# Лабораторная работа № 3.1 Построение приложений для работы с двумерной графикой

**Цель лабораторной работы**

Изучить возможности Visual Studio по созданию простейших графических изображений и анимации. Написать и отладить программу, выводящую на экран анимационное изображение

**Постановка задачи**

Класс для работы с таймером Timer формирует в приложении повторяющиеся события. События повторяются с периодичностью, указанной в миллисекундах в свойстве Interval. Установка свойства Enabled в значение true запускает таймер.

Для форм в C# предусмотрен способ, позволяющий приложению при необходимости перерисовывать окно формы в любой момент времени. Когда вся клиентская область окна формы или часть этой области требует перерисовки, форме передается событие Paint. Все, что требуется от программиста, - это создать обработчик данного события, наполнив его необходимой функциональностью.

Для рисования линий и фигур, отображения текста, вывода изображений и т.д. нужно использовать объект Graphics. Этот объект предоставляет поверхность рисования и используется для создания графических изображения. Ниже представлены два этапа работы с графикой.

Создание или получение объекта Graphics.

Рисовать возможно на любом компоненте, но быстрее всего происходит отрисовка на **pictureBox-**е.

Добавим на форму pictureBox и установим свойство Anchor: Top, Left, Right, Bottom (рисунки 1 и 2).

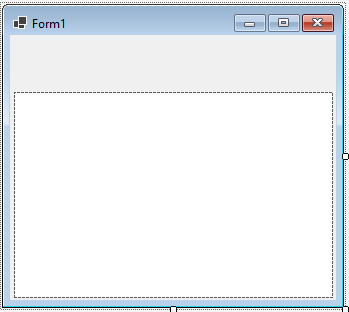


Рисунок 1 – Конструктор формы с PictureBox

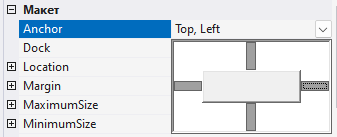


Рисунок 2 – Свойство Anchor у PictureBox

Рисование необходимо проводить в обработчике события Paint. Нарисуем круг радиуса 100 в середине полотна для рисования (рисунок 3).

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

// Для рисования необходимо создать кисть Pen.

var pen = new Pen(Color.Red, 2);

int width = 100, height = 100;

var cx = (pictureBox.Width - width) / 2;

var cy = (pictureBox.Height - height) / 2;

e.Graphics.DrawEllipse(

pen, cx, cy, width, height);

}

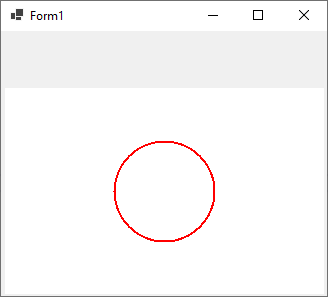


Рисунок 3 – Рисование круга в PictureBox

Нарисуем закрашенный прямоугольник шириной 100 и высотой 50 (рисунок 4). Здесь для рисования используется кисть и метод начинается не с Draw… а с Fill… (FillRectangle).

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

// Для закрашенных фигур необходима кисть

var brush = new SolidBrush(Color.Red);

int width = 100, height = 50;

var cx = (pictureBox.Width - width) / 2;

var cy = (pictureBox.Height - height) / 2;

e.Graphics.FillRectangle(

brush, cx, cy, width, height);

}

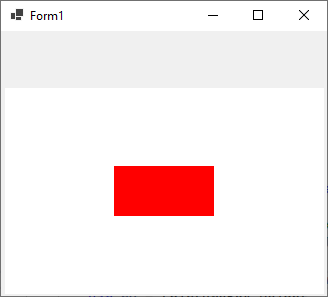


Рисунок 4 – Рисование закрашенного прямоугольника  
в PictureBox

Изменим код чтобы прямоугольник рисовался там, где была нажата ЛКМ (рисунок 5).

public partial class Form1 : Form

{

private int mouseX, mouseY; // Координаты мыши

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

// Для закрашенных фигур необходима кисть

var brush = new SolidBrush(Color.Red);

int width = 100, height = 50;

var cx = mouseX - width / 2;

var cy = mouseY - height / 2;

e.Graphics.FillRectangle(

brush, cx, cy, width, height);

}

private void pictureBox\_MouseClick(object sender, MouseEventArgs e)

