Графика и Таймер в Window Forms.

Как занести рисунок из файла на элемент из панели элементов.

Установим на кнопке изображение. Для этого выберем кнопку и в окне **Свойств** нажмем на поле **Image.** Нам откроется диалоговое окно установки изображения (рис.2.9). В этом окне выберем опцию **Local Resource** и нажмем на кнопку **Import**, после чего нам откроется диалоговое окно для выбора файла изображения. После выбора изображения мы можем установить свойство **ImageAlign**, которое управляет позиционированием изображения на кнопке. Нам доступны 9 вариантов, с помощью которых мы можем прикрепить изображение к определенной стороне кнопки. Оставим здесь значение по умолчанию - **MiddleCenter**, то есть позиционирование по центру.Затем перейдем к свойству **TextImageRelation** и установим для него значение **ImageBeforeText.** В итоге мы получим кнопку, где сразу после изображения идет

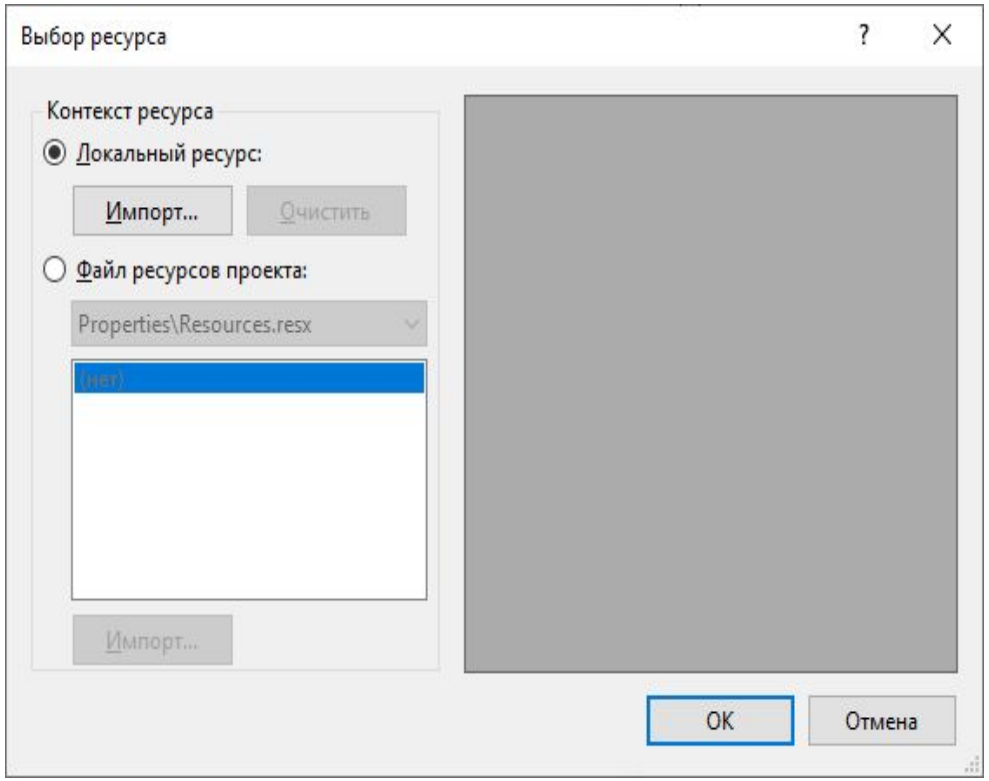


Рис.2.9. Окно установки изображения

Графические поверхности

Перед тем как начать рисовать линии, фигуры, выводить текст и манипулировать другими графическими объектами, необходимо создать графическую поверхность. Это правило действует во всех языках программирования. В библиотеке pygame языка Python – это **Surface**, в библиотеке того же языка tkinter – **Canvas** (холст), в среде Delphi – **Canvas** и т. д.

В C# эта поверхность называется Graphics.

Создать ее можно следующими тремя способами.

Способы создания графической поверхности.

