

```
public partial class Form1 : Form
{
    bool ris = false;
    Color col = Color.Red;
    int w = 4;
    public Form1()
    {
        InitializeComponent();
    }

    private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
    {
        ris = true;
    }
```

```
private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
{
    ris = false;
}

private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
{
    this.Text = "x=" + e.X + "y=" + e.Y;
    if (ris)
    {
        Graphics g = pictureBox1.CreateGraphics();
        w = trackBar1.Value;
        switch (listBox1.SelectedIndex)
        {
            case 0:
                g.FillRectangle(new SolidBrush(col), e.X, e.Y, w, w);
                break;
            case 1:
                g.FillEllipse(new SolidBrush(col), e.X, e.Y, w, w);
                break;
        }
    }
}

private void button2_Click(object sender, EventArgs e)
{
    colorDialog1.ShowDialog();
    col = colorDialog1.Color;
}
```

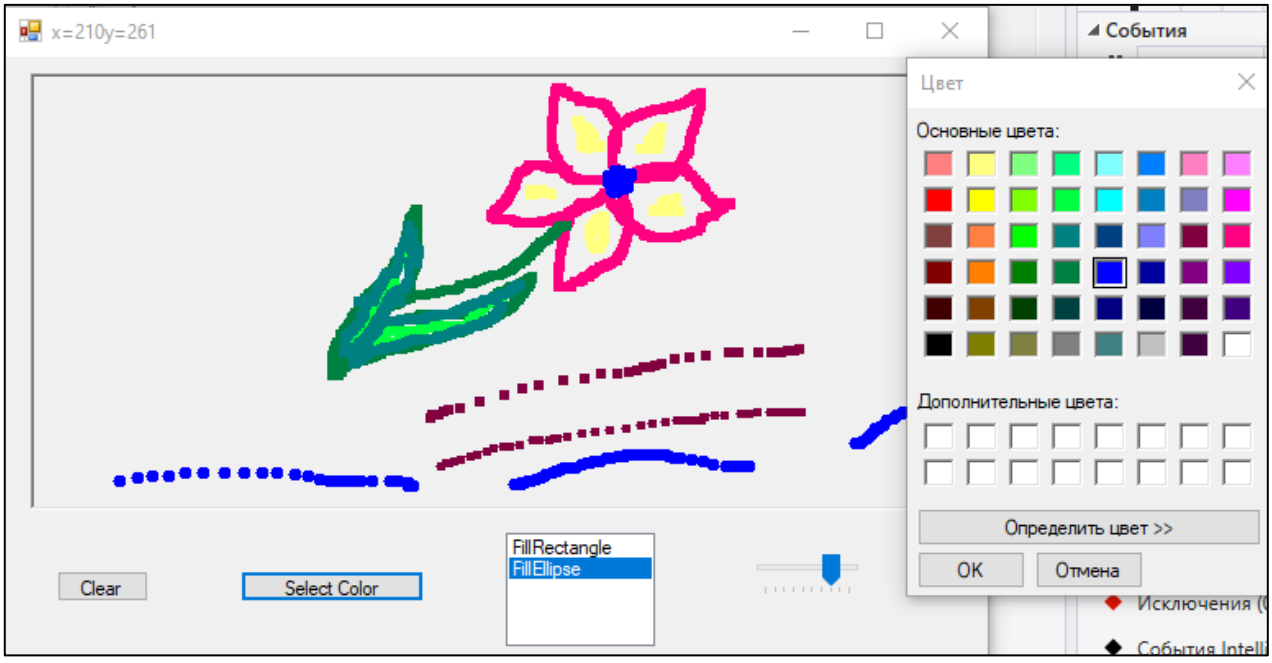

Рисунок 4.2 – Рисование мышью

2.3 Добавьте обработчик события кнопки очистки формы. Протестируйте программу.

Содержание отчета

Предоставить программы, написанные в Visual Studio на языке C# и результаты их работы. При реализации заданий в одном решении можно размещать несколько проектов.

Контрольные вопросы

1. Для чего служит компонент PictureBox?
2. Какие свойства компонента PictureBox знаете?
3. Какой метод используется для рисования линии?
4. Отличие методов DrawRectangle и FillRectangle.
5. Как создать обработчик события кнопки очистки формы?

ЛАБОРАТОРНАЯ РАБОТА 5. ДВИЖЕНИЕ ФИГУРЫ С ОТТАЛКИВАНИЕМ ОТ ГРАНИЦ В ПЛОСКОСТИ

Цель работы: написать графическое приложение, эмулирующее движение базовой фигуры таким образом, чтобы ее центр описывал заданную траекторию. При движении вдоль траектории фигура одновременно вращается вокруг центральной точки, а также меняет цвет. Скорость движения фигуры вдоль траектории и скорость ее вращения задаются пользователем.

Краткие теоретические сведения

Требуется отрисовать в левом нижнем углу «PictureBox» базовую фигуру, изображенную на рисунке 5.1. Исходный графический объект представляет собой правильный шестиугольник, в который вписан правильный треугольник. Расстояние между противоположными вершинами шестиугольника равно *a*.

Базовая фигура задается с помощью математических формул, которые описывают данные объекты, и методов задания объектов языка программирования C#.

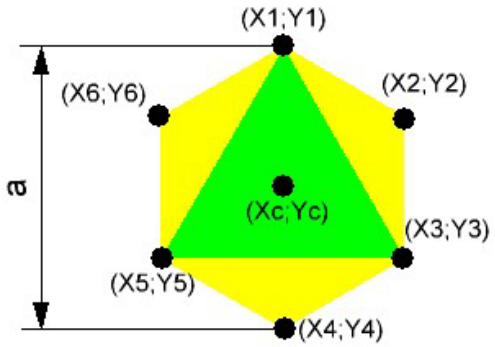

Рисунок 5.1 – Исходный графический объект

Для этого обозначим точки фигуры и выразим их координаты через координаты центральной точки (x_C, y_C) и определяющий размер фигуры *a*:

$$x_1 = x_C;\qquad y_1 = y_C - a/2;$$
$$x_2 = x_C + (a/2)\cos 30^\circ;\qquad y_2 = y_C - (a/2)\sin 30^\circ;$$
$$x_3 = x_C + (a/2)\cos 30^\circ;\qquad y_3 = y_C + (a/2)\sin 30^\circ;$$
$$x_4 = x_C;\qquad y_4 = y_C + a/2;$$
$$x_5 = x_C - (a/2)\cos 30^\circ;\qquad y_5 = y_C + (a/2)\sin 30^\circ;$$
$$x_6 = x_C - (a/2)\cos 30^\circ;\qquad y_6 = y_C - (a/2)\sin 30^\circ.$$

Отображение графики обеспечивает компонент PictureBox. Графическая поверхность компонента PictureBox представляет собой объект Graphics, методы которого и обеспечивают вывод графики. Чтобы создать графическую поверхность компонента PictureBox1 необходимо написать следующий код, обязательно добавив пространство имен System.Drawing:

```
Graphics g = pictureBox1.CreateGraphics();
```

Графическая поверхность состоит из отдельных точек – пикселов. Положение точки на графической поверхности характеризуется горизонтальной (*X*) и вертикальной (*Y*) координатами. Координаты точек отсчитываются от левого верхнего угла и возрастают слева направо (координата *X*) и сверху вниз (координата *Y*). Левая верхняя точка графической поверхности имеет координаты (0, 0).

Для формы **PictureBox** в свойстве **BorderStyle** установлено **FixedSingle** (неизменяемая одинарная рамка), то есть в процессе выполнения программы с помощью курсора мы не сможем менять размер формы.

Для отрисовки шестиугольника и треугольника используется полигон с заливкой:

```
g.FillPolygon(br1, p1); //отрисовка шестиугольника
 g.FillPolygon(br2, p2); //отрисовка треугольника
```

где p1, p2 – массив точек для шестиугольника и треугольника соответственно,

```
Brush br1 = new SolidBrush(Color.FromArgb(255,255,0)); // желтая кисть,
 Brush br2 = new SolidBrush(Color.FromArgb(0,255,0)); // зеленая кисть.
```

Цвет задается в модели aRGB: альфа (прозрачность), красный, зеленый, синий (от 0 до 255).

Порядок выполнения работы

1. Создать новый проект. Пользовательский интерфейс представлен на рисунке 5.2.

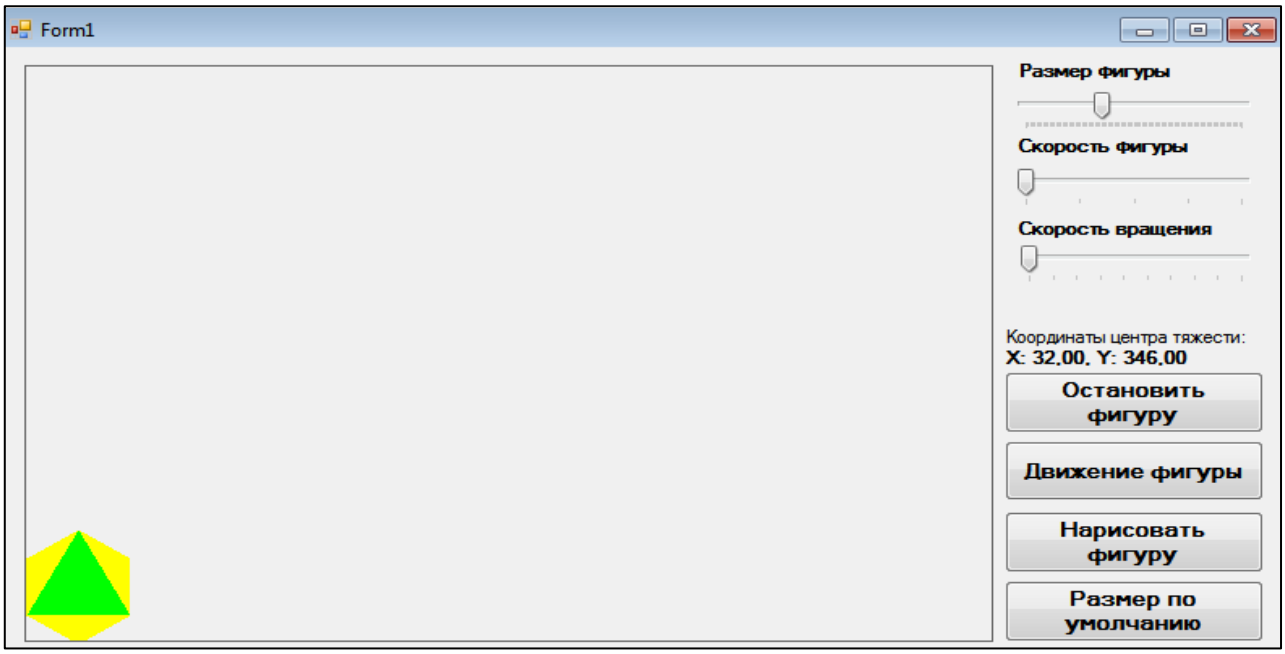

Рисунок 5.2 – Рабочее окно программы с начальным положением фигуры

Основные компоненты программы:

- PictureBox;
- Timer:

timer1 – позволяет запускать определенные действия по истечению некоторого периода времени;

- Button:

btnDraw – кнопка для отрисовки фигуры;

btnStart – кнопка запуска таймера;

btnStop – кнопка остановки таймера и возврата фигуры в начальное положение;

btnDefaultA – кнопка, устанавливающая размер фигуры по умолчанию;

- Label:

label1 – label4 – отображение текста для пояснения;

lblXcYc – отображение координат центра тяжести фигуры;

- TrackBar:

trbFigRotRate – ползунок изменения скорости вращения;

trbFigSize – ползунок изменение размера;

trbFigSpeed – ползунок изменения скорости движения.

