Лабораторная работа №7.1:
2D графика в проектах WPF

Цель:

Целью данной работы является получение навыков работы с двумерной графикой на языке высокого уровня C# в среде программирования Microsoft Visual Studio 2017 Community

Задание:

1) Реализовать примеры, описанные в лабораторной работе.
2) Реализовать “часы”: используя фигуры “эллипс” и “линия”, а также, объект DispatcherTimer, реализовать двумерные часы со стрелками.

Справочная информация:

Класс Shapes

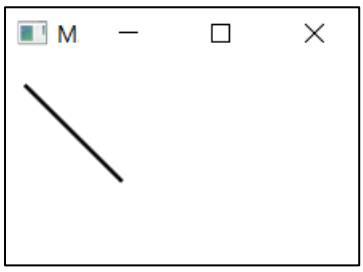
Класс, содержащий в себе реализацию простых двумерных фигур: Line (линия), Ellipse (овал), Rectangle (прямоугольник), Polygon (Многоугольник), Polyline (несколько связанных линий), Path (кривая). Для отображения объектов семейства Shapes используется компонент контейнер типа Grid или Canvas. В рамках лабораторной работы, предлагается использовать **Canvas** с именем сцена (**scene)**.

Все объекты, добавленные в сцену, размещаются в координатах этой сцены. Так, для объекта типа Canvas размером 100 на 100, верхний левый угол сцены будет иметь координаты 0:0, нижний правый угол будет иметь координаты 100:100.

Пример добавления линии в сцену:

```
//создание объекта линия
Line myLine = new Line();
//установка цвета линии
myLine.Stroke = System.Windows.Media.Brushes.Black;
//координаты начала линии
myLine.X1 = 1;
myLine.Y1 = 1;
//координаты конца линии
myLine.X2 = 50;
myLine.Y2 = 50;
//параметры выравнивания в сцене
myLine.HorizontalAlignment = HorizontalAlignment.Left;
myLine.VerticalAlignment = VerticalAlignment.Center;
//толщина линии
myLine.StrokeThickness = 2;
//добавление линии в сцену
scene.Children.Add(myLine);
```

Результат:



Примечание: любая фигура может быть исключена из сцены следующим образом:

```
//исключение фигуры из сцены
scene.Children.Remove(myLine);
```

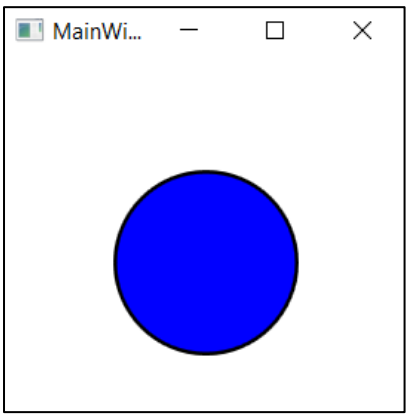
позиция фигуры может быть установлена следующим образом:

```
//позиция линии
myLine.Margin = new Thickness(50, 50, 0, 0);
```

Пример добавления в сцену окружности:

```
//создание объекта овал
Ellipse myEllipse = new Ellipse();
//создание объекта кисть
SolidColorBrush mySolidColorBrush = new SolidColorBrush();
//установка цвета в виде сочетания компонент ARGB (alpha, red, green, blue)
mySolidColorBrush.Color = Color.FromArgb(255, 255, 255, 0);
//установка объекта кисти в параметр заливки объекта овал
myEllipse.Fill = mySolidColorBrush;
//толщина и цвет обводки
myEllipse.StrokeThickness = 2;
myEllipse.Stroke = Brushes.Black;
//размеры овала
myEllipse.Width = 100;
myEllipse.Height = 100;
//позиция овала
myEllipse.Margin = new Thickness(50, 50, 0, 0);
//добавление овала в сцену
scene.Children.Add(myEllipse);
```

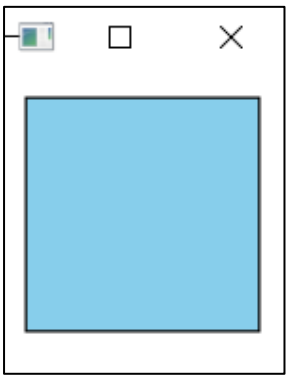
Результат:



Пример добавления в сцену прямоугольника:

```
//создание объекта прямоугольник
Rectangle myRect = new Rectangle();
//установка цвета линии обводки и цвета заливки при помощи коллекции кистей
myRect.Stroke = Brushes.Black;
myRect.Fill = Brushes.SkyBlue;
//параметры выравнивания
myRect.HorizontalAlignment = HorizontalAlignment.Left;
myRect.VerticalAlignment = VerticalAlignment.Center;
//размеры прямоугольника
myRect.Height = 50;
myRect.Width = 50;
//добавление объекта в сцену
scene.Children.Add(myRect);
```

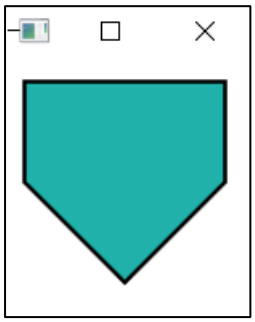
Результат:



Пример добавления в сцену многоугольника на примере пятиугольника:

```
//создание объекта многоугольник
Polygon myPolygon = new Polygon();
//установка цвета обводки, цвета заливки и толщины обводки
myPolygon.Stroke = Brushes.Black;
myPolygon.Fill = Brushes.LightSeaGreen;
myPolygon.StrokeThickness = 2;
//позиционирование объекта
myPolygon.HorizontalAlignment = HorizontalAlignment.Left;
myPolygon.VerticalAlignment = VerticalAlignment.Center;
//создание точек многоугольника
Point Point1 = new Point(0, 0);
Point Point2 = new Point(100, 0);
Point Point3 = new Point(100, 50);
Point Point4 = new Point(50, 100);
Point Point5 = new Point(0, 50);
//создание и заполнение коллекции точек
PointCollection myPointCollection = new PointCollection();
myPointCollection.Add(Point1);
myPointCollection.Add(Point2);
myPointCollection.Add(Point3);
myPointCollection.Add(Point4);
myPointCollection.Add(Point5);
//установка коллекции точек в объект многоугольник
myPolygon.Points = myPointCollection;
//добавление многоугольника в сцену
scene.Children.Add(myPolygon);
```

Результат:



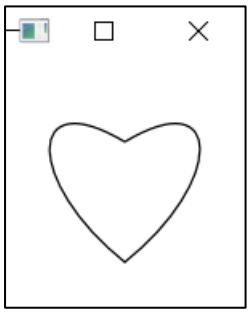
Класс Polyline работает сходим с Polygon образом, только описывает внешний контур объекта и может быть не замкнутым.

Пример добавления в сцену объекта типа Path, состоящего из двух сегментов:

```
//создание объекта путь и установка параметров его отрисовки
Path path = new Path();
path.Stroke = Brushes.Black;
path.StrokeThickness = 1;
//создание двух сегментов пути при помощи кривых Безье
```

```
//параметры - (первая контрольная точка, вторая контрольная точка, конец кривой)
BezierSegment bezierCurve1 = new BezierSegment(new Point(0, 0), new Point(0, 50), new Point(50, 90),
true);
BezierSegment bezierCurve2 = new BezierSegment(new Point(100, 50), new Point(100, 0), new Point(50,
30), true);
//создание коллекции сегментов и добавление к ней кривых
PathSegmentCollection psc = new PathSegmentCollection();
psc.Add(bezierCurve1);
psc.Add(bezierCurve2);
//создание объекта фигуры и установка начальной точки пути
PathFigure pf = new PathFigure();
pf.Segments = psc;
pf.StartPoint = new Point(50, 30);
//создание коллекции фигур
PathFigureCollection pfc = new PathFigureCollection();
pfc.Add(pf);
//создание геометрии пути
PathGeometry pg = new PathGeometry();
pg.Figures = pfc;
//создание набора геометрий
GeometryGroup pathGeometryGroup = new GeometryGroup();
pathGeometryGroup.Children.Add(pg);
//
path.Data = pathGeometryGroup;
//добавление объекта путь в сцену
scene.Children.Add(path);
```

Результат:



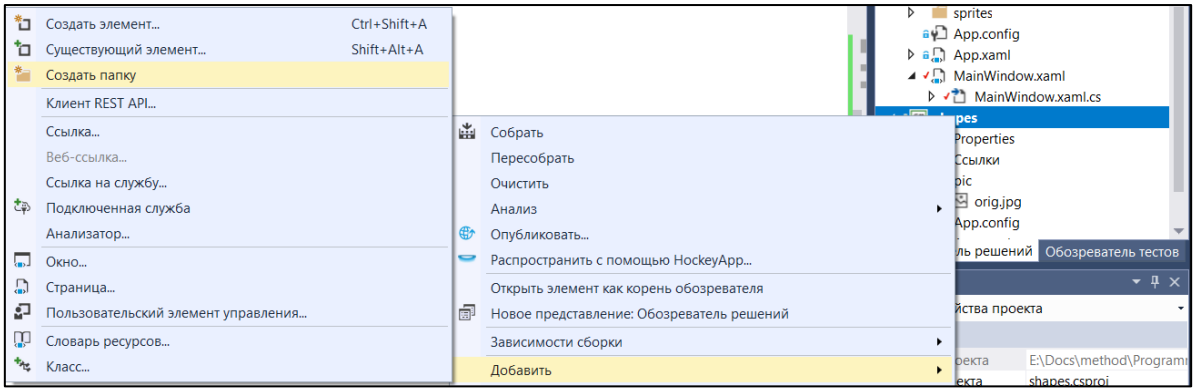
Примечание: составными частями пути могут быть линии, овалы, прямоугольники и т.д. О том, как добавить их в путь, смотрите в списке литературы.

Примечание: следует помнить, что фигуры могут перекрываться. Выше рисуется та фигура, что была добавлена последней.

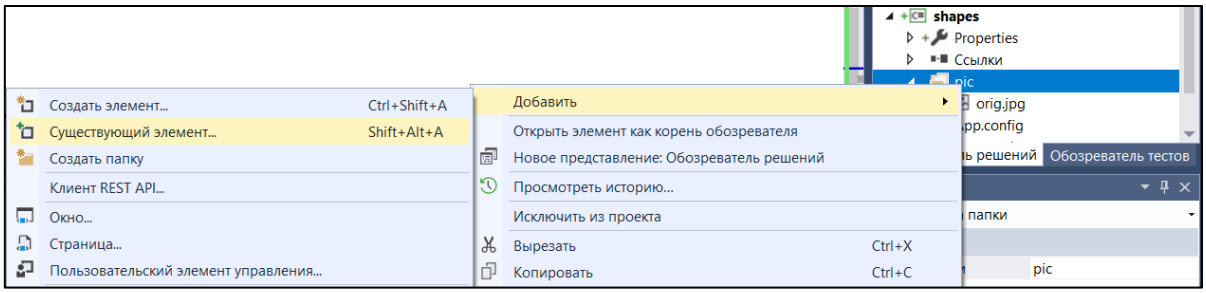
Объект ImageBrush

В приведённых выше примерах, для закраски фигур использовался сплошной цвет. Помимо этого, существуют способы задания градиента цветов и закраска объекта при помощи изображения. Второй вариант может быть осуществлён при помощи объекта кисти ImageBrush.

Для того, чтобы добавить изображение к проекту, создайте в проекте новую папку:



Затем нажмите на папку правой кнопкой мыши, выберите Добавить -> Существующий элемент:

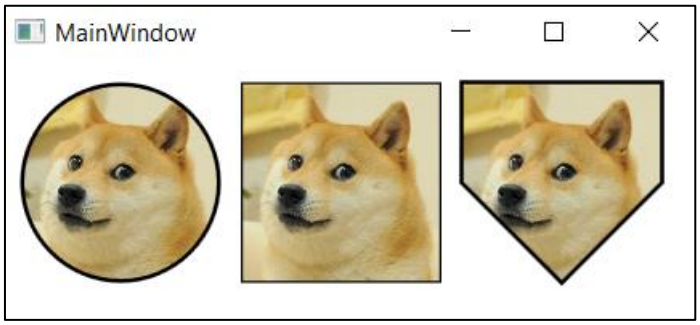


И выберите нужное изображение.