Способ 1. Обработчик события **Paint** формы в качестве параметра передает объект e типа **PaintEventArgs**. Приведем пример обработчика:

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;  // Создание графической поверхности
// Код для рисования вставляется сюда
}
```

Способ 2. С помощью класса **Image**. Для создания графической поверхности из объекта типа **Image** вызывается метод **Graphics.FromImage.**

```
                    // создается объект типа Bitmap
Bitmap myBitmap = new Bitmap("C:\temp\myPic.bmp");
Graphics g = Graphics.FromImage(myBitmap);
```

Способы создания графической поверхности.

Способ 3. Используется метод **CreateGraphics** элемента управления или формы для получения ссылки на графический объект, который позволяет рисовать в форме или по элементу управления.

```
g = this.CreateGraphics();
```

где **this** – это ссылка на форму.

В ближайшем будущем мы будем использовать третий способ.

Основные методы класса Graphics

У класса Graphics много различных методов. В следующей таблице приведены три из них:

Clear(Color)	**Очищает всю поверхность рисования и выполняет заливку поверхности указанным цветом фона.**
CopyFromScreen(Point, Point, Size)	Выполняет передачу данных о цвете, соответствующих прямоугольной области пикселей, блоками битов с экрана на поверхность рисования объекта Graphics.
Dispose()	Освобождает все ресурсы, используемые данным объектом Graphics.

Инструменты рисования

После создания графической поверхности можно рисовать линии, фигуры, поверхности. Для этого используются следующие классы:

- Класс **Pen** - используется для рисования линий, поверхностей или других геометрических представлений.
- Класс **Brush** – используется для заполнения фигур и изображений цветом.
- Класс **Font** – обеспечивает текст нужным шрифтом, при этом задается тип шрифта, размер шрифта и т.д.
- Структура **Color** -представляет различные цвета.

Перед рисованием фигур и линий необходимо задать карандаш и кисть. Карандаш задает цвет и толщину линии, кисть задает цвет и тип закраски.

Класс Pen – карандаш.

Объект карандаш можно создать следующим образом:

```
Pen myPenRed = new Pen();
```

Карандаш имеет следующие основные свойства:

Color – задает цвет карандаша. Например:

```
Color.Red   // задается красный цвет.
```

Width – задает толщину карандаша. Например:

```
myPenRed.Width = 3;
```

Можно создать карандаш следующим образом:
задавая два параметра, первый – цвет, второй – толщина:

```
Pen myPenRed = new Pen(Color.Red,3);
```

Кисть – класс Brush.

Пример:

SolidBrush myBrush1 = new SolidBrush(Color.White);

Класс	Тип кисти
SolidBrush	задает кисть сплошной заливки
TextureBrush	задает текстурную кисть, закрашивает область растровым изображением
LinearGradientBrush	задает градиентную кисть, обеспечивает плавное смешение двух цветов для получения градиентного перехода.

В дальнейшем мы в основном будем использовать класс **SolidBrush**.
Порядок работы с двумя другими классами можно найти в [].

Графические примитивы.

Метод	Действие	Пример
DrawLine(Pen, x1, y1,x2, y2)	Рисует линию. Параметр Pen определяет цвет, толщину и стиль линии; параметры x1, y1, x2, y2 — координаты точек начала и конца линии	Pen myPenGreen = new Pen(Color.Green, 3); g.DrawLine(myPenGreen, 10, 10, 350, 350);
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	Pen myPenRed = new Pen(Color.Red,3); g.DrawRectangle(myPenRed, 15, 15, 250, 250);
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	SolidBrush myBrush1 = new SolidBrush(Color.White); g.FillRectangle(myBrush1, 10, 10,100, 40);

Графические примитивы (продолжение)

Метод	Действие	Пример
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	Pen blackPen = new Pen(Color.Black); g.DrawEllipse(blackPen, 10,10,100,100);
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	SolidBrush RedBr = new SolidBrush(Color.Red); g.FillEllipse(RedBr,10,10,100,100);
DrawPolygon(Pen, P)	Рисует контур многоугольника. Параметр Pen определяет цвет, толщину и стиль линии границы многоугольника; параметр P (массив типа Point) — координаты	g.DrawPolygon(myPen, new PointF[ ] {new PointF(1, 1), new PointF (20, 10), new PointF(100, 2), new PointF(200, 50)});

Графические примитивы (продолжение)

Метод	Действие	Пример
FillPolygon(Brush, P)	Рисует закрашенный многоугольник. Параметр Brush определяет цвет и стиль закраски внутренней области многоугольника; параметр P (массив типа Point) — координаты углов многоугольника	g.FillPolygon(myBrush2, new PointF[ ] {new PointF(220, 20), new PointF(100, 100), new PointF(280, 100) });
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	Font currFont = new Font("Times New Roman", 24); g.DrawString("Тестирование шрифта... ", currFont, new SolidBrush(Color.BlueViolet), 0, 0);
DrawImage(Image, x, y)	Выводит на графическую поверхность иллюстрацию. Параметр Image определяет иллюстрацию; x и y — координату левого верхнего угла области вывода иллюстрации	Image Bit1; Bit1 = new Bitmap("C:/C#/Image1/f1.bmp"); g.DrawImage(Bit1, 10, 10, 100, 100);

Пример создания поверхности и отрисовки на ней линии и прямоугольника.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace My_Line
{
 public partial class Form1 : Form
{
public static int y = 10;
   Graphics g;  //Объявляем
                //переменную g типа Graphics
```

```
public Form1()
    {
        InitializeComponent();
        g = this.CreateGraphics();    //Создаем поверхность
на форме
    }
   private void Form1_Paint(object sender, PaintEventArgs e)
   {       //Рисуем линию с помощью Form1_Paint
        Pen myPenGreen = new Pen(Color.Green, 3);
        g.DrawLine(myPenGreen, 10, y,100, 100);
   }
  private void button1_Click(object sender, EventArgs e)
   {       //Нажимаем на кнопку и рисуем
прямоугольник
        Pen myPenGreen = new Pen(Color.Green, 3);
        g.DrawRectangle(myPenGreen, y, 100, 100, 100);
   }
 }
}
```

В результате получается следующее окно.