2. Чтобы учесть изменение цвета фигуры при движении вдоль траектории и одновременном вращении вокруг центральной точки, необходимо провести анализ, как должны изменяться цветовые координаты R, G, B в процессе движения. Необходимо учесть, что шестиугольник в начале траектории был желтого цвета (R = 255; G = 255; B = 0), в конце – синего (R = 0; G = 0; B = 255). Таким образом, координаты R и G при движении фигуры в прямом направлении убывают от 255 до 0, а координата B возрастает от 0 до 255. В промежуточных точках цветовые координаты линейно зависят от x_C (рис. 5.3).

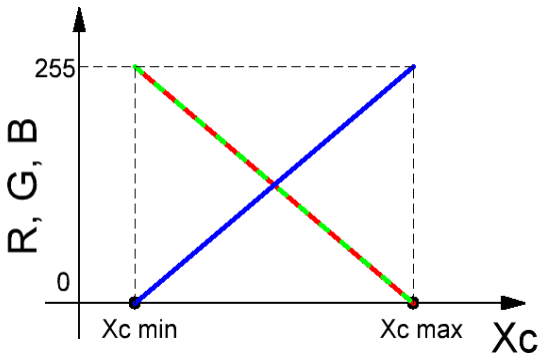

Рисунок 5.3 – Изменение цветовых координат R, G, B

Уравнения зависимости компонент R, G, B от x_c получим как уравнение прямой по двум точкам:

$$\frac{x_C - x_{C\min}}{x_{C\max} - x_{C\min}} = \frac{R - 255}{0 - 255},$$
$$\frac{x_C - x_{C\min}}{x_{C\max} - x_{C\min}} = \frac{G - 255}{0 - 255},$$
$$\frac{x_C - x_{C\min}}{x_{C\max} - x_{C\min}} = \frac{B - 0}{255 - 0}.$$

Из этих уравнений необходимо выразить R, G, B, а $x_{C\min}$ и $x_{C\max}$. определяются шириной PictureBox и размером фигуры *a*. Этот процесс в коде программы реализован в двух методах: byte colorInc() – для возрастающих цветовых координат и byte colorDec() – для убывающих цветовых координат.

Аналогичные рассуждения проводятся при учете изменение цвета треугольника от зеленого цвета (R = 0; G = 255; B = 0) в начале траектории, до пурпурного (R = 255; G = 0; B = 255) в конце.

3. Проанализировать код и создать соответствующие обработчики событий (листинг 5.1).

Листинг 5.1 – Код движения фигуры в плоскости

```
public partial class Form1 : Form
{ // объявляем переменные
    const int default_a = 120;
    int dir = 1; //направление движения по молчанию вперед
    double dx = 0.1; //приращение по x при движении
    double xc, yc; //координаты центра
    int a; //размер фигуры и координаты центра
    int v = 1; //скорость движения
    Point[] p1 = new Point[6]; //массив точек для шестиугольника
    Point[] p2 = new Point[3]; //массив точек для треугольника
    double[] x = new double[6];
    double[] y = new double[6];
    double fi; // угол, на который фигура поворачивается при обновлении таймера
    double c_fi = 0; // угол, на который фигура повернулась
    int rotation_rate = 1; // скорость вращения
```

```
    public Form1()
    {
        InitializeComponent();
    }

    // при загрузке формы
    private void Form1_Load(object sender, EventArgs e)
    {
        a = default_a; // задаем размер фигуры

        // Вычисляем координаты центра, чтобы фигура находилась в левом нижнем
углу
        // pictureBox1.Width и pictureBox1.Height - ширина и высота pictureBox
        xc = Convert.ToInt32((a / 2.0) * Math.Cos(Math.PI / 6));
        yc = Convert.ToInt32(pictureBox1.Height - a / 2.0);
        fi = 3*dir * Math.PI / 180;
        // начальное положение ползунков
        trbFigRotRate.Value = rotation_rate;
        trbFigSize.Value = a;
        trbFigSpeed.Value = v;
    }

    // расчет координат x точек при вращении
    private double[] x_rot(double[] x0, double[] y0)
    {
        double[] x1 = new double[x.Length];
        for (int i = 0; i < x.Length; i++)
        {
            x1[i] = (x0[i]) * Math.Cos(c_fi) - (y0[i]) * Math.Sin(c_fi);
        }
        return x1;
    }

    // Расчет координат y точек при вращении
    private double[] y_rot(double[] x0, double[] y0)
    {
        double[] y1 = new double[x.Length];
        for (int i = 0; i < x.Length; i++)
        {
            y1[i] = (y0[i]) * Math.Cos(c_fi) + (x0[i]) * Math.Sin(c_fi);
        }
        return y1;
    }

    // Расчет возрастающих цветовых координат
    private byte colorInc()
    {
        try
        {
            double col = 255 * (xc - (a / 2) * Math.Cos(Math.PI / 6)) /
                    (pictureBox1.Width - a * Math.Cos(Math.PI / 6));
            return Convert.ToByte(col);
        }
        catch { return 0; }
    }
```

```
    // Расчет убывающих цветовых координат
    private byte colorDec()
    {
        try
        {
            double col = -255 * ((xc - (a / 2) * Math.Cos(Math.PI / 6)) /
                  (pictureBox1.Width - a * Math.Cos(Math.PI / 6))) + 255;
            return Convert.ToByte(col);
        }
        catch { return 0; }
    }

    // Метод рисования фигуры
    private void ris()
    {
        Graphics g = pictureBox1.CreateGraphics(); // Создание графического
объекта
        Brush br1 = new SolidBrush(Color.FromArgb(colorDec(), colorDec(),
colorInc())); // кисть шестиугольника
        Brush br2 = new SolidBrush(Color.FromArgb(colorInc(), colorDec(),
colorInc())); //кисть треугольника
        g.Clear(SystemColors.Control); // стирание

        //рассчет координат точек фигуры относительно центра
        x[0] = Convert.ToInt32(0);
        y[0] = Convert.ToInt32(-a / 2.0);
        x[1] = Convert.ToInt32((a / 2.0) * Math.Cos(Math.PI / 6));
        y[1] = Convert.ToInt32(-(a / 2.0) * Math.Sin(Math.PI / 6));
        x[2] = Convert.ToInt32((a / 2.0) * Math.Cos(Math.PI / 6));
        y[2] = Convert.ToInt32((a / 2.0) * Math.Sin(Math.PI / 6));
        x[3] = Convert.ToInt32(0);
        y[3] = Convert.ToInt32(a / 2.0);
        x[4] = Convert.ToInt32(-(a / 2.0) * Math.Cos(Math.PI / 6));
        y[4] = Convert.ToInt32((a / 2.0) * Math.Sin(Math.PI / 6));
        x[5] = Convert.ToInt32(-(a / 2.0) * Math.Cos(Math.PI / 6));
        y[5] = Convert.ToInt32(-(a / 2.0) * Math.Sin(Math.PI / 6));

        //поворот фигуры
        double[] temp = x;
        x = x_rot(temp, y);
        y = y_rot(temp, y);

        //заполняем массив точек шестиугольника
        p1[0] = new Point((int)(x[0] + xc), (int)(y[0] + yc));
        p1[1] = new Point((int)(x[1] + xc), (int)(y[1] + yc));
        p1[2] = new Point((int)(x[2] + xc), (int)(y[2] + yc));
        p1[3] = new Point((int)(x[3] + xc), (int)(y[3] + yc));
        p1[4] = new Point((int)(x[4] + xc), (int)(y[4] + yc));
        p1[5] = new Point((int)(x[5] + xc), (int)(y[5] + yc));

        //заполняем массив точек треугольника
        p2[0] = new Point((int)(x[0] + xc), (int)(y[0] + yc));
        p2[1] = new Point((int)(x[2] + xc), (int)(y[2] + yc));
        p2[2] = new Point((int)(x[4] + xc), (int)(y[4] + yc));
```

```
        g.FillPolygon(br1, p1);//отрисовка шестиугольника
        g.FillPolygon(br2, p2);//отрисовка треугольника

        lblXcYc.Text = String.Format("X: {0:0.00}, Y: {1:0.00}", xc, yc); //
вывод координат центра тяжести точки
    }

    // При нажатии кнопки "Нарисовать фигуру"
    private void btnDraw_Click(object sender, EventArgs e)
    {
        ris();
    }

    // При нажатии кнопки "Движение фигуры"
    private void btnStart_Click(object sender, EventArgs e)
    {
        timer1.Start();
    }

    // При нажатии кнопки "Остановить фигуру"
    private void btnStop_Click(object sender, EventArgs e)
    {
        timer1.Stop();
        xc = Convert.ToInt32((a / 2.0) * Math.Cos(Math.PI / 6));
        yc = Convert.ToInt32(pictureBox1.Height - a / 2.0);
        c_fi = 0;
        ris();
    }

    // При нажатии кнопки "Размер по умолчанию"
    private void btnDefaultA_Click(object sender, EventArgs e)
    {
        a = default_a;
        if (!timer1.Enabled) //проверка включенности таймера(true-таймер
выключен)
        {
            xc = Convert.ToInt32((a / 2.0) * Math.Cos(Math.PI / 6));
            yc = Convert.ToInt32(pictureBox1.Height - a / 2.0);
            ris();
        }
    }

    private void timer1_Tick(object sender, EventArgs e)
    {
        if (x.Max() + xc > pictureBox1.Width)
        {
            xc = pictureBox1.Width - x.Max(); //если фигура заходит за границу,
            //уменьшаем координату центра
            dir = -1;//меняем направление движения
        }

        //проверяем касание левой границы
        if ((x.Min() + xc) < 0)
        {
            xc = -x.Min();
            dir = 1;
```

```
        }

        c_fi += fi * rotation_rate;

        ris(); //перерисовываем фигуру

        dx = v * dir; //приращение координаты зависит от направления и скорости
        fi = 1 * dir * Math.PI / (180); //угол меняет направление
        xc = xc + dx;
        yc = pictureBox1.Height / 2 + ((pictureBox1.Height - a) / 2) *
Math.Sin(0.03 * (xc - a / 2)); //движение по синусоиде
    }

    private void trbFigRotRate_Scroll(object sender, EventArgs e)
    { // Функция для изменения скорости вращения
        rotation_rate = trbFigRotRate.Value;
    }

    private void trbFigSpeed_Scroll(object sender, EventArgs e)
    { // Функция для изменения скорости без изменения направления движения
        v = trbFigSpeed.Value;
    }

    private void trbFigSize_Scroll(object sender, EventArgs e)
    { // Функция для изменения размера фигуры
        a = trbFigSize.Value;
        if (!timer1.Enabled) //проверка включенности таймера(true-таймер
выключен)
        {
            xc = Convert.ToInt32((a / 2.0) * Math.Cos(Math.PI / 6));
            yc = Convert.ToInt32(pictureBox1.Height - a / 2.0);
        }
        ris();
    }
}
```

Содержание отчета

Предоставить программу, написанную в Visual Studio на языке C# и результаты ее работы. Выполнить проверку возможности изменения скорости движения фигуры и скорости вращения данной фигуры вокруг центральной точки.