{

mouseX = e.X; // Считываем координаты мыши

mouseY = e.Y;

pictureBox.Refresh(); // Для вызова отрисовки

}

}

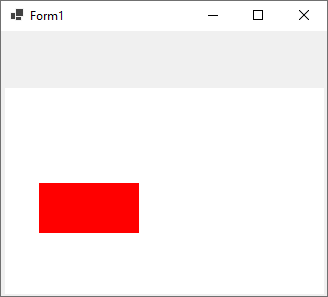


Рисунок 5 – Рисование закрашенного прямоугольника в PictureBox в точке нажатия левой кнопки мыши

Напишем код для рисования линий (рисунок 6):

public partial class Form1 : Form

{

private List<Point> points = new(); // Координаты точек

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

if (points.Count < 2) // Недостаточно точек

return;

var pen = new Pen(Color.Red, 2); e.Graphics.DrawLines(pen, points.ToArray());

}

private void pictureBox\_MouseClick(object sender, MouseEventArgs e)

{

points.Add(new(e.X, e.Y)); // Добавляем точку

pictureBox.Refresh();

}

}

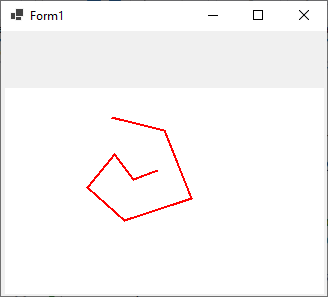


Рисунок 6 – Рисунок последовательности линий

Добавим возможность **выбора цвета** с помощью компонента ColorDialog:

private Color color = Color.Red; private void buttonColorDialog\_Click(object sender, EventArgs e)

{

var cd = new ColorDialog(); // Выбор цвета

if (cd.ShowDialog() == DialogResult.OK)

color = cd.Color;

}

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

if (points.Count < 2)

return;

var pen = new Pen(color, 2);

e.Graphics.DrawLines(pen, points.ToArray());

}

После нажатия на кнопку «Выбрать цвет» появляется диалоговое окно с палитрой (рисунок 7).

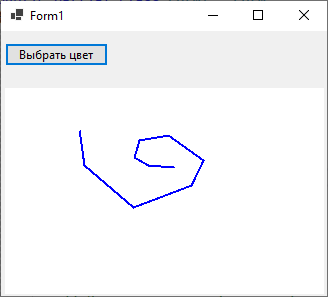
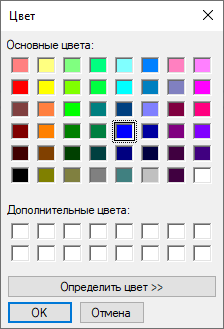
 

Рисунок 7 – Изменения цвета при рисовании

Имена большого количества методов, определенных в классе Graphics, начинаются с префикса Draw\* и Fill\*. Первые из них предназначены для рисования текста, линий и незакрашенных фигур (таких, как прямоугольные рамки), а вторые – для рисования закрашенных геометрических фигур.

Метод DrawLine рисует линию, соединяющую две точки с заданными координатами. У метода есть несколько перегруженных версий:

public void DrawLine(Pen, Point, Point);

public void DrawLine(Pen, PointF, PointF);

public void DrawLine(Pen, int, int, int, int);

public void DrawLine(Pen, float, float, float, float);

Первый параметр задает инструмент для рисования линии – перо. Перья создаются как объекты класса Pen, например:

Pen p = new Pen(Brushes.Black, 2);

Здесь создается черное перо толщиной 2 пикселя. При создании пера можно выбрать его цвет, толщину и тип линии, а также другие атрибуты.

Остальные параметры перегруженных методов DrawLine задают координаты соединяемых точек. Эти координаты могут быть заданы как объекты класса Point и PointF, а также в виде целых числе и числе с плавающей десятичной точкой.

В классах Point и PointF определены свойства X и Y, задающие, соответственно, координаты точки по горизонтальной и горизонтальной оси. При этом в классе Point эти свойства имеют целочисленные значения, а в классе PointF – значения с плавающей десятичной точкой.

Третий и четвертый варианты метода DrawLine позволяют задавать координаты соединяемых точек в виде двух пар чисел. Первая пара определяет координаты первой точки по горизонтальной и вертикальной оси, а вторая – координаты второй точки по этим же осям. Разница между третьим и четвертым методом заключается в использовании координат различных типов.