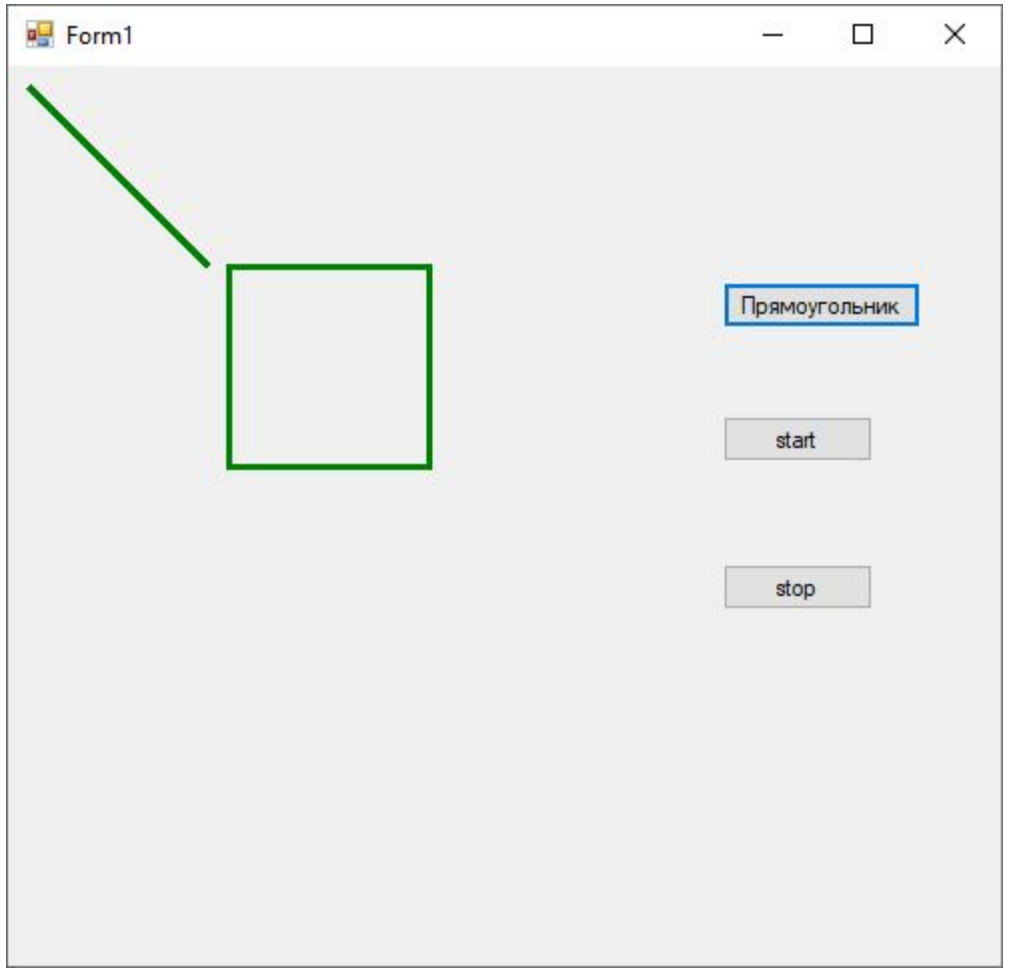


Рис. Линия и прямоугольник

Пример. Нарисуем светофор.

```
namespace My_Graph
{
    public partial class Form1 : Form
    {
        Graphics g;  //Объявляем
         //переменную g типа Graphics
        public Form1()
        {
            InitializeComponent();
            g = this.CreateGraphics();
      //Создаем поверхность на форме
        }
```

```
private void Form1_Paint(object sender, PaintEventArgs e)
     {  //Рисуем корпус и окна светофора
        SolidBrush myBrushgray = new
SolidBrush(Color.Gray);
        g.FillRectangle(myBrushgray,40, 25, 200, 400);
        SolidBrush myBrushred = new
SolidBrush(Color.Red);
        g.FillEllipse(myBrushred, 80, 25, 130, 130);
        SolidBrush myBrushYellow = new
SolidBrush(Color.Yellow);
        g.FillEllipse(myBrushYellow, 80, 155, 130, 130);
        SolidBrush myBrushGreen = new
SolidBrush(Color.Green);
        g.FillEllipse(myBrushGreen, 80, 285, 130, 130);
     }
   }
}
```

Задания для самостоятельной работы

Нарисовать:

1. ***Японский флаг.***
2. ***Российский флаг.***
3. ***Флаг Чехии.***
4. ***Олимпийский флаг***
5. ***Зимний пейзаж***

Работа с изображениями

Класс ***Image*** имеет большое количество свойств и методов для работы с изображениями. Следующие свойства задают:

Width – ширину изображения

Height – высоту изображения

Size – размер изображения.

Объявленные переменные ***Image*** присваиваются объектам класса ***Bitmap.*** Кроме того, можно создавать объекты класса ***Bitmap*** напрямую. Например, объявим переменные типа ***Image*:**

```
private Image Bit1;
private Image Bit2;
private Image Bit3;
```

Создадим объекты класса ***Bitmap***. Конструктор ***Bitmap*** возвращает объект типа ***Image***.

```
Bit1 = new Bitmap("f1.bmp")
Bit2 = new Bitmap("f2.bmp");
```

Выведем изображения на экран с помощью метода ***DrawImage*** и метода ***FormPaint:***

```
e.DrawImage(Bit1, 10, 10, 100, 100);
```

Полностью код выглядит так:

```
private void Form1_Paint(object sender, PaintEventArgs e)
 {
    Graphics p = e.Graphics;
            Image Bit1;
            Bit1 = new Bitmap("C:/C#/Image1/f1.bmp");
       p.DrawImage(Bit1, 10, 10, 100, 100);
 }
```

Как вывести рисунок из файла на форме.