Контрольные вопросы

1. Для чего служит компонент TrackBar?
2. Назовите обработчик события компонента TrackBar.
3. В каких методах в коде программы реализуется изменение цвета?
4. Как в коде программы реализуется движение фигуры по синусоиде?
5. Для чего в программе используется компонент Timer?

ЛАБОРАТОРНАЯ РАБОТА 6. ИГРА «BALLS»

Цель работы: создание простейшей игры «Balls» на языке C#.

Краткие теоретические сведения

Первые математические аспекты и приложения теории игр были изложены в классической книге Джона фон Неймана и Оскара Моргенштерна «Теория игр и экономическое поведение» в 1944 г.

Игра – конфликт двух или более сторон, в котором каждая из сторон преследует свои личные интересы. Раздел математики, изучающий конфликтные ситуации на основе их математических моделей, называется теорией игр.

Стратегия – совокупность правил, однозначно определяющих последовательность действий игрока в каждой конкретной ситуации, складывающейся в процессе игры. Эти правила определяют:

– варианты действий игроков;

– объем информации каждого игрока о поведении партнеров;

– выигрыш (или проигрыш), к которому приводит каждая совокупность действий.

Различные виды игр можно классифицировать, основываясь на том или ином признаке, характеризующим игру:

– по количеству игроков;

– по количеству стратегий (конечное или бесконечное число; если в игре все игроки имеют конечное число возможных стратегий, то она называется конечной. Если же хотя бы один из игроков имеет бесконечное количество возможных стратегий игра называется бесконечной);

– по степени информированности игроков о стратегиях, сделанных ходах и предпочтениях противника (игры с полной/неполной информацией);

– по характеру выигрышей (игры с нулевой суммой, игры с ненулевой суммой);

– по возможности предварительных переговоров и взаимодействий между игроками в ходе игры (коалиционные, кооперативные, бескоалиционные игры).

Бескоалиционные игры – это класс игр, в которых каждый игрок принимает решение независимо от других игроков (изолированно), не участвуя ни в каких переговорах и соглашениях с другими игроками.

В коалиционных (кооперативных) играх, напротив, игроки могут принимать решения по согласованию друг с другом (им разрешается обсуждать перед игрой свои стратегии и договариваться о совместных действиях), они вправе вступать в коалиции.

Порядок выполнения работы

1. Создать новый проект. Пользовательский интерфейс представлен на рисунке 6.1.

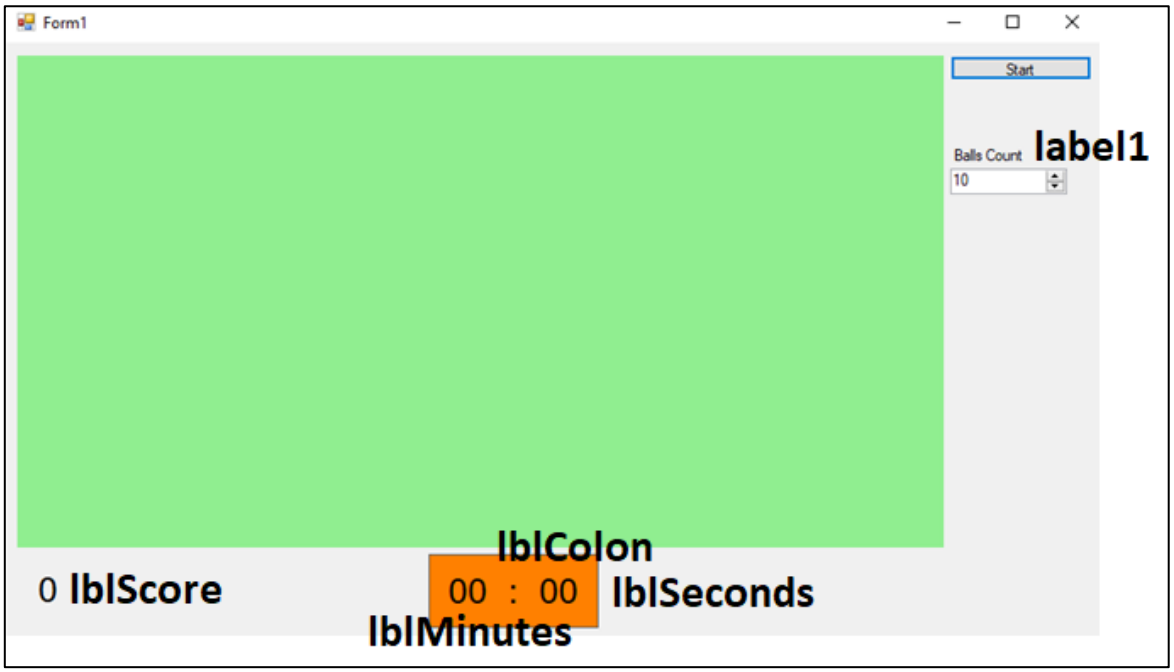

Рисунок 6.1 – Рабочее окно программы

Нанести на форму следующие компоненты:

- PictureBox (Name = pbGame). Компонент PictureBox – это область рисования. Компонент визуальный и используется для создания на форме некоторой области для рисования, т. е. назначение этого компонента – дать простое окно для рисования изображений.
- ButtonClick (Name = btnStart, Text = Start). Компонент Button (кнопка) – командная кнопка, с помощью которой пользователь может вызывать выполнение какого-либо действия (В нашем случае запуск игры).
- Timer. Компонент Timer (таймер) служит для инициирования какого-либо события через определенные интервалы (промежутки) времени. Использование таймеров помогает нам осуществить движение фигуры и её отскок при касании границ рабочей области. В программе используется два компонента Timer. Первый Name = tmGameExecution используется для организации движения фигур, второй Name = tmStopwatch – для секундомера. Во время создания формы свойству Interval таймера tmStopwatch надо присвоить значение 500. Кнопка Start предназначена как для запуска секундомера, так и для его остановки. В начале работы программы значение свойства Enabled компонента Timer равно False, поэтому таймер не работает (не генерирует события Tick).
- NumericUpDown (name = nudBallsCount). Элемент управления содержит одно числовое значение, которое можно увеличить или уменьшить, нажав кнопки со стрелками вверх или вниз.
- Label. Компонент Label предназначен для показа текста на форме нашей программы

2. При написании кода заданы начальные значения: массив скоростей – velocities, радиус шариков (ballRadius), начальное число шаров (ballsCount),

начальное значение у координаты для шаров задается генератором случайных чисел (rnd).

В процессе реализации кода и создания соответствующих обработчиков событий (листинг 6.1) необходимо учесть:

- Stopwatch.ElapsedMilliseconds (Пространство имен: System.Diagnostics). Свойство получает общее затраченное время в миллисекундах, измеренное текущим экземпляром (1с = 1000 мс). Значение свойства (Int64) – доступное только для чтения длинное целое число, представляющее общее число миллисекунд, измеренное текущим экземпляром;
- FillEllipse(Brush, RectangleF). Заполняет внутреннюю часть эллипса, определяемого ограничивающим прямоугольником, который задан структурой RectangleF;
- Invalidate(). Делает недействительной всю поверхность элемента управления и вызывает его перерисовку;
- метод void CreateBall(int n, Brush br) создает шарики и задает им скорости генератором случайных чисел;
- для таймера tmGameExecutio генерируется событие Tick. В результате реализуется движение шариков и изменение цвета;
- событие Tick так же генерируется для таймера tmStopwatch. Оно обеспечивает работу секундомера;
- событие Paint для PictureBox (pbGame) реализует отрисовку шаров, а событие MouseDown позволяет работать с мышью;
- обработчик события кнопки btnStart запускает и останавливает игру.

