**Практическая работа «Введение в графику»**

1. **Рисование графических изображений**

**Событие Paint**

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.

**Объект Graphics для рисования**

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

1. Создание или получение объекта Graphics

2. Использование объекта Graphics для рисования

Существует несколько способов создания объектов Graphics. Одним из самых используемых является получение ссылки на объект Graphics через объект PaintEventArgs при обработке события Paint формы или элемента управления, например:

private void Form1\_Paint(object sender, PaintEventArgs e)

{

Graphics g = e.Graphics;

// Далее вставляется код рисования

}

**Методы и свойства класса Graphics**

Имена большого количества методов, определенных в классе Graphics, начинаются с префикса Draw\* и Fill\*. Первый из них предназначен для рисования текста, линий и не закрашенных фигур (таких, например, как прямоугольные рамки), а второй – для рисования закрашенных геометрических фигур. Ниже рассматривается применение наиболее часто используемых методов.

Более полную информацию можно найти в документации по Visual Studio.

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 создаётся черное перо толщиной 2 пиксела. 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 чисел и чисел с плавающей десятичной точкой.

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

Третий и четвертый вариант метода DrawLine позволяет 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 (целочисленных и с плавающей десятичной точкой).

Чтобы испытать метод DrawLine в работе, создайте приложение DrawLineApp (аналогично тому, как Вы создавали предыдущее приложение). В этом приложении создайте следующий обработчик события Paint:

private void Form1\_Paint(object sender, PaintEventArgs e)

{

Graphics g = e.Graphics;

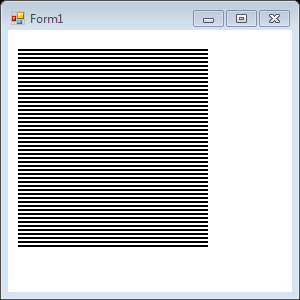
g.Clear(Color.White);

for (int i = 0; i < 50; i++)

g.DrawLine(new Pen(Brushes.Black, 2), 10, 4 \* i + 20, 200, 4 \* i + 20);

}

Здесь мы вызываем метод DrawLine в цикле, рисуя 50 горизонтальных линий, как показано на рисунке ниже.



Вызвав один раз метод DrawLines, можно нарисовать сразу несколько прямых линий, соединенных между собой. Иными словами, метод DrawLines позволяет соединить между собой несколько точек. Координаты этих точек по горизонтальной и вертикальной оси передаются методу через массив класса Point или PointF:

public void DrawLines(Pen, Point[]);

public void DrawLines(Pen, PointF[]);

Для демонстрации возможностей метода DrawLines создадим приложение. Код будет выглядеть следующим образом:

Point[] points = new Point[50];

Pen pen = new Pen(Color.Black, 2);

private void Form1\_Paint(object sender, PaintEventArgs e)

{

Graphics g = e.Graphics;

g.DrawLines(pen, points);

}

private void Form1\_Load(object sender, EventArgs e)

{

for (int i = 0; i < 20; i++)

{

int xPos;

if (i % 2 == 0)

{

xPos = 10;

}

else

{

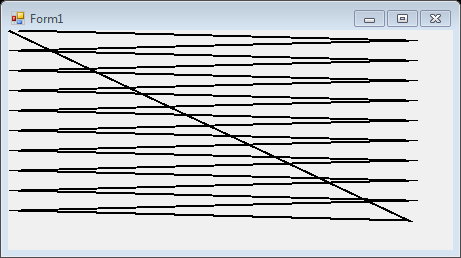
xPos = 400;

}

points[i] = new Point(xPos, 10 \* i);

}

}

Результат работы программы приведен на следующем рисунке.

Для прорисовки прямоугольников можно использовать метод DrawRectangle:

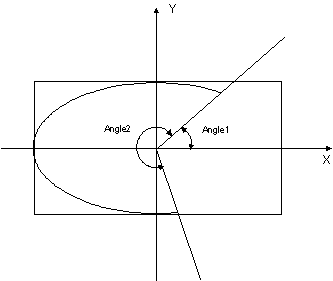
DrawRectangle(Pen, int, int, int, int);

В качестве первого параметра передается перо класса Pen. Остальные параметры задают расположение и размеры прямоугольника.

Для прорисовки многоугольников можно использовать следующий метод:

DrawPolygon(Pen, Point[]);

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 (см. рисунок ниже).