Рисунок для удобства кладите в ту же папку, где находится проект. Запустить без указания полного пути не получилось. Положим на форму рисунок из файла.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace Неупр
{
   public partial class Form1 : Form
   {
      Graphics g;  // Объявляем графическую поверхность
      Rectangle rct;  //Прямоугольник с размерами рисунка
      private Image Ball;//Создаем рисунок и называем его Ball
      int x = 400, y = 100, dx = 2, dy = 2;// Переменные типа int
```

```
private void Form1_Paint(object sender, PaintEventArgs e)
    {
        g.Clear(Color.White);  //Окрашиваем форму
     //Ставим рисунок в начальное положение
        g.DrawImage(Ball, x, y, Ball.Width, Ball.Height);
    }

// Здесь Ball.Width и Ball.Height – ширина и высота
 // рисунка. Программа определяет из сама.

     public Form1()
      {
         InitializeComponent();
         g = this.CreateGraphics();   // Создаем
поверхность
         Ball = new Bitmap("Полный путь/apple.png");
                              //Создаем рисунок
      }
   }
}
```

Компонент Timer

Для реализации движения мы будем использовать компонент ***Timer.*** Интервал времени таймера будем задавать с помощью компонента ***NumericUpDown.***

Компонент ***Timer*** представляет собой компонент, который генерирует последовательность событий ***Tick.*** Другими словами, ***Timer*** – это цикл, который повторяется через заданный интервал времени. Компонент является не визуальным. Свойства компонента приведены в таблице:

Свойство	Описание
Interval	Период генерации события Tick. Задается в миллисекундах (1000 миллисекунд – 1 секунда)
Enabled	Разрешение работы. Разрешает (значение True) или запрещает (значение False) генерацию события Tick

Компонент NumericUpDown

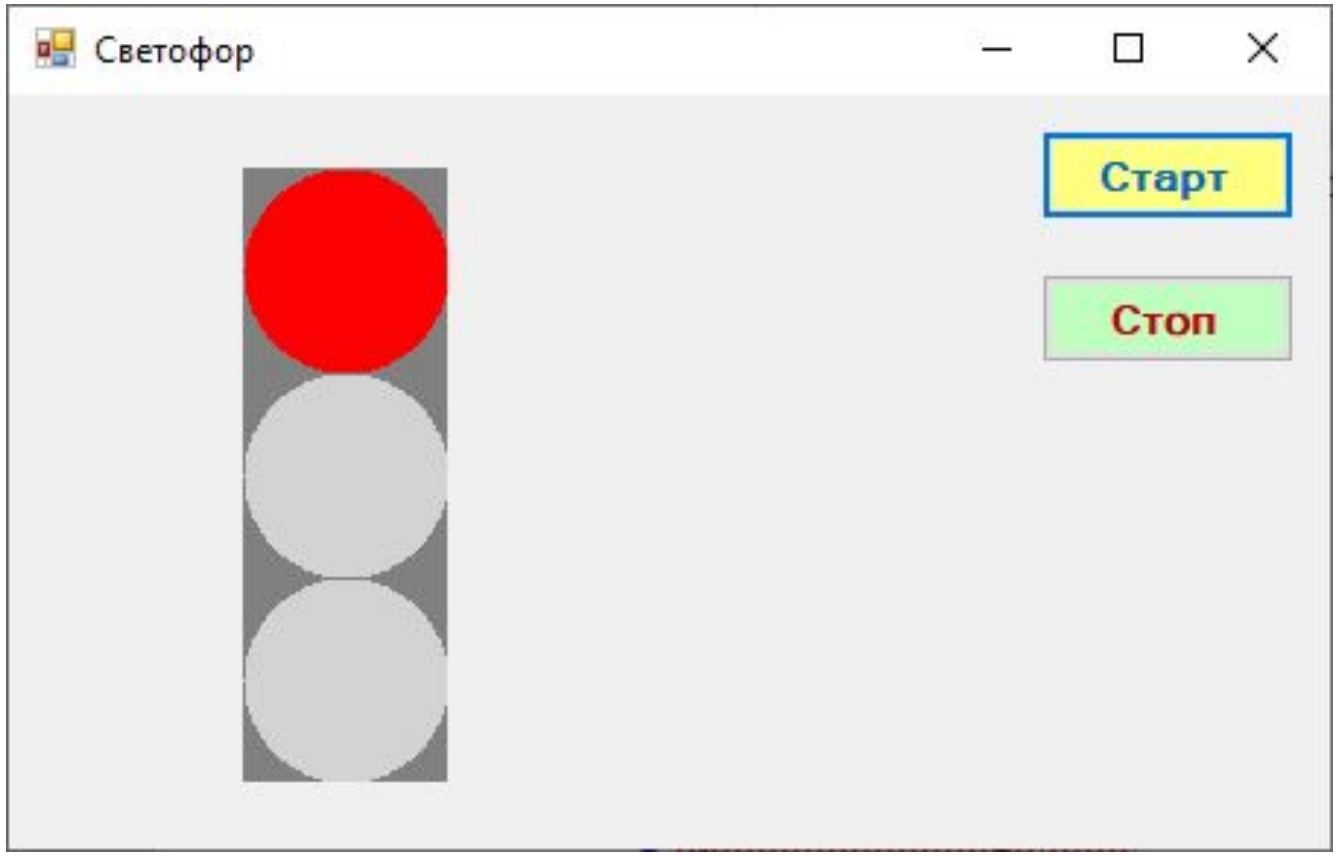