Метод DrawEllipse рисует эллипс, вписанный в прямоугольную область, расположение и размеры которой передаются ему в качестве параметров. При помощи метода DrawArc программа может нарисовать сегмент эллипса. Сегмент задается при помощи координат прямоугольной области, в которую вписан эллипс, а также двух углов, отсчитываемых в направлении против часовой стрелки. Первый угол Angle1 задает расположение одного конца сегмента, а второй Angle2 – расположение другого конца сегмента (рисунок 8).

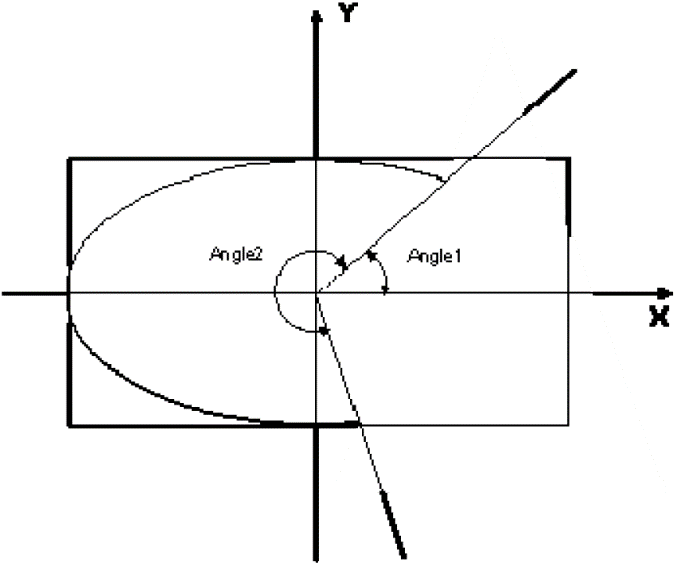


Рисунок 8 – Углы и прямоугольник, задающие сегмент эллипса

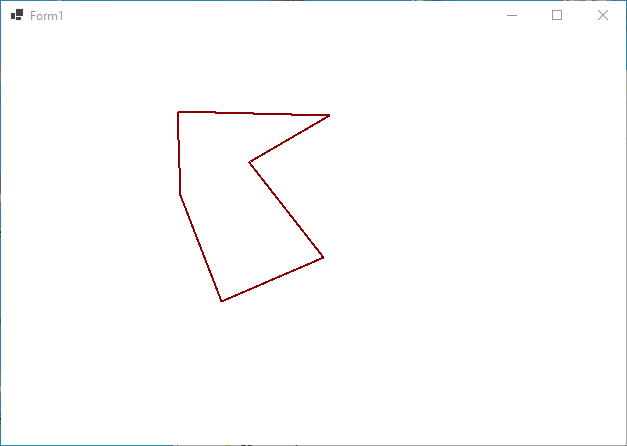


Рисунок 9 – Нарисованный многоугольник

**Создание анимации.**

Для создания простой анимации достаточно использовать таймер, при тике которого будут изменяться параметры изображения (например, координаты концов отрезка) и вызываться обработчик события Paint для рисования по новым параметрам. При таком подходе не надо заботиться об удалении старого изображения, ведь оно создается в окне заново.

Добавим таймер на форму, установим интервал 10 мс. И сразу включим **Enabled** = True. Определим основные переменные, такие как: скорость, размеры и т.д. При срабатывании таймера изменяем координаты прямоугольника, выполняем проверку на выход за границы области и рисуем.

public partial class Form1 : Form

{

private Color color = Color.Red;

private Point rectPoint = new Point(100, 100);

private int width = 100, height = 50;

private int speedX = 5, speedY = 5;

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

var brush = new SolidBrush(color);

var cx = rectPoint.X - width / 2;

var cy = rectPoint.Y - height / 2;

e.Graphics.FillRectangle(

brush, cx, cy, width, height);

}

private void timer\_Tick(object sender, EventArgs e)

{

rectPoint.X += speedX;

rectPoint.Y += speedY;

if (rectPoint.X <= 0 || rectPoint.X >= pictureBox.Width)

speedX \*= -1;

if (rectPoint.Y <= 0 || rectPoint.Y >= pictureBox.Height)

speedY \*= -1;

// Для вызова отрисовки

pictureBox.Refresh();

}

}

В результате на экране увидим двигающийся и отскакивающий от границ прямоугольник.

Движение по траектории реализуется аналогично. Для реализации движения по прямой нужно увеличивать переменные, являющиеся узловыми точками, на определенные константы: в приведенном выше примере это переменные x2 и y2. Для задания более сложной траектории можно использовать различные параметрические кривые.

