Animação no Silverlight

José Antônio da Cunha

IFRN

Animação no Silverlight

A plataforma Silverlight permite a adição de efeitos visuais dinâmicos através da introdução de animações. Uma animação pode ser vista como uma ilusão, criada pela passagem rápida de um conjunto de imagens, onde cada uma é ligeiramente diferente da anterior.

Animação

Entendendo como a Animação funciona no Silverlight

Muitas vezes, uma animação é pensada como uma série de quadros. Para executar a animação, esses quadros são apresentados um após o outro.

Essencialmente, uma animação é uma maneira de modificar o valor de uma propriedade de dependência durante um intervalo de tempo. Por exemplo, para criar um botão que aumenta e diminui, você pode modificar sua propriedade Largura (Width).

Para um objeto brilhar, você pode alterar as propriedades do LinearGradientBrush que ele usa. O segredo para criar a animação correta é determinar quais propriedades você precisa modificar.

Animação

As regras de Animação

Você precisa estar ciente das seguintes regras fundamentais:

•**Animação do Silverlight são baseadas no tempo**. Você define o estado inicial, o estado final, e a duração de sua animação. O Silverlight calcula a taxa de quadros.
•**Animação age sobre propriedades**. A animação do Silverlight pode fazer apenas uma coisa: modificar o valor de uma propriedade em um intervalo de tempo.
•**Cada tipo de dado exige uma classe de animação diferente**. Por exemplo, a propriedade Button.Width usa o tipo de dados double. Para animá-lo, você usa a classe **DoubleAnimation**.

Animação

Criando uma animação simples

Criar uma animação é um processo de múltiplas etapas. Você precisa criar três componentes distintos: um objeto para realizar a sua animação, um storyboard para gerenciar sua animação, e um manipulador de evento (um gatilho de eventos) para iniciar o seu storyboard.

Animação

A Classe de animação

O Silverlight inclui dois tipos de classes de animação. Cada tipo de animação utiliza uma estratégia diferente para o valor de variáveis:

•**Linear interpolation**(Interpolação linear): O valor da propriedade varia suavemente e continuamente ao longo da duração da animação. O Silverlight inclui três classes: DoubleAnimation, PointAnimation e ColorAnimation.

•**Key-frame animation**: Os valores podem saltar repentinamente de um valor para outro, ou eles podem combinar saltos e períodos de interpolação linear. O silverlight inclui quatro classes: ColorAnimationUsingKeyFrames, DoubleAnimationUsingKeyFrames, PointAnimationUsingKeyFrames, e ObjectAnimationUsingKeyFrames.

Animação

As animações são definidas usando marcação XAML. Por exemplo, veja o exemplo para criar uma DoubleAnimation.

<DoubleAnimation From=“160” To=“300” Duration=“0:0:5”></DoubleAnimation>

Esta animação dura 5 segundos (conforme indicado pela propriedade Duração, que tem um valor de tempo no formato horas: minutos: Segundos.FraçãoSegundos). Enquanto a animação está sendo executada, ela muda o valor-alvo 160-300. Esta mudança ocorre de forma suave e contínua, porque o DoubleAnimation usa a interpolação linear.

Animação

A classe Storyboard

O Storyboard gere a linha do tempo da sua animação. Você pode usar um storyboard para agrupar várias animações, e também tem a capacidade de controlar a reprodução da animação pausá-la, interrompê-la e alterar sua posição.

Mas a característica mais básica fornecida pela classe Storyboard é a sua capacidade de apontar para uma determinada propriedade de um elemento específico usando as propriedades **TargetProperty**. Em outras palavras, o storyboard faz a ponte entre a animação e a propriedade que você deseja animar.

Animação

Veja um exemplo que define como um storyboard aplica um DoubleAnimation para a propriedade **Width** de um retângulo chamado “rct”.

```
<Rectangle Width="100" Height="100" Fill="Red" x:Name="rct"/>
 <StoryBoard x:Name="storyboard" Storyboard.TargetName=“rct"
                                Storyboard.TargetProperty="Width">
 <DoubleAnimation From="160" To="300" Duration="0:0:5"></DoubleAnimation>
 </StoryBoard>
```

Animação

Iniciando uma animação com evento Trigger

```
<UserControl.Triggers>
        <EventTrigger>
            <EventTrigger.Actions>
                <BeginStoryboard>
                    <Storyboard>
                        <DoubleAnimation Storyboard.TargetName=“rct"
                                         Storyboard.TargetProperty="Width"
                                         From="160" To="300" Duration="0:0:5">

                        </DoubleAnimation>
                    </Storyboard>
                </BeginStoryboard>
            </EventTrigger.Actions>
        </EventTrigger>
    </UserControl.Triggers>
    <Grid x:Name="LayoutRoot" Background="White">
        <Rectangle x:Name=“rct" Width="160" Height="30“ Fill=“Red”></Button>
    </Grid>
```