Компонент ***NumericUpDown*** предназначен для ввода числовых данных. Данные в поле редактирования можно ввести путем набора на клавиатуре или при помощи командных кнопок ***Увеличить*** и ***Уменьшить***, которые находятся справа от поля редактирования.

Свойства компонента ***NumericUpDown*** приведены в таблице:

Свойство	Описание
Value	Значение, соответствующее содержимому поля редактирования
Maximum	Максимально возможное значение, которое можно ввести в поле компонента
Minimum	Минимально возможное значение, которое можно ввести в поле компонента
Increment	Величина, на которую увеличивается или уменьшается значение свойства Value при каждом щелчке мышью на кнопках ***Увеличить*** или ***Уменьшить***

Задание: Автоматический Светофор.

Сделайте проект – светофор, работающий автоматически. Цвета чередуются. По кнопке Старт – запускается, по кнопке Стоп – выключается.



Неуправляемое движение графического объекта

//Положим на форму рисунок из файла.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace Неупр
{
   public partial class Form1 : Form
   {
      Graphics g; // Объявляем графическую
// поверхность
      Rectangle rct;//Прямоугольник с разм-ми рисунка
      private Image Ball;//Создаем рисунок  Ball
      int x = 400, y = 100, dx = 2, dy = 2;// Переменные  int
```

```
private void Form1_Paint(object sender, PaintEventArgs e)
      {
         g.Clear(Color.White);//Окрашиваем форму
     //Ставим рисунок в начальное положение
       g.DrawImage(Ball, x, y, Ball.Width, Ball.Height);
     }
   //  Здесь Ball.Width и Ball.Height –
                  //  ширина и высота рисунка.
   //  Программа определяет из сама.

       public Form1()
        {
           InitializeComponent();
           g = this.CreateGraphics(); //Создаем поверхность
           Ball = new Bitmap("Полный путь/apple.png");
           // Создаем рисунок
        }
     }
   }
```

Движение рисунка с отражением.

