Практическая работа №17_2

Тема: Разработка приложений с анимацией

Цель работы: изучение приемов создания приложений Windows Forms с анимацией

Задачи:

– изучение приемов создания приложений Windows Forms с графическими объектами;

– создание приложений с анимацией.

Материально-техническое обеспечение:

Место проведения: Компьютерный класс.

Время на выполнение работы: 2 часа.

Оборудование: ПК

Средства обучения: операционная система, текстовый процессор MS Word, программные средства определенного вида

Исходные данные:

1. Конспект занятия.
2. Задание для практической работы.

Перечень справочной литературы:

1) Программирование на языке высокого уровня. Программирование на языке С++: учеб. пособие / Т. И. Немцова, С. Ю. Голова, А. И. Терентьев ; под ред. Л. Г. Гагариной. – М. : ИД «ФОРУМ» : ИНФРА-М, 2018. – 512 с. – (Среднее профессиональное образование).

Краткие теоретические сведения:

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 каике либо величины, и вызываться принудительная перерисовка окна. Напоминаем, что вся отрисовка при создании анимации должна находиться в обработчике события Paint.

Младшее событие **timer.Tick** можно связать через обобщенный делегат **EventHandler** с **лямбда-выражением** (разновидность анонимной функции).

Тогда вместо объявления функции обработчика этого события мы можем использовать следующую конструкцию:

```
timer.Tick += new EventHandler((o, ev) =>
{
// обработка младшего события — составное лямбда-выражение
});
```

Здесь «=>» — лямбда-оператор, связывающий анонимный метод (функцию) с лямбда-выражением.

Создание анимации

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

Ход работы:

Требования к содержанию отчета:

- Номер и название практической работы.
- Цель работы.
- По каждой заданию (задаче/примеру) экранные формы (при наличии) и листинг программного кода, показывающие порядок выполнения практической работы, и результаты, полученные в ходе её выполнения.
- Ответы на контрольные вопросы в тетради.

Порядок выполнения работы:

Все проекты практической работы размещать в своей сетевой в новой папке **Пр17_2_ФИО**

В начале каждого файла проекта установить комментарии: пр.р.№_____ (указать номер), свою Фамилию. Формулировку задания

Задание 1. Анимация движения по траектории

Пояснение задачи: Движение по траектории реализуется аналогично рассмотренному заданию 3 с секундной стрелкой. Для реализации движения по прямой нужно увеличивать переменные, являющиеся узловыми точками, на определенные константы: в приведенном примере 3 это переменные *x2* и *y2*. Для задания более сложной траектории можно использовать различные параметрические кривые.

В случае движения по 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 $x^3 + y^3 = 3 \cdot a \cdot x \cdot y$. Параметр $3 \cdot a$ определяется как диагональ квадрата, сторона которого равна наибольшей хорде петли. При переходе к параметрическому виду получаем:

$$\begin{cases} x = \dfrac{3at}{1+t^3} \\ y = \dfrac{3at^2}{1+t^3} \end{cases}, \text{где } t = tg\varphi.$$

Описание ряд интересных кривых для создания траектории движения можно найти в Википедии в статье Циклоидальная кривая.

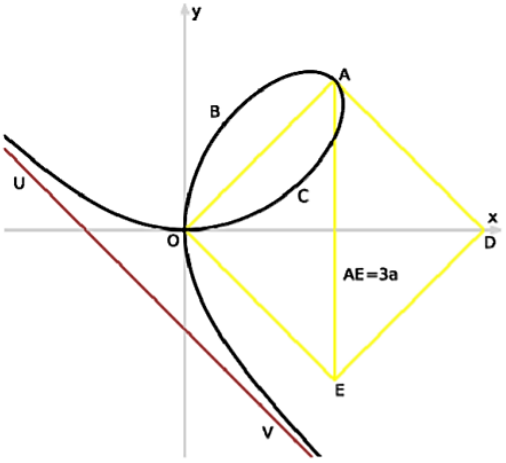
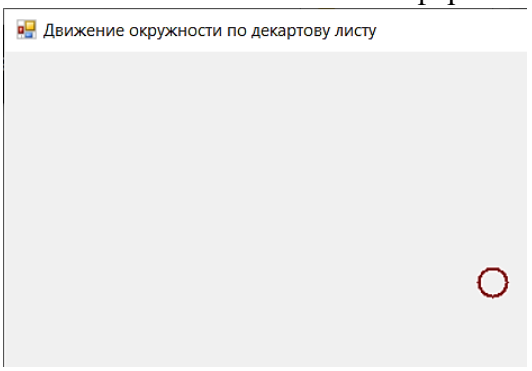