Листинг 6.1 – Код игры «BALLS»

```
public partial class Form1 : Form
{
    private RectangleF[] ballBounds;
    private Brush[] brushes;
    private Color backgroundColor = Color.LightGreen;
    private float[] velocities;
    private float ballRadius = 15;
    private int ballsCount = 10;
    private Random rnd;
    private List<int> crackingBallsIndexes;
    private Stopwatch stopwatch = new Stopwatch();
    private int score = 0;

    public Form1()
    {
        InitializeComponent();
    }

    private void InitializeGame()
    {
        ballsCount = (int)nudBallsCount.Value;
        score = 0;
```

```
    crackingBallsIndexes = new List<int>();
    ballBounds = new RectangleF[ballsCount];
    velocities = new float[ballsCount];
    brushes = new Brush[ballsCount];
    rnd = new Random();

    for (int n = 0; n < ballsCount; n++)
    {
        CreateBall(n, Brushes.Blue);
    }
}

void CreateBall(int n, Brush br)
{
    double y = rnd.NextDouble() * (pbGame.Height - ballRadius*2);
    var bounds = new RectangleF(0, (float)y, ballRadius*2, ballRadius*2);
    var v = rnd.Next(2, 8);
    velocities[n] = v;
    ballBounds[n] = bounds;
    brushes[n] = br;
}

private void btnStart_Click(object sender, EventArgs e)
{
    if (!tmGameExecution.Enabled)
    {
        InitializeGame();
        tmGameExecution.Start();
        btnStart.Text = "Stop";
        nudBallsCount.Enabled = false;
        tmStopwatch.Start();
        stopwatch.Restart();
        lblSeconds.Text = "00";
        lblMinutes.Text = "00";
        lblScore.Text = "0";
    }
    else
    {
     tmGameExecution.Stop();
     btnStart.Text = "Start";
     pbGame.Invalidate();
     nudBallsCount.Enabled = true;
     tmStopwatch.Stop();
     stopwatch.Stop();
     lblColon.Visible = true;
     MessageBox.Show("You catched "+score+" balls in "+lblMinutes.Text +
               " minutes "+lblSeconds.Text +" seconds!");
     }
}

private void tmGameExecution_Tick(object sender, EventArgs e)
{
    for (int i = 0; i < ballsCount; i++)
    {
        if (!crackingBallsIndexes.Contains(i))
```

```
        {
            ballBounds[i].X += velocities[i];
            if (velocities[i] > 0)
            {
                if (ballBounds[i].X+ballRadius*2>=pbGame.Width-ballRadius)
                {
                    velocities[i] = -velocities[i];
                }
            }
            else
            {
                if (ballBounds[i].X <= 0 + ballRadius)
                {
                    velocities[i] = -velocities[i];
                }
            }
        }
        else
        {
            ballBounds[i].X += 2;
            ballBounds[i].Y += 2;
            ballBounds[i].Width -= 2;
            ballBounds[i].Height -= 2;
            if (ballBounds[i].Width <= 2)
            {
                crackingBallsIndexes.Remove(i);
                CreateBall(i,brushes[i]==Brushes.Blue?Brushes.Red:Brushes.Blue);
            }
        }
    }
    pbGame.Invalidate();
}

private void tmStopwatch_Tick(object sender, EventArgs e)
{
    lblColon.Visible = !lblColon.Visible;
    var seconds = stopwatch.ElapsedMilliseconds / 1000;
    var minutes = seconds / 60;
    var displayedSeconds = seconds % 60;
    if (displayedSeconds < 10)
    {
        lblSeconds.Text = "0" + displayedSeconds.ToString();
    }
    else
    {
        lblSeconds.Text = displayedSeconds.ToString();
    }

    if (minutes < 10)
    {
        lblMinutes.Text = "0" + minutes.ToString();
    }
    else
    {
        lblMinutes.Text = minutes.ToString();
    }
```

```
    }

    private void pbGame_Paint(object sender, PaintEventArgs e)
    {
        e.Graphics.Clear(backgroundColor);

        if (tmGameExecution.Enabled)
        {
            for (int i = 0; i < ballsCount; i++)
            {
                e.Graphics.FillEllipse(brushes[i], ballBounds[i]);
            }
        }
    }

    private void pbGame_MouseDown(object sender, MouseEventArgs e)
    {
        for (int i = 0; i < ballsCount; i++)
        {
            if ((Math.Abs(e.Location.X - (ballBounds[i].Left + ballRadius)) <
ballRadius) && (Math.Abs(e.Location.Y - (ballBounds[i].Top + ballRadius)) <
ballRadius))
            {
                if (!crackingBallsIndexes.Contains(i))
                {
                    crackingBallsIndexes.Add(i);
                    score++;
                    lblScore.Text = score.ToString();
                    break;
                }
            }
        }
    }
}
```

3. Протестировать работу программы с разным количеством шаров (рис. 6.2).

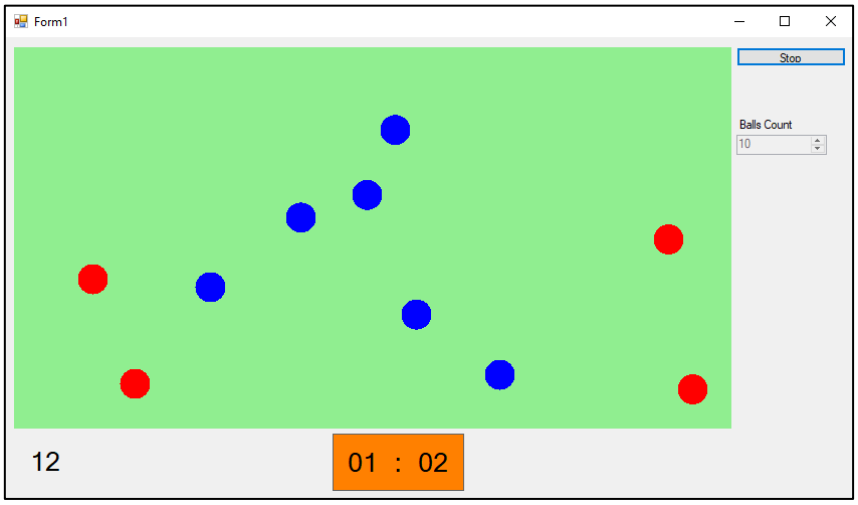

Рисунок 6.2 – Рабочее окно программы в процессе игры

Содержание отчета

Предоставить программу, написанную в Visual Studio на языке C# и результаты ее работы.

Контрольные вопросы

1. Для чего служит компонент NumericUpDown?
2. Какие Timer используются в коде и для чего?
3. Для чего используется метод void CreateBall (int n, Brush br)?
4. Для чего используется событие Paint?
5. Для чего в программе используется событие MouseDown?

ЛАБОРАТОРНАЯ РАБОТА 7. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ СРЕДСТВ ДЛЯ ОТОБРАЖЕНИЯ ГРАФИЧЕСКОЙ ИНФОРМАЦИИ

Цель работы: изучить возможности построения графиков в Visual Studio.Net с помощью компонента Chart библиотеки визуализации System.Windows.Forms.DataVisualization.

Краткие теоретические сведения

Обычно результаты расчетов представляются в виде графиков и диаграмм. Библиотека .NET Framework имеет мощный элемент управления Chart для отображения на экране графической информации.

Построение графика (диаграммы) производится после вычисления таблицы значений функции y = f(x) на интервале [Xmin, Xmax] с заданным шагом. Полученная таблица передается в специальный массив Points объекта Series компонента Chart с помощью метода DataBindXY.

Компонент Chart осуществляет всю работу по отображению графиков: строит и размечает оси, рисует координатную сетку, подписывает название осей и самого графика, отображает переданную таблицу в виде всевозможных графиков или диаграмм. При необходимости компоненту Chart передаются данные о толщине, стиле и цвете линий, параметрах шрифта подписей, шагах разметки координатной сетки и другие настройки. В процессе работы программы изменение параметров возможно через обращение к соответствующим свойствам компонента Chart. Так, например, свойство AxisX содержит значение максимального предела нижней оси графика и при его изменении во время работы программы автоматически изменяется изображение графика.

Порядок выполнения работы

1. Составить программу, отображающую графики функций sin(x) и cos(x) на интервале [Xmin, Xmax]. Предусмотреть возможность изменения разметки координатных осей, а также шага построения таблицы.

Все элементы на форме создать в коде. Прежде всего, следует определить в коде класса все необходимые переменные и константы (листинг 7.1).

Листинг 7.1 – Описание переменных

```
public Form1()
{
    InitializeComponent();
}

private double XMin = -Math.PI; // Левая граница графика
private double XMax = Math.PI; // Правая граница графика
private double Step = (Math.PI * 2) / 10; // Шаг графика
private double[] x; // Массив значений X - общий для обоих графиков
private double[] y1;
private double[] y2;
Chart chart; // следует описать глобальную переменную типа Chart, к которой
             // мы будем обращаться из разных методов
```

2. Поскольку данный класс не входит в пространства имен, подключаемых по умолчанию, следует выполнить дополнительные действия. Во-первых, в обозревателе решений нужно щёлкнуть правой кнопкой по секции ссылки и добавить ссылку на библиотеку визуализации (рис. 7.1).

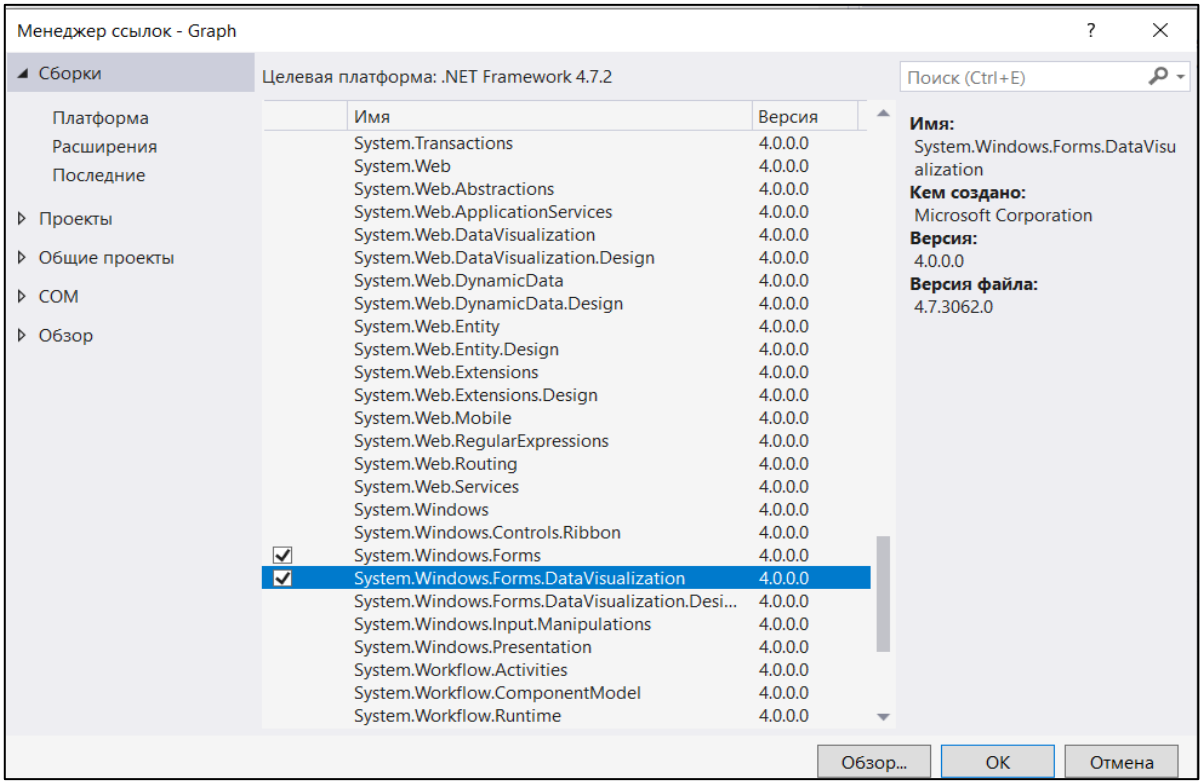

Рисунок 7.1 – Добавление ссылки на библиотеку визуализации

Кроме того, следует подключить соответствующее пространство имен:

```
using System.Windows.Forms.DataVisualization.Charting;
```

3. Далее следует определить метод, который будет рассчитывать

количество шагов и вычислять значения функций в каждой точке, внося вычисленные значения в массивы x, y1 и y2 (листинг 7.2).