Animação

Iniciando uma animação com código

```
<UserControl.Resources>
        <Storyboard x:Name="storyboard">
            <DoubleAnimation Storyboard.TargetName="rct"
                          Storyboard.TargetProperty="Width"
                                  From="100" To="300" Duration="0:0:5">
           </DoubleAnimation>
        </Storyboard>
    </UserControl.Resources>
    <Grid x:Name="LayoutRoot" Background="White">
        <Rectangle x:Name="rct" Width="100" Height="100"/>
    </Grid>
```

```
public MainPage()
        {
            InitializeComponent();
            storyboard.Begin();
        }
```

Animação

Configurando as propriedades da animação

From – valor inicial da animação. No exemplo anterior, a animação inicia com 160 pixels.

TO – valor final da animação. No exemplo anterior, a animação inicia com 300 pixels.

By– Ao invés de usar TO, você pode usar a propriedade By. A propriedade By é usada para criar uma animação que altera um valor por um valor definido, ao invés de um alvo específico. Por exemplo, você pode criar uma animação que amplia um botão por 10 pixels mais do que seu tamanho atual, como mostrado aqui:

```
<DoubleAnimation Storeyboard.TargetName="cmdGrow" By="10"
   Storyboard.TargetProperty="Width" Duration="0:0:5"></DoubleAnimation>
```

Duration – A propriedade Duration, define o intervalo de tempo entre o momento do início da animação e o final.

Animação

RepeatBehavior

A propriedade **RepeatBehavio**r permite que você controle como uma animação se repete. Se você deseja repetir um determinado número de vezes, indicar o número de vezes para repetir, seguido por um x. Por exemplo, essa animação se repete duas vezes:

```
<DoubleAnimation Storeyboard.TargetName="rct" By="10"
            RepeatBehavior="2x"
   Storyboard.TargetProperty="Width" Duration="0:0:5"></DoubleAnimation>
```

```
<DoubleAnimation Storeyboard.TargetName="rct" By="10"
            RepeatBehavior="0:0:13" To="300"
   Storyboard.TargetProperty="Width" Duration="0:0:5"></DoubleAnimation>
```

```
<DoubleAnimation Storeyboard.TargetName="rct" By="10"
            RepeatBehavior="Forever" To="300"
   Storyboard.TargetProperty="Width" Duration="0:0:5"></DoubleAnimation>
```

Animação

Animações simultâneas

A classe Storyboard tem a capacidade de manter mais do que uma animação. Melhor de tudo, essas animações são gerenciados como um grupo. O que significa que começaram ao mesmo tempo.

```
<UserControl.Resources>
        <Storyboard x:Name="storyboard" Storyboard.TargetName=“rct">
            <DoubleAnimation Storyboard.TargetProperty="Width" To="300"
                                Duration="0:0:5"></DoubleAnimation>
            <DoubleAnimation Storyboard.TargetProperty="Height" To="300"
                                Duration="0:0:5"></DoubleAnimation>
        </Storyboard>
    </UserControl.Resources>
    <Grid x:Name="LayoutRoot" Background="White" Loaded="LayoutRoot_Loaded">
        <Button x:Name=“rct" Width="100" Height="30"></Button>
    </Grid>
```

Animação

Como controlar a reprodução

Você já viu como iniciar uma animação usando o método Storyboard.Begin(). A classe Storyboard também fornece alguns métodos a mais que permitem parar ou pausar uma animação.

```
private void cmdStart_Click(object sender, RoutedEventArgs e)
        {
            fadeStoryboard.Begin();
        }

        private void cmdPause_Click(object sender, RoutedEventArgs e)
        {
            fadeStoryboard.Pause();
        }

        private void cmdResume_Click(object sender, RoutedEventArgs e)
        {
            fadeStoryboard.Resume();
        }

        private void cmdStop_Click(object sender, RoutedEventArgs e)
        {
            fadeStoryboard.Stop();
        }
```

Animação

Easing Function

Antes de considerar as funções easing, é importante compreender quando uma função de easing é aplicada.

Cada classe (função) Easing deriva de EasingFunctionBase e herda uma propriedade única chamada EasingMode. Esta propriedade tem três valores possíveis: **EasingIn** (o que significa que o efeito é aplicado para o início da animação**), EaseOut** (que significa que é aplicada ao final), e **EaseInOu**t (que significa que é aplicada tanto no início e no final).