Рис. Декартов лист

1) Создайте новый проект **pr17_2.1_Фамилия** типа Windows Forms. Измените название формы.
2) Разместите на форме размером **800*800** компонент **Timer**



- ✓ выполните установку свойства **Enabled** в значение **true, Interval = 20**.
- ✓ Реализуйте обработчик события **Paint,** в котором будет рисоваться секундная стрелка соответствующего стиля

```
public partial class Form1 : Form
{
    private int x1, y1, x2, y2;
    private double a, t, fi;

    private Pen pen = new Pen(Color.DarkRed, 2);
    ссылка: 1
    public Form1()
    {
        InitializeComponent();
    }

    ссылка: 1
    private void Form1_Paint(object sender, PaintEventArgs e)
    {
        Graphics g = e.Graphics;
        g.DrawEllipse(pen, x2, y2, 20, 20);
    }
```

- ✓ Реализуйте обработчики событий **Load** и **Tick,** в которых будут изменяться параметры

изображения:

```
private void Form1_Load(object sender, EventArgs e)
{
    //определяем центр экрана
    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));
}
ссылка: 1
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();
}
```

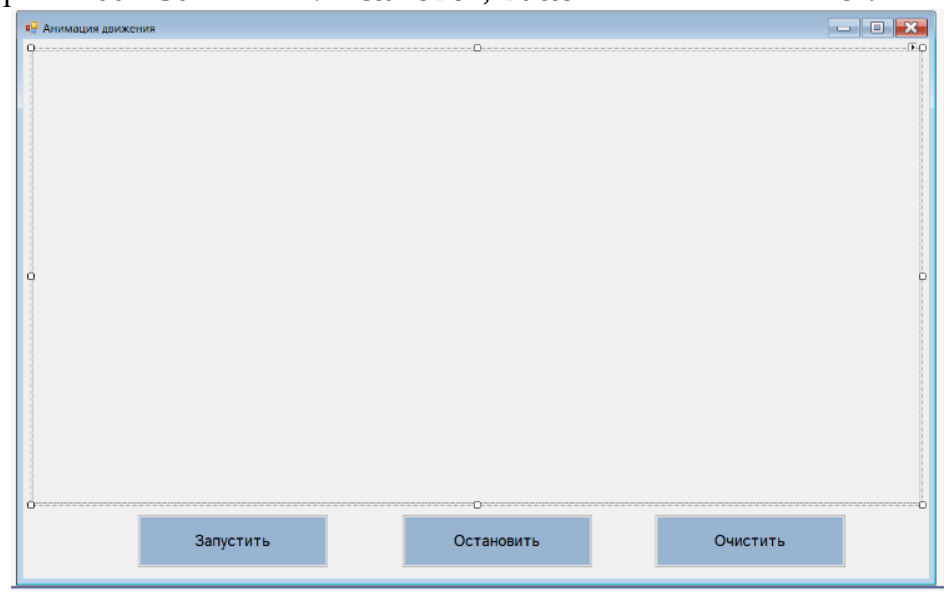
✓ Протестируйте приложение. При необходимости измените свойства компонентов и код.

Задание 2. Произвольная анимация объекта

Попробуем создать какую-нибудь простую анимацию, при помощи **PictureBox** и **Timer**, которая запустится после нажатия на кнопку **Button**. Следовательно, для этого будем обрабатывать событие нажатия на кнопку и событие "срабатывания" таймера. Также заведем все нужные для рисования объекты и переменные.

1) Создайте новый проект **pr17_2.2_Фамилия** типа Windows Forms. Измените название формы.
2) Разместите на форме размером **1200*750** объекты: **PictureBox**, **Button** и компонент **Timer:**

✓ для и компонента **Timer** выполните установку свойства **Interval = 150**.