В классе Graphics определен ряд методов, предназначенных для рисования закрашенных фигур. Имена некоторых из этих методов, имеющих префикс Fill\*:

FillRectangle (рисование закрашенного прямоугольника),

FillRectangles (рисование множества закрашенных многоугольников),

FillPolygon (рисование закрашенного многоугольника),

FillEllipse (рисование закрашенного эллипса),

FillPie (рисование закрашенного сегмента эллипса),

FillClosedCurve (рисование закрашенного сплайна),

FillRegion (рисование закрашенной области типа Region).

Есть два отличия методов с префиксом Fill\* от одноименных методов с префиксом Draw\*. Прежде всего, методы с префиксом Fill\* рисуют закрашенные фигуры, а методы с префиксом Draw\* – не закрашенные. Кроме этого, в качестве первого параметра методам с префиксом Fill\* передается не перо класса Pen, а кисть класса SolidBrush. Ниже приведем пример кода программы, выводящей закрашенный прямоугольник:

SolidBrush B = new SolidBrush(Color.DeepPink);

g.FillRectangle(B, 0, 0, 100, 100);

1. Создание простейшей анимации

Работа с таймером

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. Каждый тик таймера порождает событие Tick, обработчик которого обычно и создают в приложении. В этом обработчике могут изменяться какие-либо величины, и вызываться принудительная перерисовка окна. Для создания анимации весь код, рисующий что-либо на форме, должен находиться в обработчике события Paint.

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.

В качестве примера рассмотрим код анимации секундной стрелки часов:

// Глобальные переменные

private int x1, y1, x2, y2, r;

private double a;

private Pen pen = new Pen(Color.DarkRed, 2);

// Перерисовка формы

private void Form1\_Paint(object sender, PaintEventArgs e)

{

Graphics g = e.Graphics;

// Рисуем секундную стрелку g.DrawLine(pen, x1, y1, x2, y2);

}

// Действия при загрузке формы

private void Form1\_Load(object sender, EventArgs e)

{

r = 150; // Радиус стрелки

a = 0; // Угол поворота стрелки

// Определяем центр формы – начало стрелки

x1 = ClientSize.Width / 2;

y1 = ClientSize.Height / 2;

// Конец стрелки

x2 = x1 + (int)(r \* Math.Cos(a));

y2 = y1 – (int)(r \* Math.Sin(a));

}

// Действия при очередном «тике» таймера

private void timer1\_Tick(object sender, EventArgs e)

{

a -= 0.1; // Уменьшаем угол на 0,1 радиану

// Новые координаты конца стрелки

x2 = x1 + (int)(r \* Math.Cos(a));

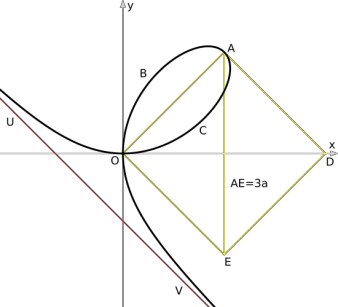
y2 = y1 – (int)(r \* Math.Sin(a));

// Принудительный вызов события Paint Invalidate();

}

**Движение по траектории**

Движение по траектории реализуется аналогично выше рассмотренному примеру. Для реализации движения по прямой нужно увеличивать переменные, являющиеся узловыми точками, на определённые константы: в приведённом 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 – это плоская кривая третьего порядка, удовлетворяющая уравнению в прямоугольной системе x3+y3=3·a·x·y. Параметр 3·a определяется как диагональ квадрата, сторона которого равна наибольшей хорде петли.

При переходе к параметрическому виду получаем:

,

где t=tg φ.

Программная реализация выглядит следующим образом:

private int x1, y1, x2, y2;

private double a, t, fi;

private Pen pen = new Pen(Color.DarkRed, 2);

private void Form1\_Load(object sender, EventArgs e)

{

x1 = ClientSize.Width / 2;

y1 = ClientSize.Height / 2;

a = 150;

fi = -0.5;

t = Math.Tan(fi);

x2 = x1 + (int)((3 \* a \* t) / (1 + t \* t \* t));

y2 = y1 – (int)((3 \* a \* t \* t) / (1 + t \* t \* t));

}

private void Form1\_Paint(object sender, PaintEventArgs e)

{

Graphics g = e.Graphics;

g.DrawEllipse(pen, x2, y2, 20, 20);

}

private void timer1\_Tick(object sender, EventArgs e)

{

fi += 0.01;

t = Math.Tan(fi);