В случае движения на плоскости обычно изменению подвергается один параметр. Рассмотрим пример реализации движения окружности по декартову листу. Декартов лист (рисунок 10) – это плоская кривая третьего порядка, удовлетворяющая уравнению в прямоугольной системе . Параметр 3*a* определяется как диагональ квадрата, сторона которого равна наибольшей хорде петли.

При переходе к параметрическому виду получаем:



где .

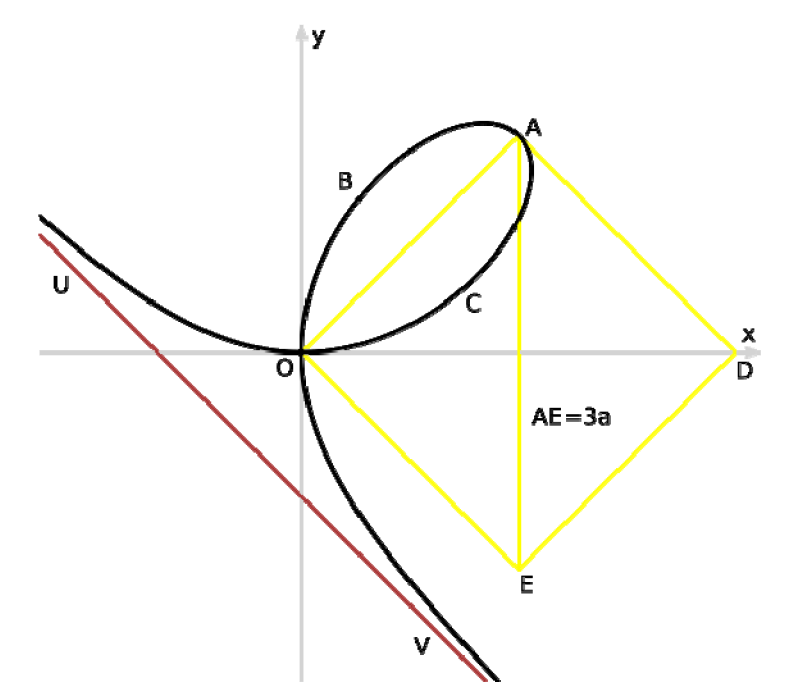


Рисунок 10 – Декартов лист

**Задание на лабораторную работу**

1. Соблюдая принципы ООП и правила проектирования приложения написать Windows-приложение в соответствии с вариантом заданий.
2. Заголовок окна приложения должен содержать ФИО исполнителя, группу и номер варианта.
3. В программе должна быть предусмотрена обработка исключений, возникающих из-за ошибочного ввода пользователя.
4. Структура программы должна быть четко поделена на графическую и модельную часть.
5. Модель должна быть инкапсулирована в отдельных от GUI классах.
6. Визуализация должна быть в классе формы.
7. Написать комментарии, поясняющие работу программы.

**Варианты заданий на лабораторную работу**

* 1. Создайте программу, показывающую пульсирующее сердце.
  2. Создайте приложение, отображающее вращающийся винт самолета.
  3. Разработайте программу анимации двигающегося человечка.
  4. Создайте программу, показывающую движение окружности по синусоиде.
  5. Создайте приложение, отображающее движение окружности по спирали.
  6. Разработайте программу анимации падения снежинки.
  7. Создайте программу, показывающую скачущий мячик.
  8. Создайте приложение, отображение движение окружности вдоль границы окна. Учтите возможность изменения размеров окна.
  9. Разработайте программу анимации летающего бумеранга.
  10. Создайте программу, показывающую падение нескольких звезд одновременно.
  11. Создайте приложение, отображающее хаотичное движение звезды в окне.
  12. Разработайте программу анимации взлета ракеты. Старт осуществляется по нажатию специальной «красной» кнопки.
  13. Создайте программу, показывающую движение окружности вдоль многоугольника. Число вершин вводится пользователем до анимации.
  14. Создайте приложение, отображающее броуновское движение молекулы в окне.
  15. Разработайте программу анимации движения планет в Солнечной системе.
  16. Создайте программу, показывающую движение квадратика по траектории, состоящей из 100 точек, хранящихся в специальном массиве.
  17. Создайте приложение, имитирующее механические часы.
  18. Разработайте программу анимации падения нескольких листков с дерева. Движение не должно быть линейным.
  19. Создайте программу. показывающую движение окружности по спирали с плавно изменяющейся скоростью.
  20. Создайте приложение, отображающее движение автомобиля с вращающимися колесами.