Листинг 7.2 – Метод CalcFunction()

```
private void CalcFunction() //Расчёт значений графика
{
    // Количество точек графика
    int count = (int)Math.Ceiling((XMax - XMin) / Step) + 1;
    // Создаём массивы нужных размеров
    x = new double[count];
    y1 = new double[count];
    y2 = new double[count];
    // Расчитываем точки для графиков функции
    for (int i = 0; i < count; i++)
    {
        // Вычисляем значение X
        x[i] = XMin + Step * i;
        // Вычисляем значение функций в точке X
        y1[i] = Math.Sin(x[i]);
        y2[i] = Math.Cos(x[i]);
    }
}
```

4. После расчёта значений нужно отобразить графики на форме с помощью элемента Chart. Элемент управления Chart нельзя выбрать с помощью панели элементов – его нужно создавать прямо в коде программы. Вторым шагом следует создать область отображения графика и настроить внешний вид осей (листинг 7.3).

Листинг 7.3 – Метод CreateChart()

```
private void CreateChart()
{   // Создаём новый элемент управления Chart
    chart = new Chart();
    // Помещаем его на форму
    chart.Parent = this;
    // Задаём размеры элемента
    chart.SetBounds(10, 10, ClientSize.Width - 20, ClientSize.Height - 20);
    // Создаём новую область для построения графика
    ChartArea area = new ChartArea();
    // Даём ей имя (чтобы потом добавлять графики)
    area.Name = "myGraph";
    // Задаём левую и правую границы оси X
    area.AxisX.Minimum = XMin;
    area.AxisX.Maximum = XMax;
    // Определяем шаг сетки
    area.AxisX.MajorGrid.Interval = Step;
    // Добавляем область в диаграмму
    chart.ChartAreas.Add(area);
    // Создаём объект для первого графика
    Series series1 = new Series();
    // Ссылаемся на область для построения графика
    series1.ChartArea = "myGraph";
    // Задаём тип графика - сплайны
    series1.ChartType = SeriesChartType.Spline;
```

```
    // Указываем ширину линии графика
    series1.BorderWidth = 3;
    // Название графика для отображения в легенде
    series1.LegendText = "sin(x)";
    // Добавляем в список графиков диаграммы
    chart.Series.Add(series1);
    // Аналогичные действия для второго графика
    Series series2 = new Series();
    series2.ChartArea = "myGraph";
    series2.ChartType = SeriesChartType.Spline;
    series2.BorderWidth = 3;
    series2.LegendText = "cos(x)";
    chart.Series.Add(series2);
    // Создаём легенду, которая будет показывать названия
    Legend legend = new Legend();
    chart.Legends.Add(legend);
}
```

5. Наконец, все эти методы следует откуда-то вызвать. Чтобы графики появлялись сразу после запуска программы, надо вызывать их в обработчике события Load формы (листинг 7.4).

Листинг 7.4 – Обработчик события Load формы

```
private void Form1_Load(object sender, EventArgs e)
{
    // Создаём элемент управления
    CreateChart();
    // Расчитываем значения точек графиков функций
    CalcFunction();
    // Добавляем вычисленные значения в графики
    chart.Series[0].Points.DataBindXY(x, y1);
    chart.Series[1].Points.DataBindXY(x, y2);
}
```

6. Протестировать работу программы (рис. 7.2)

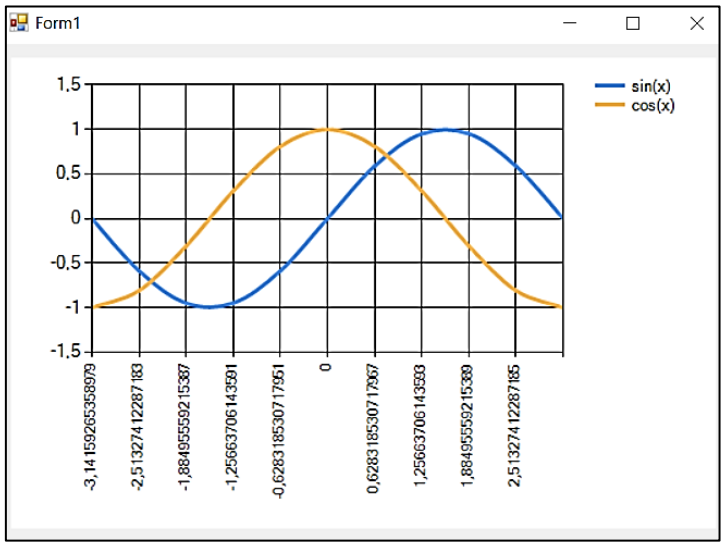

Рисунок 7.2 – Окно программы при запуске

7. Создать приложение построения графика функции $y = ax^2 + bx + c$. Постоянные коэффициенты *a, b, c* вводить с формы.

Содержание отчета

Предоставить программы, написанные в Visual Studio на языке C# и результаты их работы.

Контрольные вопросы

1. Как можно добавить ссылку на библиотеку визуализации System.Windows.Forms.DataVisualization?
2. Поясните работу метода private void CalcFunction()?
3. Как в коде создается элемент управления Chart?
4. Для чего используется обработчик события Load формы?
5. С помощью какой команды можно изменить толщину линии графика?

ЛАБОРАТОРНАЯ РАБОТА 8. КОМПОНЕНТ CHART

Цель работы: изучить возможности построения графиков, используя компонент Chart из группы «Данные» панели элементов.

Краткие теоретические сведения

В Microsoft Visual Studio элемент управления dataGridView позволяет организовывать данные в виде таблицы. Данные могут быть получены из базы данных, коллекции, внутренних переменных – массивов или других объектов программы.

Данный элемент размещен на панели инструментов ToolBox во вкладках «All Windows Forms» или «Data». После размещения на форме, система создает объект (переменную) с именем dataGridView1. С помощью этого имени можно программно оперировать методами и свойствами этого элемента управления.

Для задания размеров DataGridView используются свойства Width и Height:

```
dataGridView1.Width = 350;
dataGridView1.Height = 200;
```

Добавить столбец в dataGridView можно:

– с помощью специального мастера;

– программным путем.

Столбцы в dataGridView организованы в виде коллекции Columns типа DataGridViewColumnCollection. Чтобы добавить столбец программным путем используется метод (команда) Add из коллекции Columns.

Чтобы удалить столбец используется один из двух методов из коллекции Columns:

- метод RemoveAt() – удаляет столбец по заданному индексу в коллекции;
- метод Remove() – удаляет столбец по его имени.

Порядок выполнения работы

1. На «Form1» поместить компоненты: Chart1, dataGridView1, Button1.

В свойствах найти «Series». Оно отвечает за заполнение компонента данными и является коллекцией. Коллекция позволяет на одном компоненте строить несколько серий.

Кнопкой «Добавить» можно увеличивать число серий или уменьшать кнопкой «Удалить» (рис. 8.1).

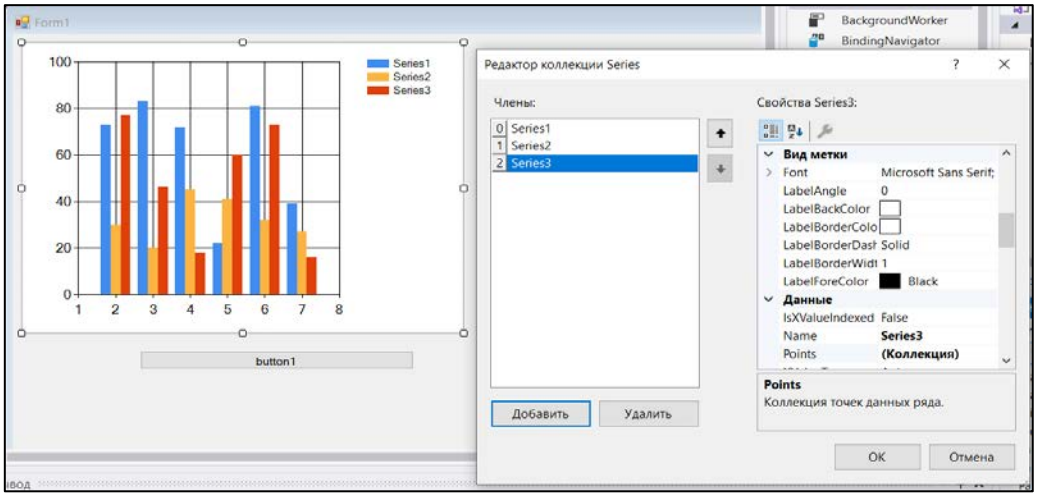

Рисунок 8.1 – Добавление серий

В нашей задаче серия одна и у нее индекс «0». Тип настраиваемых данных выбирается в свойстве «ChartType» редактора коллекции «Series». По умолчанию стоит «Column». Мы выбираем «Spline» (рис. 8.2).

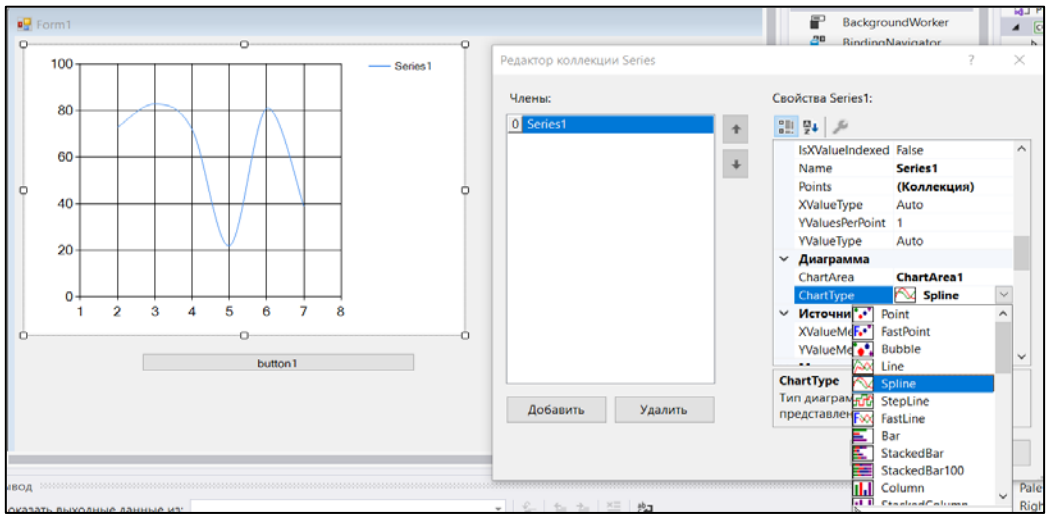

Рисунок 8.2 – Настройка данных

2. В редакторе коллекции «Series» можно поменять толщину графика – «BorderWidth», цвет – «Color». Легенду можно удалить, поскольку у нас один график. Для этого в свойствах проекта необходимо открыть «Legent» → «Коллекция» и нажать в редакторе коллекции «Legend1» кнопку «Удалить» (рис. 8.3).