Пример использования изображения для закраски фигуры:

```
//создание объекта овал
Ellipse myEllipse = new Ellipse();
//кисть для заполнения прямоугольника изображением
ImageBrush ib = new ImageBrush();
//позиция изображения будет указана как координаты левого верхнего угла
//изображение будет растянуто по размерам прямоугольника, описанного вокруг фигуры
ib.AlignmentX = AlignmentX.Left;
ib.AlignmentY = AlignmentY.Top;
//загрузка изображения и назначение кисти
ib.ImageSource = new BitmapImage(new Uri(@"pack://application:,,,/pic/orig.jpg", UriKind.Absolute));
myEllipse.Fill = ib;
//толщина и цвет обводки
myEllipse.StrokeThickness = 2;
myEllipse.Stroke = Brushes.Black;
//размеры овала
myEllipse.Width = 100;
myEllipse.Height = 100;
//позиция овала
myEllipse.Margin = new Thickness(0, 0, 0, 0);
//добавление овала в сцену
scene.Children.Add(myEllipse);
```

Результат для фигур окружность, квадрат и многоугольник:



Примечание: изображения формата png, gif, tiff и т.д. поддерживают прозрачность.

В данном примере, изображение добавляется целиком и растягивается по размерам фигуры.

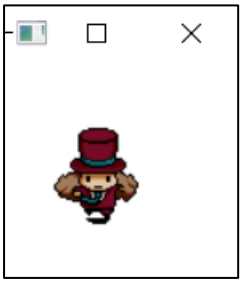
Пример овала с параметрами Width = 200 Height = 100:

Однако, объект ImageBrush позволяет настраивать параметры отображения и область изображения используемую в текущий момент. Это может быть использовано для создания системы спрайтовой анимации, на основе последовательности кадров анимации, хранимой в одном изображении.

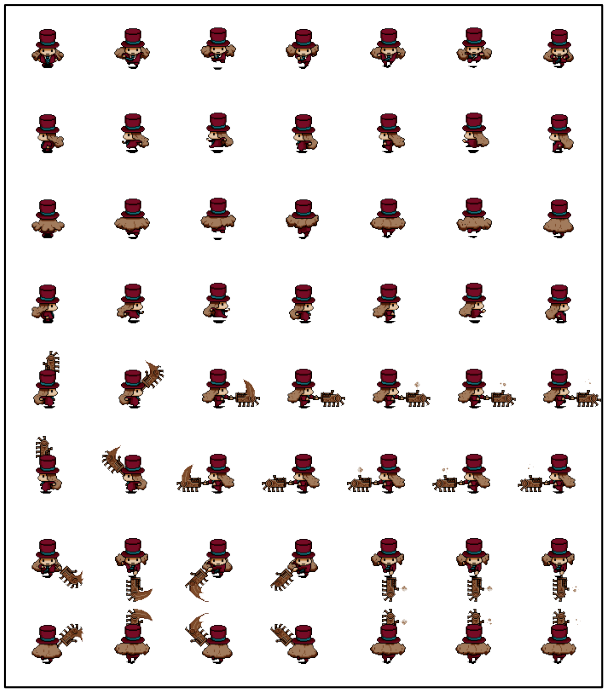
Пример использования части изображения для закраски фигуры:

```
Rectangle viky = new Rectangle(); //объект для рисования кадра анимации
//ширина и высота прямоугольника, совпадает с размерами кадра
viky.Height = 100;
viky.Width = 100;
//кисть для заполнения прямоугольника фрагментом изображения
ImageBrush ib = new ImageBrush();
//настройки, позиция изображения будет указана как координаты левого верхнего угла
//изображение будет выведено без растяжения/сжатия
ib.AlignmentX = AlignmentX.Left;
ib.AlignmentY = AlignmentY.Top;
ib.Stretch = Stretch.None;
//участок изображения который будет нарисован
//в данном случае, второй кадр первой строки
ib.Viewbox = new Rect(100, 0, 200, 100);
ib.ViewboxUnits = BrushMappingMode.Absolute;
//загрузка изображения и назначение кисти
ib.ImageSource = new BitmapImage(new Uri(@"pack://application:,,,/sprites/VictoriaSprites.gif",
UriKind.Absolute));
viky.Fill = ib;
//изначальная позиция прямоугольника
viky.Margin = new Thickness(0, 0, 0, 0);
//добавление прямоугольника в сцену
scene.Children.Add(viky);
```