Далее приведен код программы, который содержит все необходимое пояснение реализации анимации.

✓ Реализуйте обработчик метода группы **Draw,** в котором будет рисоваться объект анимации

```
public partial class Form1 : Form
{
    Graphics gr;        //объявляем объект - графику, на которой будем рисовать
    SolidBrush fon;     //объявляем объект - заливки, для заливки соответственно фона
    SolidBrush fig;     //и внутренности рисуемой фигуры


    int rad;           // переменная для хранения радиуса рисуемых кругов
    Random rand;       // объект, для получения случайных чисел

    Pen p = new Pen(Color.Lime);    // /объявляем объект-карандаш,
                                    // которым будем рисовать контур и задали его цвет

    ссылка: 1
    public Form1()
    {
        InitializeComponent();
    }

    //опишем функцию, которая будет рисовать круг по координатам его центра
    Ссылок: 2
    void DrawCircle(int x, int y)
    {
        int xc, yc;
        xc = x - rad;
        yc = y - rad;
        gr.FillEllipse(fig, xc, yc, rad, rad);
        gr.DrawEllipse(p, xc, yc, rad, rad);
    }
```

✓ Реализуйте обработчики событий нажатия на кнопку и "срабатывания" таймера:

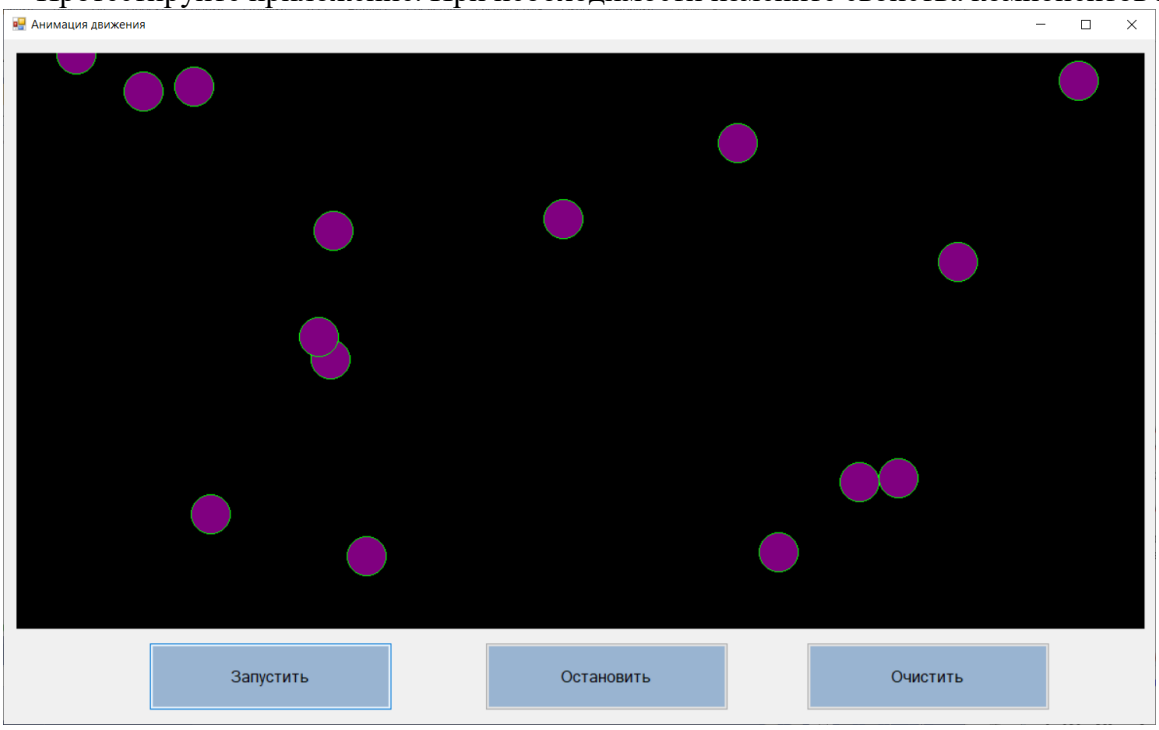
```
private void StartButton_Click(object sender, EventArgs e)
{
    gr = pictureBox1.CreateGraphics();  //инициализируем объект типа графики привязав к PictureBox
    fon = new SolidBrush(Color.Black); // и для заливки
    fig = new SolidBrush(Color.Purple);

    rad = 40;                           //задали радиус для круга
    rand = new Random();                //инициализируем объект для рандомных числе

    gr.FillRectangle(fon, 0, 0, pictureBox1.Width, pictureBox1.Height); // закрасим черным
                                                                        // нашу область рисования

    //вызываем написанную нами функцию, для прорисовки круга будем рисовать просто пятнадцать кругов
    int x, y;

    for (int i = 0; i < 15; i++)
    {
        x = rand.Next(pictureBox1.Width);
        y = rand.Next(pictureBox1.Height);
        DrawCircle(x, y);
    }

    timer1.Enabled = true;  //включим в работу наш таймер,
    // то есть теперь будет происходить событие Tick и его будет обрабатывать функция On_Tick (по умолчанию)
```

```
private void timer1_Tick(object sender, EventArgs e)
{
    //сначала будем очищать область рисования цветом фона
    gr.FillRectangle(fon, 0, 0, pictureBox1.Width, pictureBox1.Height);


    // затем опять случайным образом выбираем координаты точки и рисуем их при помощи описанной нами функции
    int x, y;

    for (int i = 0; i < 15; i++)
    {
        x = rand.Next(pictureBox1.Width);
        y = rand.Next(pictureBox1.Height);
        DrawCircle(x, y);
    }
}
```