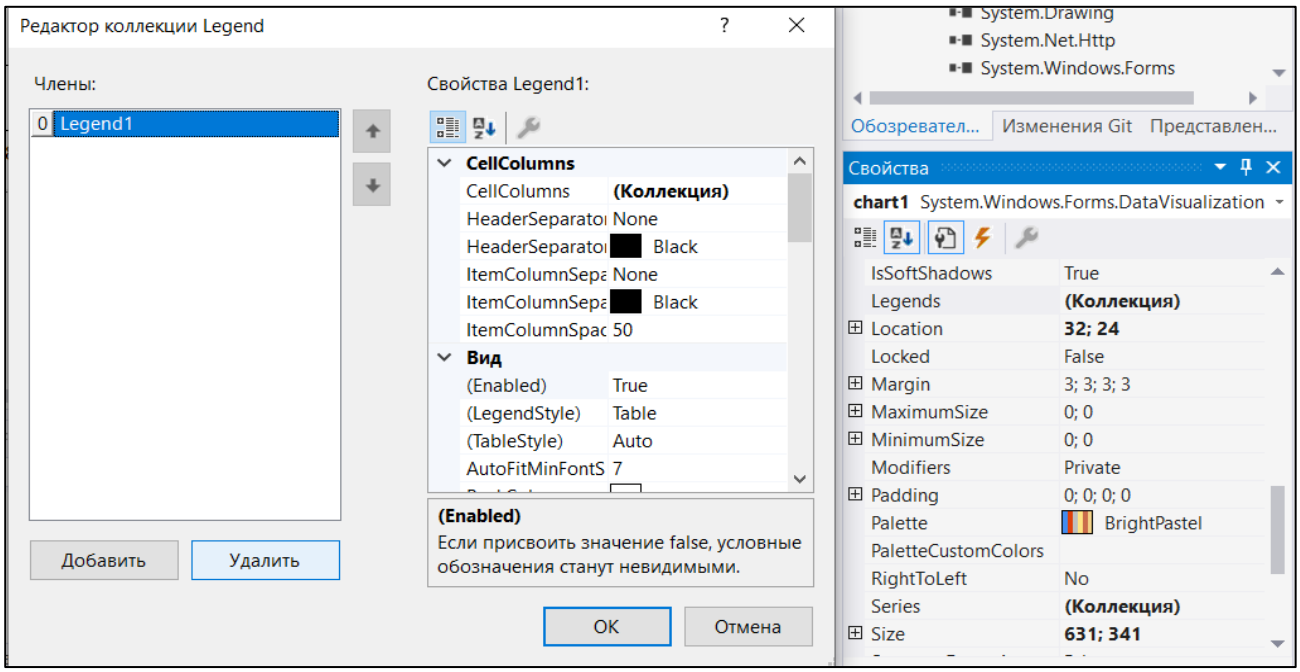

Рисунок 8.3 – Удаление «Legend»

Далее в свойствах проекта открыть «Titles» → «Коллекция». В «text» написать название графика и используя «Font», «BorderWidth», «ForeColor» измените цвет, толщину и размер шрифта (рис. 8.4).

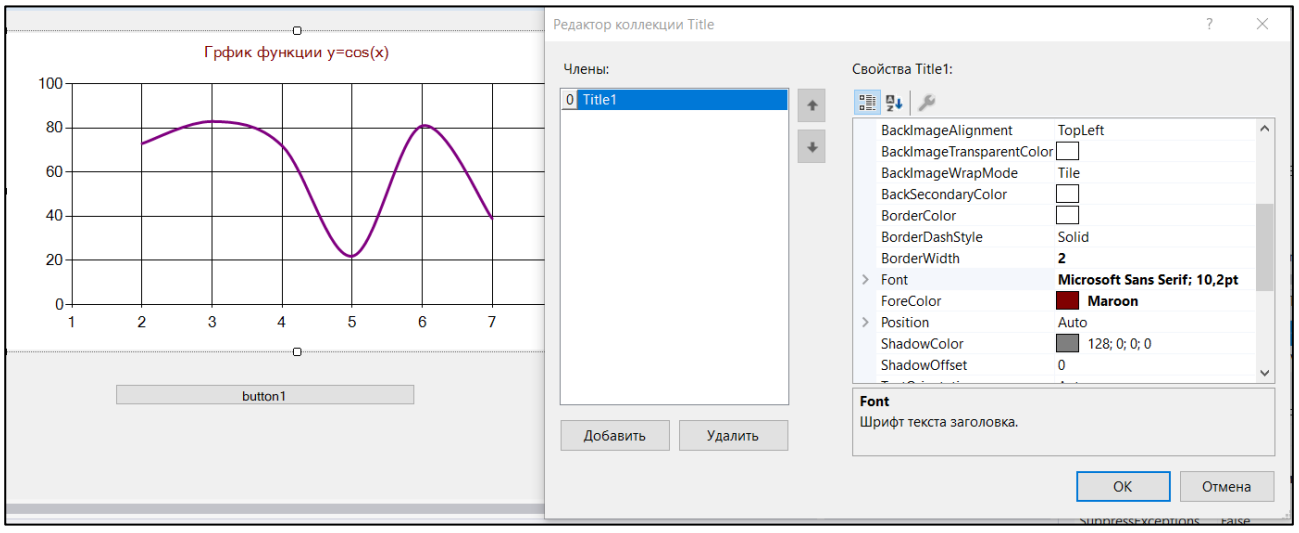

Рисунок 8.4 – Добавление заголовка

3. Создать приложение построения графика функции y = cosx. Вид приложения при запуске изображен на рисунке 8.5.

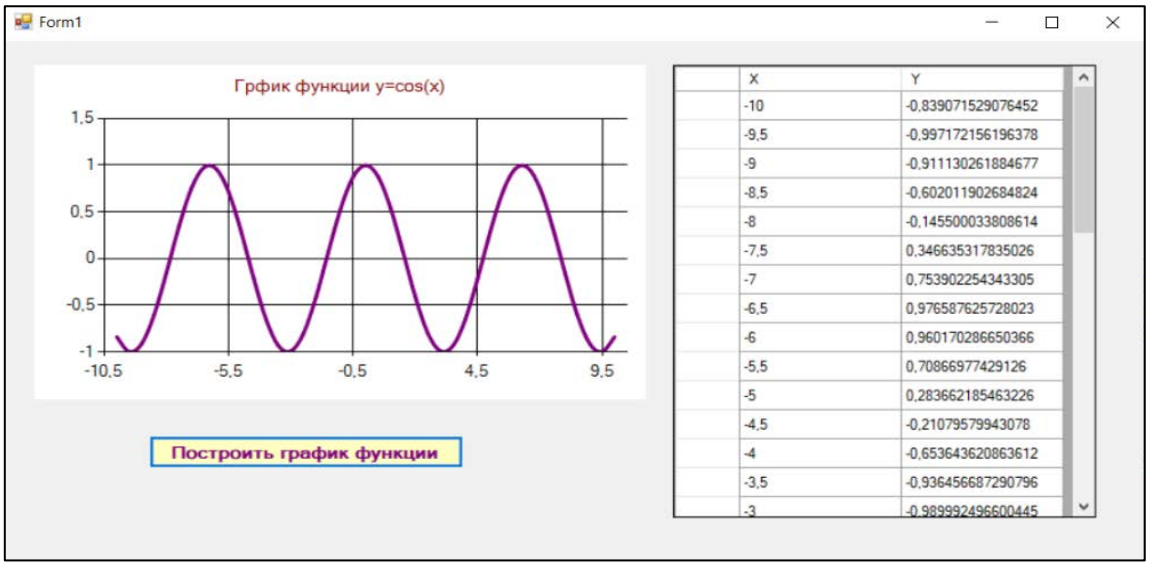

Рисунок 8.5 – Вид приложения при запуске

4. Проанализировать код и создать соответствующие обработчики событий (листинг 8.1).

Листинг 8.1 – Код построения графика функции y = cosx

```
public partial class Form1 : Form
{
    List<double> coord_x = new List<double>();
    List<double> coord_y = new List<double>();
    int razmer;
    public Form1()
    {
        InitializeComponent();
        dataGridView1.ColumnCount = 2;
    }

    private void button1_Click(object sender, EventArgs e)
    {
        double xLeft = -10;
        double xRight = 10;
        double h = 0.5; //шаг по оси x
        double x, y;//текущие координаты
        //обращаемся к диаграмме chart1, далее указываем нашу серию Series[0]
        // и далее обращаемся к свойству Points- это наши точки для диаграммы
        // эти точки перед расчетами очищаем
        this.chart1.Series[0].Points.Clear();
        x = xLeft;
        while (x <= xRight)
        {
            y = Math.Cos(x);
            coord_x.Add(x);
            coord_y.Add(y);
            this.chart1.Series[0].Points.AddXY(x, y);
            x += h;
        }
        razmer = coord_x.Count;//сколько элементов в List
```

```
            dataGridView1.RowCount = razmer;
            for (int i=0; i<razmer; i++)
            {
                dataGridView1.Rows[i].Cells[0].Value = coord_x[i];
                dataGridView1.Rows[i].Cells[1].Value = coord_y[i];
            }
        }
}
```

5. Создать новый проект. Построить графики для двух произвольно подобранных элементарных функций.

6. Создать новый проект. Постройте графики, используя ввод точек с формы (табл. 8.1).

Таблица 8.1 – Табличный способ задания функций

x	y1	y2
5	25	0,2
6	36	0,166667
7	49	0,142857
8	64	0,125
9	81	0,111111
10	100	0,1
11	121	0,090909
12	144	0,083333
13	169	0,076923
14	196	0,071429
15	225	0,066667
16	256	0,0625
17	289	0,058824
18	324	0,055556

На Form1 (рис. 8.6) поместить компоненты: numericUpDown1, Chart1, dataGridView1, кнопки: btnGraf и btnClear.