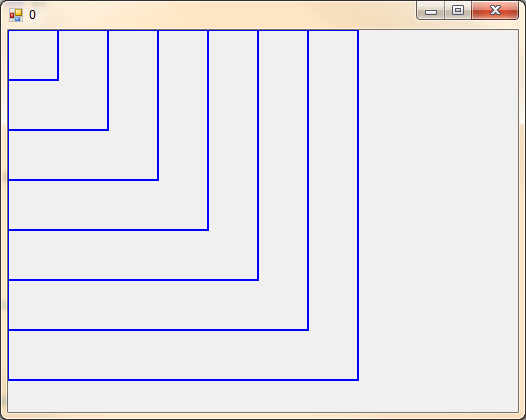
x2 = x1 + (int)((3 \* a \* t) / (1 + t \* t \* t));

y2 = y1 – (int)((3 \* a \* t \* t) / (1 + t \* t \* t));

Invalidate();

}

1. Рисование изображений с использованием рекурсии

Рассмотрим пример кода для создания набора самоподобных структур. В нашем случае это будет набор увеличивающихся квадратов.

При проектировании данной программы были созданы два метода:

private void MyDraw(Graphics g, int N, int x, int y)

{

if (N == 0)

return;

else

{

}

}

// Отрисовка прямоугольника

g.DrawRectangle(new Pen(Brushes.Blue, 2), 0, 0, x, y);

// Увеличение x и y на 50

x += 50;

y += 50;

N--;

// Рекурсивный вызов с новыми параметрами

MyDraw(g, N, x, y);

private void Form1\_Paint(object sender, PaintEventArgs e)

{

Graphics g = e.Graphics;

// Первый вызов метода и вход в рекурсию

MyDraw(g, 7, 50, 50);

}

Координаты левого верхнего угла всех прямоугольников неизменны и находятся в точке (0, 0). Поэтому в параметрах метода MyDraw достаточно передавать x и y для правого нижнего угла. Также в параметрах передается N, значение которой определяет текущую вложенность рекурсии (сколько вызовов рекурсии еще будет).

Формирование задержки с помощью таймера

Графические конструкции иногда требуется рассматривать динамически в процессе их построения. Поэтому зачастую используется такая схема вывода графики:

1. Вывод графического элемента
2. Задержка на n миллисекунд
3. Повторение этапов 1 и 2 до вывода всех графических элементов

Реализация задержки с помощью таймера возможна следующим способом:

// Глобальное поле flag

private bool flag = false;

// Далее следует часть программы,

// где необходимо организовать задержку

// Включаем таймер

timer1.Enabled = true;

// Устанавливаем flag в значение true

flag = true;

// Организуем бесконечный цикл

while (flag);

// Выключаем таймер после выхода из цикла

timer1.Enabled = false;

// Обработчик тика таймера

private void timer1\_Tick\_1(object sender, EventArgs e)

{

// Сбрасываем flag в значение false

flag = false;

}

Идея данного подхода заключается в организации бесконечного цикла, который будет проверять значение некого флага. Значение флага может измениться при наступлении события Tick таймера, то есть через заданный в таймере промежуток времени. Однако бесконечный цикл, описанный выше, останется бесконечным и программа просто зависнет. В чем же дело? Дело в том, что при такой организации цикла программа не может опросить очередь сообщений, в которое и будет поступать, в том числе, и событие Tick от таймера. Тем самым мы не попадем никогда в обработчик события timer1\_Tick\_1. Что бы решить данную проблему, надо в теле цикла написать Application.DoEvents(), что фактически будет заставлять приложение опрашивать очередь сообщений и, в свою очередь, приведет к срабатыванию обработчика события timer1\_Tick\_1.

Индивидуальные задания

|  |  |  |
| --- | --- | --- |
| Вариант | Задание 1. Создание простейшей анимации | Задание 2. Рисование  с использованием рекурсии |
| 1 | Создайте программу, показывающую пульсирующее сердце. | Напишите приложение, которое строит приведенное ниже изображение. Число рекурсий задается при первом вызове рекурсивного метода. По таймеру на каждом шаге число окружностей увеличивается в четыре раза (в рекурсивном методе происходит четыре рекурсивных вызова). |