✓ Реализуйте обработчики событий нажатия на кнопки **Остановить** и **Очистить**:

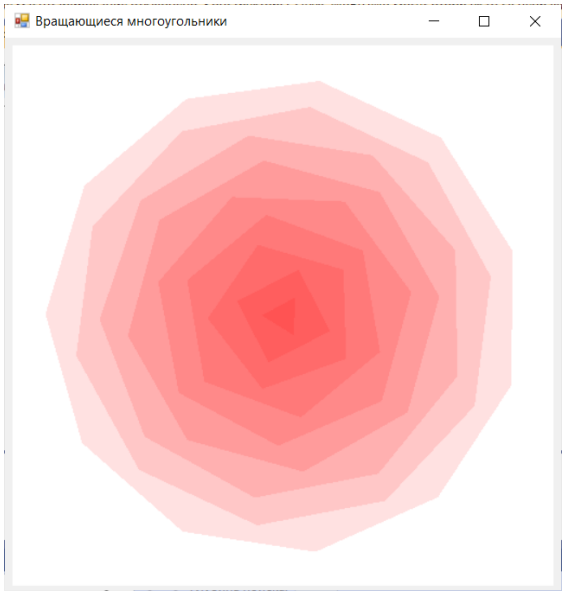
```
private void StopButton_Click(object sender, EventArgs e)
{
    timer1.Enabled = false;  //выключим работу таймера

}

ссылка: 1
private void ClearButton_Click(object sender, EventArgs e)
{
    // очистка области рисования (заливка белым цветом фона)
    fon = new SolidBrush(Color.White); // цвет заливки
    gr.FillRectangle(fon, 0, 0, pictureBox1.Width, pictureBox1.Height);
}
```

✓ Протестируйте приложение. При необходимости измените свойства компонентов и код.



Задание 3. Вращающиеся многоугольники.

1) Создайте новый проект **pr17_2.3_Фамилия** типа Windows Forms.
2) Измените заголовок формы, установите размер **530*550**:



3) Добавьте компонент **Timer,** выполните установку свойства **Enabled** в значение **False, Interval = 1**
4) Далее приведен код программы, который содержит все необходимое пояснение реализации анимации.