По форме двигается рисунок, отражаясь от краев формы. Скорость движения рисунка зависит от интервала таймера.

```
private void timer1_Tick(object sender, EventArgs e) //Так выглядит
  {                                                   // обработчик компонента Таймер.
       rct.X = x;  //Левый верхний угол прямоугольника будет иметь
                   // координаты x, y
      rct.Y = y;
      rct.Width = Ball.Width;
      rct.Height = Ball.Height;
      Invalidate(rct);  //Перерисовываем прямоугольник цветом фона
                                          // (т. е.  Стираем рисунок)
        //Invalidate очищает прямоугольник.
        //Если скобки пустые – всю форму.
      x += dx;   //Меняем координаты рисунка
      y += dy;
      g.DrawImage(Ball, x, y, Ball.Width, Ball.Height);
       //Ставим рисунок в новые координаты
       //Отражение от стенок
      if ((x < 0) || (x > (ClientRectangle.Width - Ball.Width))) dx = -dx;
      if ((y < 0) || (y > (ClientRectangle.Height - Ball.Height))) dy = -dy;
    }
```

Задание 2. «Бегущая кнопка».

Кнопка с рисунком двигается слева направо. Доходит до края формы и останавливается.

По кнопке **«Старт»** рисунок начинает движение. Если рисунок дошел до правого края формы, то он останавливается сам. Его можно запустить заново с левого края формы, если нажать кнопку **«Старт».**

Движение рисунка можно остановить в любой момент по кнопке **«Стоп».** А затем, по нажатии кнопки **«Старт»** рисунок начнёт движение заново с левого края формы.

Во время движения кнопка **«Старт»** неактивна. Если кнопка с картинкой не двигается, то кнопка **«Стоп»** неактивна.

У компонента button (кнопка) есть свойство **Enabled.** Если оно – **True,** то кнопка активна, иначе (**False**) – нет.