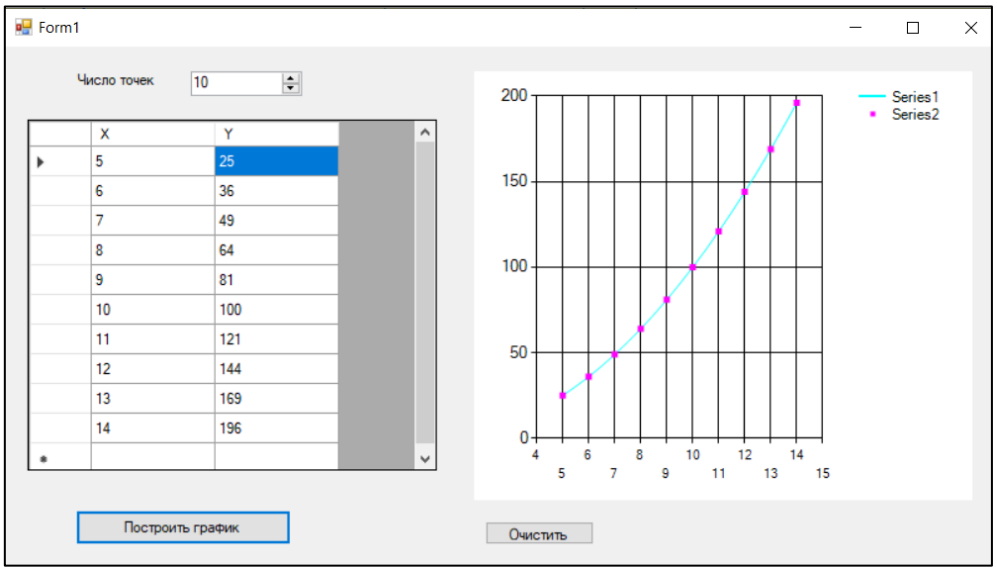

Рисунок 8.6 – Вид приложения при запуске

Вид редактора коллекции «Series» приведен на рисунке 8.7.

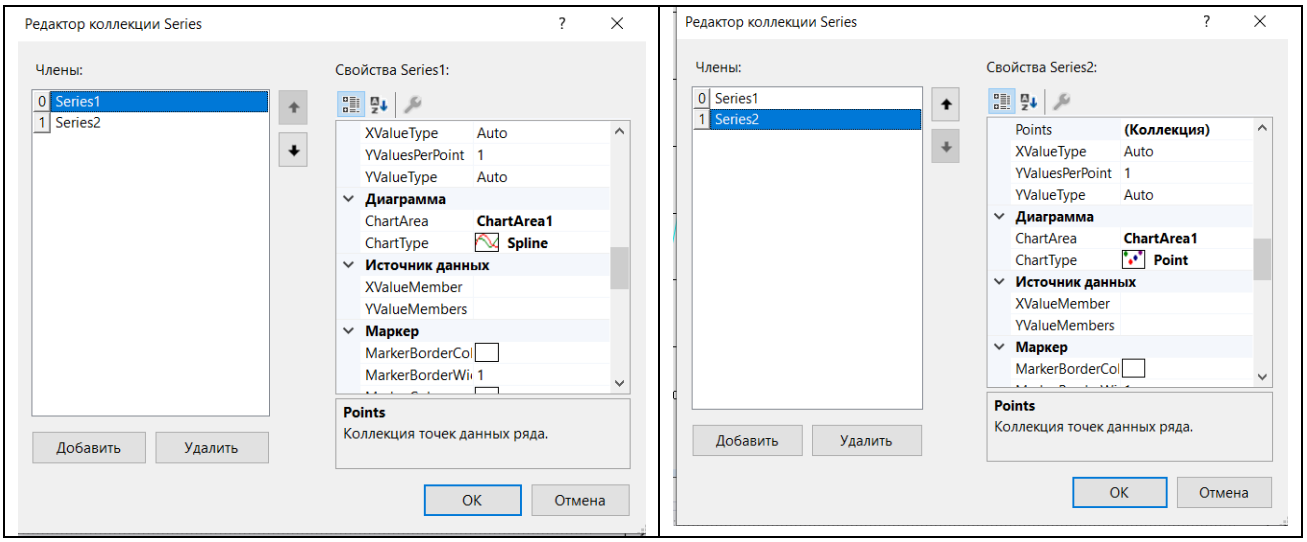

Рисунок 8.7 – Вид редактора коллекции Series

7. Проанализировать код и создать соответствующие обработчики событий (листинг 8.2).

Листинг 8.2 – Код построения графика таблично заданной функции

```
public partial class Form1 : Form
{
    public int N_Point = 1;

    public Form1()
    {
        InitializeComponent();
        dataGridView1.ColumnCount = 2;
        dataGridView1.Columns[0].Name = "X";
        dataGridView1.Columns[1].Name = "Y";
    }

    private void btnGraf_Click(object sender, EventArgs e)
    {
        for (int i = 0; i <= N_Point; i++)
        {
            double x = Convert.ToDouble(dataGridView1.Rows[i].Cells[0].Value);
            double y = Convert.ToDouble(dataGridView1.Rows[i].Cells[1].Value);
            this.chart1.Series[0].Points.AddXY(x, y);
            this.chart1.Series[1].Points.AddXY(x, y);
        }
    }

    private void numericUpDown1_ValueChanged(object sender, EventArgs e)
    {
        N_Point = Convert.ToInt32(numericUpDown1.Text);
        dataGridView1.RowCount = N_Point + 1;
    }

    private void btnClear_Click(object sender, EventArgs e)
```

```
    {
        dataGridView1.Rows.Clear();
        dataGridView1.Columns.Clear();
        chart1.Series[0].Points.Clear();
        chart1.Series[1].Points.Clear();
        numericUpDown1.Text = Convert.ToString(1);
        dataGridView1.ColumnCount = 2;
        dataGridView1.Columns[0].Name = "X";
        dataGridView1.Columns[1].Name = "Y";
    }
}
```

Содержание отчета

Предоставить программы, написанные в Visual Studio на языке C# и результаты их работы.

Контрольные вопросы

1. Для чего используется элемент управления DataGridView?
2. Как программно задать размеры DataGridView?
3. Добавление столбца программным путем в DataGridView.
4. Программное добавление строки в DataGridView.
5. Установка высоты заданной строки dataGridView1.

ЛАБОРАТОРНАЯ РАБОТА 9. ИГРА «КРЕСТИКИ-НОЛИКИ»

Цель работы: создание игры «Крестики-нолики» на языке C#.

Краткие теоретические сведения

«Крестики-нолики» (Tic Tac Toe Game) – логическая игра между двумя противниками на квадратном поле 3 на 3 клетки или бóльшего размера (вплоть до «бесконечного поля»). Один из игроков играет «крестиками», второй – «ноликами».

Сначала делает ход первый игрок. Хотя первый ход обычно делают «крестики», первый игрок может сходить и «ноликом», если захочет. Их нужно расположить на игровом поле таким образом, чтобы три одинаковых символа встали в ряд.

Если вы делаете ход первым, самым выгодным будет поставить символ в центральный квадрат. Это увеличит ваши шансы на победу, давая возможность построить ряд «крестиков» или «ноликов» в четырех направлениях (больше, чем дает любой другой ход).

Если противник делает первый ход не в центральный квадрат поля, вам следует поставить туда свой символ. Но, если противник делает ход в центр, наилучшим вариантом для вас будет ход в один из угловых квадратов.

Порядок выполнения работы

1. Создать приложение Windows Form (.NET Framework).
2. Изменить заголовок формы (свойство «Text»).

3. На форму поместить «gropBox» и в него 9 кнопок. Удалить текст с каждой кнопки (свойство «Text»). Создать единую функцию для обработки 9 кнопок. Для этого для всех кнопок прописать событие «Click» → button1_Click. Цвет кнопок и размер шрифта изменяется свойствами «BackColor» и «Font». Дизайн приложения разработать самостоятельно.

4. На форму так же следует поместить «label1» (для вывода информации о следующем ходе) и кнопку «NEW GAME» (для перехода на новую игру) (рис. 9.1).

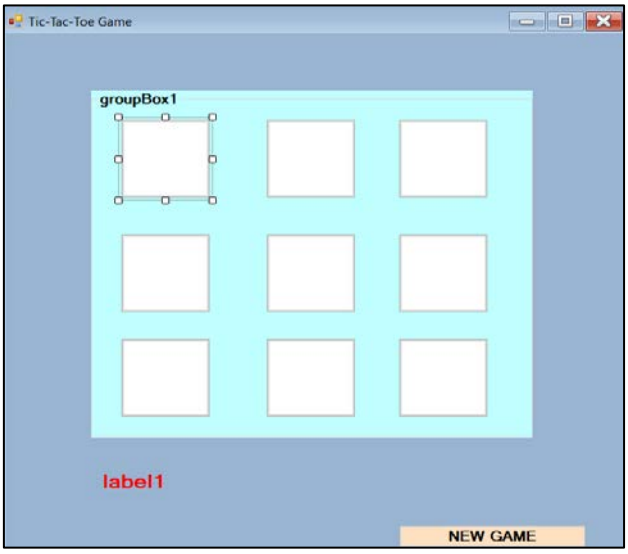

Рисунок 9.1 – Рабочее окно программы

5. Проанализировать и реализовать код. Создать соответствующие обработчики событий (листинг 9.1).