Animação

Easing Function

Em animações que usam interpolação linear, o crescimento e o encolhimento acontecem de forma constante e mecânica. Para um efeito mais natural, você pode adicionar a função de atenuação. O exemplo a seguir adiciona uma função de atenuação chamado ElasticEase. O resultado final é que o botão se move como uma molas, oscilando para trás e para frente, sobre seu tamanho. O movimento elástico, diminue gradualmente, até parar depois de dez oscilações. Vejam que quem controla as oscilações é a propriedade **oscillation**.

Animação

Easing Function

A easing function requer muito menos trabalho do que outras abordagens, como animação baseada em quadros. Basta você definir a propriedade EasingFunction de um objeto de animação.

```
<UserControl.Triggers>
        <EventTrigger>
            <EventTrigger.Actions>
                <BeginStoryboard>
                    <Storyboard x:Name="growStoryboard">
                        <DoubleAnimation Storyboard.TargetName="rct"
                                        Storyboard.TargetProperty="Height"
                                        To="400" Duration="0:0:5">
                            <DoubleAnimation.EasingFunction>
                                <ElasticEase EasingMode="EaseOut" Oscillations="10"></ElasticEase>
                            </DoubleAnimation.EasingFunction>
                        </DoubleAnimation>
                    </Storyboard>
                </BeginStoryboard>
            </EventTrigger.Actions>
        </EventTrigger>
    </UserControl.Triggers>
```

Animação

Easing Function (Outras propriedades)

- ✓EBackEase;
- ✓lasticEase;
- ✓BounceEase;
- ✓CircleEase;
- ✓CubicEase;
- ✓QuadraticEase;
- ✓QuarticEase;
- ✓QuinticEase;
- ✓SineEase;
- ✓PowerEase;
- ✓ExponentialEase.

Veja os exemplos no site: http://tinyurl.com/animationeasing

Animação

Transformação

Quando você usa as transformações, não apenas altera os limites de um elemento. Em vez disso, você pode alterar, inverter, distorcer, esticar, ampliar, encolher, ou rotacionar o elemento.

```
<Storyboard x:Name="rotateStoryboard">
        <DoubleAnimation Storyboard.TargetName="rotateTrans"
                         Storyboard.TargetProperty="Angle"
                         To="360" Duration="0:0:0.8" RepeatBehavior="Forever">
        </DoubleAnimation>
</Storyboard>
```

```
<Button Content="Um botão" Margin="5" RenderTransformOrigin="0.5,0.5"
                           MouseEnter="Button_MouseEnter" >
        <Button.RenderTransform>
            <RotateTransform x:Name="rotateTrans"></RotateTransform>
        </Button.RenderTransform>
</Button>
```

```
private void Button_MouseEnter(object sender, MouseEventArgs e)
    {
        rotateStoryboard.Begin();
    }
```

Animação

Transformação

```
<Storyboard x:Name="rotateStoryboard">
        <DoubleAnimation Storyboard.TargetName="projection"
                       Storyboard.TargetProperty="RotationY"
                        From="0“  To="360" Duration="0:0:3" RepeatBehavior="Forever">
         </DoubleAnimation>
         <DoubleAnimation Storyboard.TargetName="projection" RepeatBehavior="Forever"
                        Storyboard.TargetProperty="RotationZ"
                        From="0" To="360" Duration="0:0:30">
         </DoubleAnimation>
         <DoubleAnimation Storyboard.TargetName="projection" RepeatBehavior="Forever"
                        Storyboard.TargetProperty="RotationX"
                        From="0" To="360" Duration="0:0:40"></DoubleAnimation>
</Storyboard>
```

```
<Border CornerRadius="2" Padding="10" Height="140" Width="170" BorderBrush="SlateGray"
                                     BorderThickness="4">
            <Border.Projection>
                  <PlaneProjection x:Name="projection"></PlaneProjection>
            </Border.Projection>
</Border>
```

Animação

Projeções e Perspectivas

Assim como você pode animar e transformar, você também pode animar projeções em perspectiva, ou seja, a **classe PlaneProjection** permite simular uma superfície plana, título 3-D, etc. Por exemplo, imagine que você tenha um grupo de elementos envolvidos em um controle Border, e que usa uma border PlaneProjection, como mostro aqui:

```
<Grid x:Name="LayoutRoot" Background="White" Loaded="LayoutRoot_Loaded">
      <Border CornerRadius="2" Padding="10" Height="140" Width="170"
                              BorderBrush="SlateGray" BorderThickness="4">
          <Border.Projection>
              <PlaneProjection x:Name="projection"></PlaneProjection>
          </Border.Projection>
          <Button x:Name="btnOK" Width="50" Height="50" Content="OK"></Button>
      </Border>
   </Grid>
```