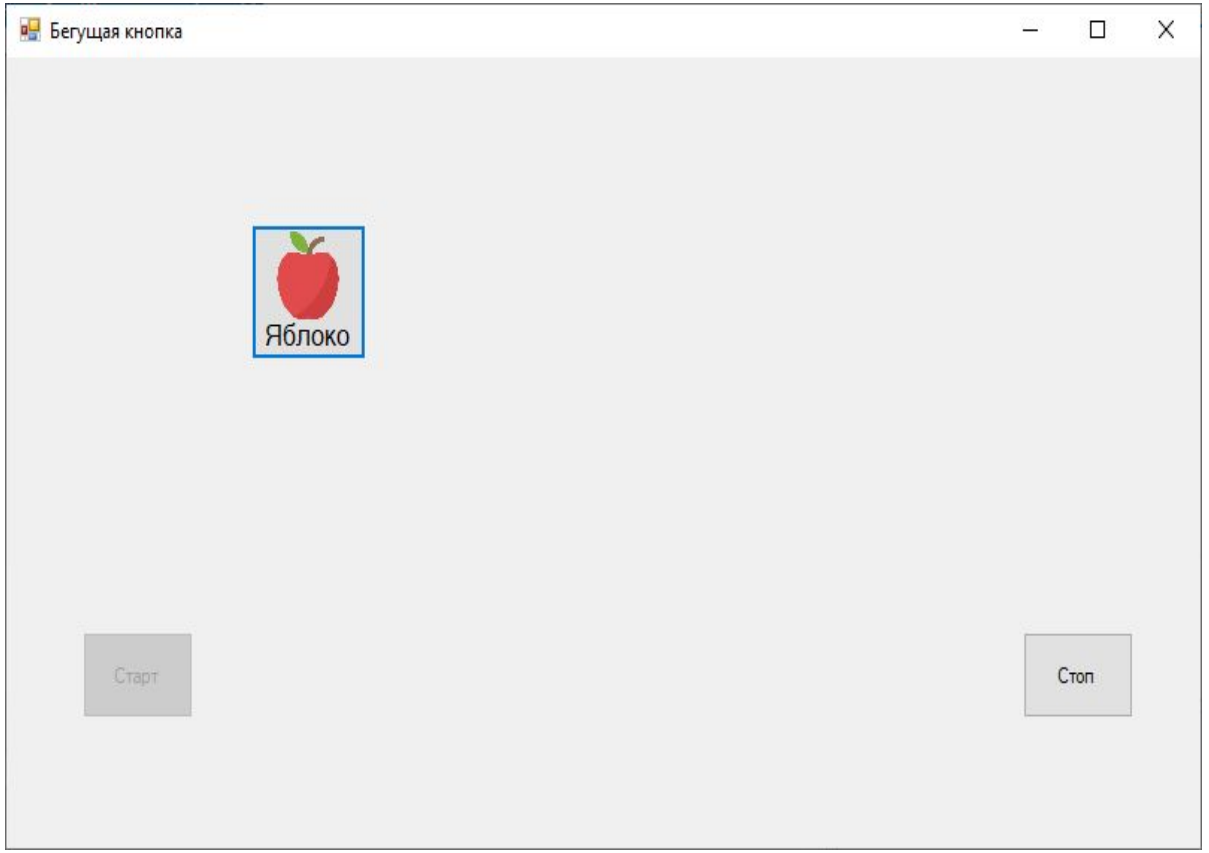


Рис. 2. Фрагмент работы проекта Бегущая кнопка

Управляемое движение объекта с помощью клавиш

Для реализации движения мы используем событие формы **Form1_KeyDown.**
Обработчику события **private void Form1_KeyDown(object sender, KeyEventArgs e)**
Через параметр **«e»** передается код нажатой клавиши. Анализируя нажатия клавиш смещаем изображение, изменяя его начальные координаты.

```
namespace Upr_dv
{
    public partial class Form1 : Form
    {
        private Image Bit;
        int x = 300, y = 100, w = 40, h = 40;

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            g.Clear(Color.White);
            g.DrawImage(Bit, x, y, w, h);
        }
```

Управляемое движение объекта с помощью клавиш

```
private void Form1_KeyDown
          (object sender, KeyEventArgs e)
  {
     rct.X = x;
     rct.Y = y;
     rct.Width = w;
     rct.Height = h;
     Invalidate(rct);
     //Обработка кодов клавиш
     if (e.KeyCode == Keys.Down) y += 10;
     if (e.KeyCode == Keys.Up) y -= 10;
     if (e.KeyCode == Keys.Left) x -= 10;
     if (e.KeyCode == Keys.Right) x +=10;
     g.DrawImage(Bit, x, y, w, h);
  }
```

```
Graphics g;
Rectangle rct;
public Form1()
 {
   InitializeComponent();
   g = this.CreateGraphics();
   Bit = new Bitmap ("D:/C#
      /My_Projects /My_im/
                          krest.png");
    }
   }
}
```

Движение объекта с помощью мыши.

Событиям ***MouseDown, MouseUp и MouseMove*** в качестве аргумента передается объект ***e***, который обеспечивает эти события текущими координатами ***(x,y)*** курсора мыши. В примере с помощью этих событий мы обеспечим движение объекта вслед за курсором мыши.

```
namespace My_im
{
    public partial class Form1 : Form
    {
        private Image Bit;
        int x = 10, y = 10, w = 40, h = 40;
        Graphics g;
        bool p = false;//Признак нажатой клавиши
        Rectangle rct;
        private void button1_Click(object sender, EventArgs e)
        {
            g = this.CreateGraphics();
            g.DrawImage(Bit, 100, 100, w, h);
        }
```

Движение объекта с помощью мыши.

```
private void Form1_MouseDown(object sender, MouseEventArgs e)
   {
       p = true;  //Нажали
   }
private void Form1_MouseUp(object sender, MouseEventArgs e)
   {
       p = false;  //Отпустили
   }
private void Form1_MouseMove(object sender, MouseEventArgs e)
   {
       if (p)  //Если нажата, то
       {
           rct.X = x ;
           rct.Y = y ;
           rct.Width = w;
           rct.Height =h;
           Invalidate(rct);
           x = e.X;
           y = e.Y;
           g.DrawImage(Bit, x, y, w, h);
       }
   }
```

```
private void Form1_Paint(object sender, PaintEventArgs e)
        {
          g.Clear(Color.White);
          g.DrawImage(Bit, x, y, w, h);
        }

public Form1()
 {
    InitializeComponent();
    g = this.CreateGraphics();
    Bit = new Bitmap("D:/C#/My_Projects
                              /My_im/krest.png");
        }
    }
}
```