Листинг 9.1 – Tic Tac Toe Game

```
public partial class Form1 : Form
{
    string t = "";
    Random rnd = new Random(); //переменная отвечает кто первым будет ходить
крестики или нолики
    int[] array = new int[9];
    Button[] btn = new Button[9];

    public Form1()
    {
        InitializeComponent();
        if (rnd.Next(1, 10) <= 5)
        {
            t = "Крестики";
            label1.Text = "Ходят X";
        }
        else
        {
            t = "Ходят 0";
            label1.Text = "Ходят 0";
        }
```

```
        btn[0] = button1; //добавляем все кнопки в массив
        btn[1] = button2;
        btn[2] = button3;
        btn[3] = button4;
        btn[4] = button5;
        btn[5] = button6;
        btn[6] = button7;
        btn[7] = button8;
        btn[8] = button9;
    }

    private void button1_Click(object sender, EventArgs e)
    {
        if (t == "Крестики")
        {
            // проверяем какая кнопка нажата
            for (int i = 0; i < btn.Length; i++)
            {
                if (sender == btn[i])
                {
                    btn[i].Text = "X";
                    label1.Text = "Ходят 0";
                    t = "1";
                    btn[i].Enabled = false; //кнопка станет неактивной
                    array[i] = 1;
                    break;
                }
            }
        }
        else //нолики
        {
            for (int i = 0; i < btn.Length; i++)
            {
                if (sender == btn[i])
                {
                    btn[i].Text = "0";
                    label1.Text = "Ходят X";
                    t = "Крестики";
                    btn[i].Enabled = false; //кнопка станет неактивной
                    array[i] = 2;
                    break;
                }
            }
        }

        if(array[0]==1 && array[1] == 1 && array[2] == 1)
        {
            btn[0].BackColor = System.Drawing.Color.Red;
            btn[1].BackColor = System.Drawing.Color.Red;
            btn[2].BackColor = System.Drawing.Color.Red;
            for (int i = 0; i < btn.Length; i++)
            {
                btn[i].Enabled = false;
            }
            button10.Enabled = true;
            label1.Text = "Выиграли крестики";
```

```
}
else if (array[3] == 1 && array[4] == 1 && array[5] == 1)
{
    btn[3].BackColor = System.Drawing.Color.Red;
    btn[4].BackColor = System.Drawing.Color.Red;
    btn[5].BackColor = System.Drawing.Color.Red;
    for (int i = 0; i < btn.Length; i++)
    {
        btn[i].Enabled = false;
    }
    button10.Enabled = true;
    label1.Text = "Выиграли крестики";
}
else if (array[6] == 1 && array[7] == 1 && array[8] == 1)
{
    btn[6].BackColor = System.Drawing.Color.Red;
    btn[7].BackColor = System.Drawing.Color.Red;
    btn[8].BackColor = System.Drawing.Color.Red;
    for (int i = 0; i < btn.Length; i++)
    {
        btn[i].Enabled = false;
    }
    button10.Enabled = true;
    label1.Text = "Выиграли крестики";
}
else if (array[0] == 1 && array[3] == 1 && array[6] == 1)
{
    btn[0].BackColor = System.Drawing.Color.Red;
    btn[3].BackColor = System.Drawing.Color.Red;
    btn[6].BackColor = System.Drawing.Color.Red;
    for (int i = 0; i < btn.Length; i++)
    {
         btn[i].Enabled = false;
    }
    button10.Enabled = true;
    label1.Text = "Выиграли крестики";
}
else if (array[1] == 1 && array[4] == 1 && array[7] == 1)
{
    btn[1].BackColor = System.Drawing.Color.Red;
    btn[4].BackColor = System.Drawing.Color.Red;
    btn[7].BackColor = System.Drawing.Color.Red;
    for (int i = 0; i < btn.Length; i++)
    {
        btn[i].Enabled = false;
    }
    button10.Enabled = true;
    label1.Text = "Выиграли крестики";
}
else if (array[2] == 1 && array[5] == 1 && array[8] == 1)
{
    btn[2].BackColor = System.Drawing.Color.Red;
    btn[5].BackColor = System.Drawing.Color.Red;
    btn[8].BackColor = System.Drawing.Color.Red;
    for (int i = 0; i < btn.Length; i++)
    {
```

```
            btn[i].Enabled = false;
        }
        button10.Enabled = true;
        label1.Text = "Выиграли крестики";
    }
    else if (array[0] == 1 && array[4] == 1 && array[8] == 1)
    {
        btn[0].BackColor = System.Drawing.Color.Red;
        btn[4].BackColor = System.Drawing.Color.Red;
        btn[8].BackColor = System.Drawing.Color.Red;
        for (int i = 0; i < btn.Length; i++)
        {
            btn[i].Enabled = false;
        }
        button10.Enabled = true;
        label1.Text = "Выиграли крестики";
    }
    else if (array[2] == 1 && array[4] == 1 && array[6] == 1)
    {
        btn[2].BackColor = System.Drawing.Color.Red;
        btn[4].BackColor = System.Drawing.Color.Red;
        btn[6].BackColor = System.Drawing.Color.Red;
        for (int i = 0; i < btn.Length; i++)
        {
            btn[i].Enabled = false;
        }
        button10.Enabled = true;
        label1.Text = "Выиграли крестики";
    }
    else if (array[0] == 2 && array[1] == 2 && array[2] == 2)
    {
        btn[0].BackColor = System.Drawing.Color.Green;
        btn[1].BackColor = System.Drawing.Color.Green;
        btn[2].BackColor = System.Drawing.Color.Green;
        for (int i = 0; i < btn.Length; i++)
        {
            btn[i].Enabled = false;
        }
        button10.Enabled = true;
        label1.Text = "Выиграли нолики";
    }
    else if (array[3] == 2 && array[4] == 2 && array[5] == 2)
    {
        btn[3].BackColor = System.Drawing.Color.Green;
        btn[4].BackColor = System.Drawing.Color.Green;
        btn[5].BackColor = System.Drawing.Color.Green;
        for (int i = 0; i < btn.Length; i++)
        {
            btn[i].Enabled = false;
        }
        button10.Enabled = true;
        label1.Text = "Выиграли нолики";
    }
    else if (array[6] == 2 && array[7] == 2 && array[8] == 2)
    {
        btn[6].BackColor = System.Drawing.Color.Green;
```

```
        btn[7].BackColor = System.Drawing.Color.Green;
        btn[8].BackColor = System.Drawing.Color.Green;
        for (int i = 0; i < btn.Length; i++)
        {
            btn[i].Enabled = false;
        }
        button10.Enabled = true;
        label1.Text = "Выиграли нолики";
    }
    else if (array[0] == 2 && array[3] == 2 && array[6] == 2)
    {
        btn[0].BackColor = System.Drawing.Color.Green;
        btn[3].BackColor = System.Drawing.Color.Green;
        btn[6].BackColor = System.Drawing.Color.Green;
        for (int i = 0; i < btn.Length; i++)
        {
            btn[i].Enabled = false;
        }
        button10.Enabled = true;
        label1.Text = "Выиграли крестики";
    }
    else if (array[1] == 2 && array[4] == 2 && array[7] == 2)
    {
        btn[1].BackColor = System.Drawing.Color.Green;
        btn[4].BackColor = System.Drawing.Color.Green;
        btn[7].BackColor = System.Drawing.Color.Green;
        for (int i = 0; i < btn.Length; i++)
        {
            btn[i].Enabled = false;
        }
        button10.Enabled = true;
        label1.Text = "Выиграли нолики";
    }
    else if (array[2] == 2 && array[5] == 2 && array[8] == 2)
    {
        btn[2].BackColor = System.Drawing.Color.Green;
        btn[5].BackColor = System.Drawing.Color.Green;
        btn[8].BackColor = System.Drawing.Color.Green;
        for (int i = 0; i < btn.Length; i++)
        {
            btn[i].Enabled = false;
        }
        button10.Enabled = true;
        label1.Text = "Выиграли нолики";
    }
    else if (array[0] == 2 && array[4] == 2 && array[8] == 2)
    {
        btn[0].BackColor = System.Drawing.Color.Green;
        btn[4].BackColor = System.Drawing.Color.Green;
        btn[8].BackColor = System.Drawing.Color.Green;
        for (int i = 0; i < btn.Length; i++)
        {
            btn[i].Enabled = false;
        }
        button10.Enabled = true;
        label1.Text = "Выиграли нолики";
```

```
        }
        else if (array[2] == 2 && array[4] == 2 && array[6] == 2)
        {
            btn[2].BackColor = System.Drawing.Color.Green;
            btn[4].BackColor = System.Drawing.Color.Green;
            btn[6].BackColor = System.Drawing.Color.Green;
            for (int i = 0; i < btn.Length; i++)
            {
                btn[i].Enabled = false;
            }
            button10.Enabled = true;
            label1.Text = "Выиграли нолики";
        }
    }

    private void button10_Click(object sender, EventArgs e)
    {
        for(int i=0; i<btn.Length;i++)
        {
            array[i] = 0;
            btn[i].Text = "";
            if (rnd.Next(1, 10) <= 5)
            {
                t = "Крестики";
                label1.Text = "Ходят X";
            }
            else
            {
                t = "Ходят 0";
                label1.Text = "Ходят 0";
            }
            btn[i].Enabled = true;
            btn[i].BackColor = System.Drawing.Color.White;
        }
    }
}
```

6. Протестировать работу программы (рис. 9.2).

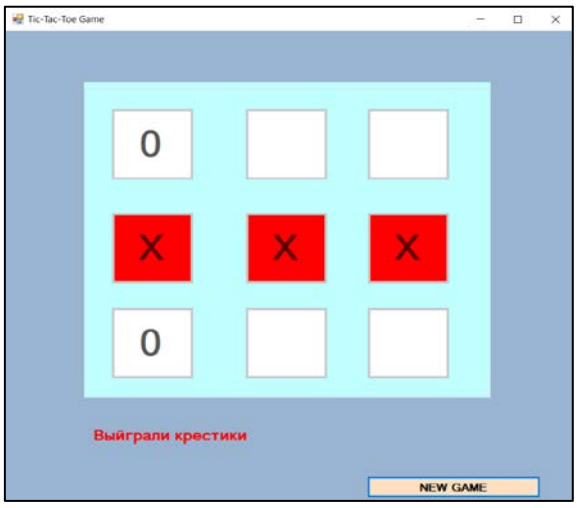

Рисунок 9.2 – Окно программы при запуске

Содержание отчета

Предоставить программу, написанную в Visual Studio на языке C# и результаты ее работы.

Контрольные вопросы

1. Для чего служат контейнеры в Windows Forms?
2. Назовите, какие Вы знаете контейнеры в Windows Forms?
3. При помощи каких свойств можно изменить цвет кнопок и размер шрифта?
4. Поясните алгоритм работы программы.
5. За что отвечает переменная, объявленная в строке: Random rnd = new Random();?

ЛИТЕРАТУРА

1. Windows Forms C# – уроки создания приложения на itProger / [Электронный ресурс] // Сообщество программистов и онлайн-школа IT профессий itProger. – Режим доступа: https://itproger.com/course/csharp-app. – Дата доступа: 17.04.2024.

2. Боресков, А. В. Основы компьютерной графики : учебник и практикум / А. В. Боресков, Е. В. Шикин ; Московский государственный университет имени М. В. Ломоносова. – Москва : Юрайт, 2020. – 219 с.

3. Колошкина, И. Е. Компьютерная графика : учебник и практикум / И. Е. Колошкина, В. А. Селезнев, С. А. Дмитроченко. – 3-е изд., испр. и доп. – Москва : Юрайт, 2021. – 233 с.

4. Культин, Н. Б. Основы программирования в Microsoft Visual C# / Н. Б. Культин. – Санкт-Петербург: БХВ, 2011. – 368 с.

Учебное издание

РАЗРАБОТКА МУЛЬТИМЕДИЙНЫХ ПРИЛОЖЕНИЙ. ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ C#

Методические указания
по выполнению лабораторных работ

Составители:
Дунина Елена Брониславовна
Соколова Анна Сергеевна

Редактор *Р.А. Никифорова*
Компьютерная верстка *А.С. Соколова*

Подписано к печати 24.05.2024. Усл. печ. листов 3,6.
Уч.-изд. листов 4,3. Заказ № 136.

Учреждение образования «Витебский государственный технологический университет»
210038, г. Витебск, Московский пр., 72.
Отпечатано на ризографе учреждения образования
«Витебский государственный технологический университет».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 1/172 от 12 февраля 2014 г.
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 3/1497 от 30 мая 2017 г.