Veja a animação com a projeção a seguir:

Animação

Projeções e Perspectivas

```
<UserControl.Resources>
        <Storyboard x:Name="spinStoryboard">
            <DoubleAnimation Storyboard.TargetName="projection"
                             RepeatBehavior="Forever"
                             Storyboard.TargetProperty="RotationY"
                             From="0" To="360"
                             Duration="0:0:3">
            </DoubleAnimation>
            <DoubleAnimation Storyboard.TargetName="projection"
                             RepeatBehavior="Forever"
                             Storyboard.TargetProperty="RotationZ"
                             From="0" To="360"
                             Duration="0:0:30">
            </DoubleAnimation>
            <DoubleAnimation Storyboard.TargetName="projection"
                             RepeatBehavior="Forever"
                             Storyboard.TargetProperty="RotationX"
                             From="0" To="360"
                             Duration="0:0:40">
            </DoubleAnimation>
        </Storyboard>
    </UserControl.Resources>
```

Animação

Animações Brushes

Animações Brushes é uma outra técnica comum em animações em Silverlight, e é tão fácil como animar transformações. Novamente, a técnica consiste em modificar uma sub-propriedade que você deseja mudar, usando o tipo de animação adequadas.

No exemplo, o ponto central do gradiente radial deriva ao longo da elipse, dando-lhe um efeito tridimensional. Ao mesmo tempo, a cor exterior do gradiente muda de azul para preto

```
<Grid x:Name="LayoutRoot" Background="White" Loaded="LayoutRoot_Loaded">
        <Ellipse x:Name="ellipse" Margin="5" Grid.Row="1" Stretch="Uniform" >
            <Ellipse.Fill>
                <RadialGradientBrush x:Name="ellipseBrush" RadiusX="1" RadiusY="1"
                                GradientOrigin="0.7,0.3">
                    <GradientStop x:Name="ellipseBrushStop" Color="White"
                                                        Offset="0"></GradientStop>
                    <GradientStop Color="Blue" Offset="1"></GradientStop>
                </RadialGradientBrush>
            </Ellipse.Fill>
        </Ellipse>
    </Grid>
```

Animação

Animações Brushes para o exemplo anterior

```
<UserControl.Resources>
        <Storyboard x:Name="ellipseStoryboard">
            <PointAnimation Storyboard.TargetName="ellipseBrush"
                            Storyboard.TargetProperty="GradientOrigin"
                            From="0.7,0.3" To="0.3,0.7" Duration="0:0:10"
                            AutoReverse="True"
                            RepeatBehavior="Forever">
            </PointAnimation>
            <ColorAnimation Storyboard.TargetName="ellipseBrushStop"
                            Storyboard.TargetProperty="Color" To="Black"
                            Duration="0:0:10"
                            AutoReverse="True"
                            RepeatBehavior="Forever">
            </ColorAnimation>
        </Storyboard>
    </UserControl.Resources>
```

Animação

Animação Pixel Shaders

Pixel shaders é um recurso interessante, mas apenas ocasionalmente útil. Mas combinado com animação, eles se tornam muito mais versátil. Você pode usá-los para criar efeitos impressionantes interatividade com o usuário (por exemplo, aumentando o brilho em um botão quando o usuário move o mouse sobre ele). O melhor de tudo, você pode animar as propriedades de um pixel shader tão facilmente como você animar qualquer outra coisa.

Animação

Animating Brushes

```
<Storyboard x:Name="ellipseStoryboard">
        <PointAnimation Storyboard.TargetName="ellipseBrush"
                        Storyboard.TargetProperty="GradientOrigin"
                          From="0.7,0.3" To="0.3,0.7" Duration="0:0:10" AutoReverse="True"
                          RepeatBehavior="Forever">
         </PointAnimation>
          <ColorAnimation Storyboard.TargetName="ellipseBrushStop"
                        Storyboard.TargetProperty="Color"
                         To="Black" Duration="0:0:10" AutoReverse="True"
                        RepeatBehavior="Forever">
           </ColorAnimation>
</Storyboard>
```

```
<Ellipse x:Name="ellipse" Margin="5" Grid.Row="1" Stretch="Uniform" >
          <Ellipse.Fill>
              <RadialGradientBrush x:Name="ellipseBrush" RadiusX="1" RadiusY="1"
                        GradientOrigin="0.7,0.3">
                  <GradientStop x:Name="ellipseBrushStop" Color="Wheat"
                        Offset="0"></GradientStop>
                  <GradientStop Color="Blue" Offset="1"></GradientStop>
              </RadialGradientBrush>
          </Ellipse.Fill>
</Ellipse>
```

Animação

Animação por key-Frame

A animação key-Frame é uma animação que é feita de muitos pequenos segmentos. Cada segmento representa um valor inicial, final ou intermediário na animação. Quando você executa a animação, move-se suavemente de um valor para outro.