Результат:



Пример набора анимаций (источник: http://ludumdare.com/compo/2013/08/24/h-bomb-sprite-sheets-of-victoria/):



Переключая значения `ib.Viewbox` по событию тика таймера, можно получить эффект анимации:

```
//определение номера текущего кадра (текущий кадр + 1 + общее число кадров) % общее число кадров
currentFrame = (currentFrame + 1 + frameCount) % frameCount;
//вычисление координат кадра - номер текущего кадра * ширина/высота одного кадра
var frameLeft = currentFrame * frameW;
var frameTop = currentRow * frameH;
//изменение отображаемого участка
(viky.Fill as ImageBrush).Viewbox = new Rect(frameLeft, frameTop, frameLeft + frameW, frameTop + frameH);
```

Перенос, поворот и масштабирование фигур

Существуют ситуации, когда размер кадра анимации или объекта на изображении не совпадают с размерами фигуры. В этом случае, к фрагменту изображения можно применить масштабирование:

```
//кисть для заполнения прямоугольника фрагментом изображения
ImageBrush ib = new ImageBrush();
...
//увеличение фрагмента изображения в 2 раза по ширине и уменьшение в 2 раза по высоте
ib.Transform = new ScaleTransform(2, 0.5);
```

Такое преобразование называется трансформацией объекта и может быть применено не только к изображению, но и к фигурам.

TranslationTransform

Одним из способов перемещения объектов, является установка свойства Margin объекта. Другим, задание перемещения при помощи матрицы переноса:

```
//изначально объект находится в точке 0:0
myRect.Margin = new Thickness(0, 0, 0, 0);
//объект переносится в точку 50:50
myRect.RenderTransform = new TranslateTransform(50, 50);
```

RotationTransform

Для того, чтобы повернуть фигуру, используется матрица поворота:

```
//изначально объект находится в точке 50:50
myRect.Margin = new Thickness(50, 50, 0, 0);
//поворот на 45 градусов вокруг точки 50:50
```

```
myRect.RenderTransform = new RotateTransform(45, 50, 50);
```

Результат:

ScaleTransform

Увеличить или уменьшить фигуру можно при помощи матрицы масштабирования:

```
//изначальный размер объекта
myRect.Height = 100;
myRect.Width = 100;
//установка
myRect.RenderTransform = new ScaleTransform(2, 0.5);
```

Результат:

SkewTransform

Последним видом преобразования, является сдвиг/наклон изображения:

```
//наклон изображения на -45 градусов относительно оси X
myRect.RenderTransform = new SkewTransform(-45, 0);
```

Результат:

TransformGroup

Описанные выше преобразования могут быть комбинированы при помощи группы преобразований:

```
//создание группы преобразований
TransformGroup tg = new TransformGroup();
//создание преобразования переноса в точку 50:50
TranslateTransform tt = new TranslateTransform(50, 50);
//создание поворота на 45 градусов вокруг точки 50:50
RotateTransform rt = new RotateTransform(45, 50, 50);
//добавление преобразований поворота и переноса в группу
tg.Children.Add(rt);
```

```
tg.Children.Add(tt);
//назначение группы для фигуры
myRect.RenderTransform = tg;
```

Примечание: порядок добавления преобразований имеет значения. Разная последовательность преобразований приводит к разным результатам.

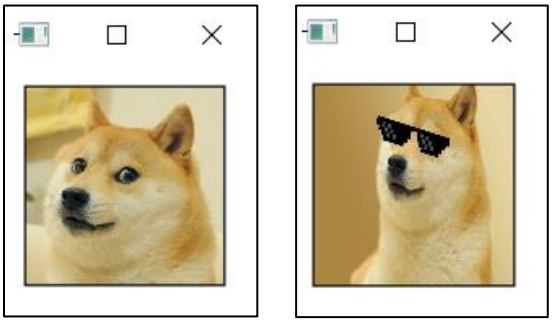
События объектов Shapes

Объекты семейства Shapes поддерживают практически тот же набор событий что и визуальные компоненты (например, Button). Это значит, при помощи фигур можно реализовать элементы управления, либо описать их реакцию на нажатие клавиш, нажатие кнопки мыши и т.д.