✓ Выполните объявление необходимых переменных и объектов:

```
public partial class Form1 : Form
{
    SolidBrush bgcol = new SolidBrush(Color.White);//цвет фона

    int cnt1 = 0; //счетчик таймера
    float speedA = 0; //переменная скорости вращения для каждого многоугольника
    float speedB = 0;
    float speedC = 0;
    float speedD = 0;
    float speedE = 0;
    float speedF = 0;
    float speedG = 0;
    float speedH = 0;
    float speedI = 0;

    ссылка: 1
    public Form1()
    {
        InitializeComponent();
    }
```

✓ Реализуйте обработчики событий **Load** и **Tick,** в которых будут увеличиваться угол вращения и запускаться процесс рисования:

```
private void Form1_Load(object sender, EventArgs e)
{
    timer1.Start();
}
```

```
private void timer1_Tick(object sender, EventArgs e)
{
    cnt1++;//увеличиваем счетчик

    speedA += (float)1.1;    //увеличиваем угол вращения с каждым шагом (для каждой фигуры разный)
    speedB += (float)1;
    speedC += (float)0.9;
    speedD += (float)0.8;
    speedE += (float)0.7;
    speedF += (float)0.6;
    speedG += (float)0.5;
    speedH += (float)0.4;
    speedI += (float)0.3;

    //по завершении анимации обнуляем значения переменных
    if (cnt1 == 3600)
    {
        cnt1 = 0;
        speedA = 0;
        speedB = 0;
        speedC = 0;
        speedD = 0;
        speedE = 0;
        speedF = 0;
        speedG = 0;
        speedH = 0;
        speedI = 0;
    }
    RotatingFigures(); //вызываем функции рисования фигур
}
```

✓ Реализуйте обработчик дополнительных методов **SetPolygon** и **RotatingFigures,** в котором будет создаваться объект анимации и настраиваться его параметры

```
// создаем функцию, которая рисует многоугольник с заданным кол-вом вершин по радиусу описанной окружности и координатам центра окружности
Ссылок: 9
private void SetPolygon(PointF center, int vertexes, float radius, Graphics graphics, Bitmap bmp, float speed)
{
    var angle = Math.PI * 2 / vertexes; //угол правильного многоугольника вычисляется по формуле 2*pi/(кол-во вершин)
    //создаем массив точек, по которым будем строить многоугольник
    var points = Enumerable.Range(0, vertexes).Select(i => PointF.Add(center, new SizeF((float)Math.Sin(i * angle) * radius, (float)Math.Cos(i * angle) * radius)));

    SolidBrush transparent = new SolidBrush(Color.FromArgb(30, 255, 0, 0));     //задаем полупрозрачную заливку

    graphics.TranslateTransform((float)pictureBox1.Width / 2, (float)pictureBox1.Height / 2);   //смещаем начало координат в центр pictureBox
    graphics.RotateTransform(speed);    //поворачиваем графику на определенный угол
    graphics.FillPolygon(transparent, points.ToArray());    //заполняем многоугольник
}
```

```
private void RotatingFigures()
{
    Bitmap btmp = new Bitmap(pictureBox1.Width, pictureBox1.Height);
    Graphics a = Graphics.FromImage(btmp); //для каждой фигуры создаем графику
    Graphics b = Graphics.FromImage(btmp);
    Graphics c = Graphics.FromImage(btmp);
    Graphics d = Graphics.FromImage(btmp);
    Graphics e = Graphics.FromImage(btmp);
    Graphics f = Graphics.FromImage(btmp);
    Graphics g = Graphics.FromImage(btmp);
    Graphics h = Graphics.FromImage(btmp);
    Graphics i = Graphics.FromImage(btmp);

    g.FillRectangle(bgcol, 0, 0, 500, 500);//заливаем фон

    SetPolygon(new PointF(0, 0), 3, 20, a, btmp, speedA); //рисуем фигуры
    SetPolygon(new PointF(0, 0), 4, 45, b, btmp, speedB);
    SetPolygon(new PointF(0, 0), 5, 70, c, btmp, speedC);
    SetPolygon(new PointF(0, 0), 6, 95, d, btmp, speedD);
    SetPolygon(new PointF(0, 0), 7, 120, e, btmp, speedE);
    SetPolygon(new PointF(0, 0), 8, 145, f, btmp, speedF);
    SetPolygon(new PointF(0, 0), 9, 170, g, btmp, speedG);
    SetPolygon(new PointF(0, 0), 10, 195, h, btmp, speedH);
    SetPolygon(new PointF(0, 0), 11, 220, i, btmp, speedI);

    pictureBox1.BackgroundImage = btmp; //переносим рисунок из буфера на pictureBox
}
```

✓ Протестируйте приложение. При необходимости измените свойства компонентов и код.