|  |  |  |
| --- | --- | --- |
| 2 | Создайте приложение, отображающее вращающийся винт самолета. | Напишите приложение, которое строит ряд уменьшающихся окружностей по таймеру. Число окружностей удваивается на каждом шаге (в рекурсивном методе происходит два рекурсивных вызова). Число рекурсий задается при первом вызове рекурсивного метода. |
| 3 | Разработайте программу анимации двигающегося человечка. | Напишите приложение, которое строит ряд окружностей по таймеру. Число окружностей удваивается на каждом шаге (в рекурсивном методе происходит два рекурсивных вызова). Центры окружностей выбираются каждый раз произвольно (случайно). Линии связывают центры окружностей «предка» и «порожденных» от нее. Число рекурсий задается при первом вызове рекурсивного метода. |
| 4 | Создайте программу, показывающую движение окружности по синусоиде. | Напишите приложение, которое по таймеру строит ряд увеличивающихся окружностей, центры которых лежат на окружности. Число окружностей задается при первом вызове рекурсивного метода. |
| 5 | Создайте приложение, отображающее движение окружности по спирали. | Напишите приложение, которое по таймеру строит ряд увеличивающихся окружностей. Число окружностей удваивается на каждом шаге (в рекурсивном методе происходит два рекурсивных вызова). Центры окружностей выбираются каждый раз произвольно (случайно). Толщина линий также увеличивается. Число рекурсий задается при первом вызове рекурсивного метода. |

|  |  |  |
| --- | --- | --- |
| 6 | Разработайте программу анимации летающего бумеранга. | Постройте ковер Серпинского. Каждый вызов рекурсивной функции происходит по таймеру. |
| 7 | Создайте программу, показывающую скачущий мячик. | Напишите приложение, которое по таймеру строит ряд квадратов, центры которых лежат на окружности. Число квадратов задается при первом вызове рекурсивного метода. |
| 8 | Создайте приложение, отображающее движение окружности вдоль границы окна. Учтите возможность изменения размеров окна. | Напишите приложение, которое по таймеру строит ряд увеличивающихся окружностей, центры которых лежат на спирали. Число окружностей задается при первом вызове рекурсивного метода. |

|  |  |  |
| --- | --- | --- |
| 9 | Разработайте программу анимации падения снежинки. | Напишите приложение, которое по таймеру строит ряд увеличивающихся окружностей, центры которых лежат на окружности. Число окружностей задается при первом вызове рекурсивного метода. |
| 10 | Создайте программу, показывающую падение нескольких звезд одновременно. | Напишите приложение, которое по таймеру строит ряд квадратов, центры которых лежат на окружности. Число квадратов задается при первом вызове рекурсивного метода. |

|  |  |  |
| --- | --- | --- |
| 11 | Разработайте программу анимации взлета ракеты. Старт осуществляется по нажатию специальной «красной» кнопки. | Напишите приложение, которое по таймеру строит ряд окружностей. Центр окружностей совпадает с центром экрана. Число окружностей задается при первом вызове рекурсивного метода. |
| 12 | Создайте программу, показывающую движение окружности вдоль многоугольника. Число вершин вводится пользователем до анимации. | Напишите приложение, которое по таймеру строит ряд увеличивающихся окружностей по диагонали. Число окружностей задается при первом вызове рекурсивного метода. |

|  |  |  |
| --- | --- | --- |
| 13 | Создайте приложение, отображающее хаотичное движение звезды в окне. | Напишите приложение, которое по таймеру строит ряд увеличивающихся окружностей, центры которых лежат на окружности. Число окружностей задается при первом вызове рекурсивного метода. |
| 14 | Создайте приложение, имитирующие механические часы. | Напишите приложение, которое по таймеру строит ряд окружностей по диагонали. Число окружностей задается при первом вызове рекурсивного метода. |
| 15 | Разработайте программу анимации движения планет в Солнечной системе. | Постройте ковер Серпинского. Каждый вызов рекурсивной функции происходит по таймеру. |
| 16 | Разработайте программу анимации падения несколько листков с дерева. Движение не должно быть линейным. | Напишите приложение, которое по таймеру строит ряд квадратов. Центр квадратов совпадает с центром экрана. Число квадратов задается при первом вызове рекурсивного метода. |
| 17 | Создайте программу, показывающую движение окружности по спирали с плавно изменяющейся скоростью. | Напишите приложение, которое по таймеру строит ряд окружностей, центры которых лежат на окружности. Число окружностей задается при первом вызове рекурсивного метода. |
| 18 | Создайте приложение, отображающее броуновское движение молекулы в окне. | Напишите приложение, которое по таймеру строит ряд увеличивающихся окружностей по диагонали. Число окружностей задается при первом вызове рекурсивного метода. |
| 19 | Создайте программу, показывающую движение квадратика по траектории, состоящей из 100 точек, и хранящихся в специальном массиве. | Напишите приложение, которое по таймеру строит ряд увеличивающихся окружностей, центры которых лежат на спирали. Число окружностей задается при первом вызове рекурсивного метода. |
| 20 | Создайте приложение, отображающее движение автомобиля с вращающимися колесами. | Напишите приложение, которое по таймеру строит ряд квадратов. Центр квадратов совпадает с центром экрана. Число квадратов задается при первом вызове рекурсивного метода. |