Пример добавления обработчика события:

```
//добавление обработчика события, которое срабатывает, когда курсор мыши попадает в область фигуры
//эта часть должна быть там же, где создаётся фигура
myRect.MouseEnter += MyRect_MouseEnter;
…
//реализация обработчика события
private void MyRect_MouseEnter(object sender, MouseEventArgs e)
{
    //создание новой кисти
    ImageBrush ib = new ImageBrush();
    //загрузка нового изображения и назначение кисти
    ib.ImageSource = new BitmapImage(new Uri(@"pack://application:,,,/pic/orig2.png",
UriKind.Absolute));
    myRect.Fill = ib;
}
```

Результат до наведения курсора на прямоугольник и после:



В некоторых случаях, может понадобится получение позиции курсора мыши в сцене (на объекте Canvas), либо на фигуре. Получить координаты мыши можно следующим образом:

```
//обработчик события окна – движение мыши, срабатывает, когда перемещается курсор мыши
private void Window_MouseMove(object sender, MouseEventArgs e)
{
    //получение координат мыши в координатах объекта Canvas с именем scene
    Point pos = Mouse.GetPosition(scene);
    //вывод координат на экран, в компоненты типа Label
    l1.Content = pos.X;
    l2.Content = pos.Y;
}
```

Обработчик события назначен для окна, а не для объекта scene потому, что для срабатывания события приписанного к объекту scene, объект scene должен быть в фокусе. То же самое касается обработки нажатия клавиш и других событий, не подразумевающих переход фокуса на объект.

В случае, если обработчики событий должны быть приписаны именно к объекту Canvas, можно использовать следующий код:

```
scene.Focusable = true;
```

это позволит переводить фокус на объект и приписанные к нему события будут срабатывать корректно.

Обнаружение пересечений

Каждая фигура содержит в себе описание ограничивающего объёма. Ограничивающий объём, это минимально возможный прямоугольник, описанный вокруг фигуры. Используя такие прямоугольники, можно обнаружить пересечение фигур, или вхождение точки в фигуру.

Пример:

```
//получение координат мыши в сцене
Point pos = Mouse.GetPosition(scene);
//получение экранных координат ограничивающего объёма для прямоугольника
Rect rect = myRect.RenderTransform.TransformBounds(myRect.RenderedGeometry.Bounds);
//проверка, находится ли курсор мыши внутри ограничивающего объёма прямоугольника
if (rect.Contains(pos) == true)
{
    MessageBox.Show("Точка входит в прямоугольник!");
}
// получение экранных координат ограничивающего объёма для эллипса
Rect ellipse = myEllipse.RenderTransform.TransformBounds(myEllipse.RenderedGeometry.Bounds);
//проверка, пересекаются ли прямоугольник и эллипс
if (rect.IntersectsWith(ellipse) == true)
{
    MessageBox.Show("Фигуры пересекаются!");
}
```

При необходимости, можно получить

```
//получение ограничивающего прямоугольника одной фигуры
Rect rect = myRect.RenderedGeometry.Bounds;
//в rect будет записана область пересечения rect и ограничивающего объёма эллипса
rect.Intersect(myEllipse.RenderedGeometry.Bounds);
```

Список литературы:

Обзор класса Shapes:
https://docs.microsoft.com/ru-ru/dotnet/framework/wpf/graphics-multimedia/shapes
Класс Polygon:
https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.shapes.polygon?view=netframework-4.8
Класс Rectangle:
https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.shapes.rectangle?view=netframework-4.8
Класс Polyline:
https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.shapes.polyline?view=netframework-4.8
Transforms Overview:
https://docs.microsoft.com/en-us/dotnet/framework/wpf/graphics-multimedia/transforms-overview
Структура Rect:
https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.rect?view=netframework-4.8
Наборы спрайтов:
https://opengameart.org/
https://itch.io/game-assets/free/tag-2d