Veja o exemplo usando **PointAnimatio**.

```
<PointAnimation Storyboard.TargetName=“ellipseBrush”
                Storyboard.TargetProperty=“GradientOrigin”
                From=“0.7,0.3” To=“0.3,0.7” Duration=“0:0:10”
                AutoReverse=“True” RepeatBehavior=“Forever>
</PointAnimation>
```

Animação

O exemplo anterior usando key-Frame

```
<PointAnimationUsingKeyFrame Storyboard.TargetName="ellipseBrush"
                             Storyboard.TargetProperty="GradientOrigin"
                             AutoReverse="True" RepeatBehavior="Forever">
   <LinearPointKeyFrame Value="0.7,0.3" KeyTime="0:0:0"></LinearPointKeyFrame>
  <LinearPointKeyFrame Value="0.3,0.7" KeyTime="0:0:10"></LinearPointKeyFrame>
</PointAnimationUsingKeyFrame>
```

Esta animação inclui dois key-Frames. O primeiro define o valor dos pontos quando a animação começa. O segundo key-Frame define o valor final, que é alcançado após 10 segundos. O objeto PointAnimationUsingKeyFrames executa uma interpolação linear para mover-se suavemente a partir do valor de 1º key-Frame para o segundo, assim como faz PointAnimation para ir de From pata To.

Animação

Exemplo usando key-Frame

```
<UserControl.Resources>
        <Storyboard x:Name="ellipseStoryboard">
            <PointAnimationUsingKeyFrames Storyboard.TargetName="ellipseBrush"
                                          Storyboard.TargetProperty="GradientOrigin"
                                          AutoReverse="True" RepeatBehavior="Forever">
                <LinearPointKeyFrame Value="0.7,0.3" KeyTime="0:0:0"></LinearPointKeyFrame>
                <LinearPointKeyFrame Value="0.3,0.7" KeyTime="0:0:10"></LinearPointKeyFrame>
            </PointAnimationUsingKeyFrames>
        </Storyboard>
    </UserControl.Resources>
```

```
<Grid x:Name="LayoutRoot" Background="White" Loaded="LayoutRoot_Loaded">
        <Ellipse x:Name="ellipse" Margin="5" Grid.Row="1" Stretch="Uniform">
            <Ellipse.Fill>
                <RadialGradientBrush x:Name="ellipseBrush" RadiusX="1" RadiusY="1"
GradientOrigin="0.7,0.3">
                    <GradientStop x:Name="ellipseBrushStop" Color="White"
Offset="0"></GradientStop>
                    <GradientStop Color="Blue" Offset="1"></GradientStop>
                </RadialGradientBrush>
            </Ellipse.Fill>
        </Ellipse>
    </Grid>
```

Animação

Discrete key-Frame

A animação por key-Frame que você viu nos exemplos anteriores usava key frames linear. Como resultado, a transição entre os valores dos key-frame é suave. Outra opção é usar Discrete key frames. Neste caso, a animação é feita sem interpolação. Quando o key time é atingido, a propriedade muda abruptamente para o novo valor.

Animação

Exemplo usando Discrete key-Frame

```
<UserControl.Resources>
      <Storyboard x:Name="ellipseStoryboard">
         <PointAnimationUsingKeyFrames Storyboard.TargetName="ellipseBrush"
                                       Storyboard.TargetProperty="GradientOrigin"
                                       AutoReverse="True" RepeatBehavior="Forever">
            <DiscretePointKeyFrame Value="0.7,0.3" KeyTime="0:0:0"></DiscretePointKeyFrame>
            <DiscretePointKeyFrame Value="0.3,0.7" KeyTime="0:0:5"></DiscretePointKeyFrame>
            <DiscretePointKeyFrame Value="0.5,0.9" KeyTime="0:0:8"></DiscretePointKeyFrame>
            <DiscretePointKeyFrame Value="0.9,0.6" KeyTime="0:0:10"></DiscretePointKeyFrame>
            <DiscretePointKeyFrame Value="0.8,0.2" KeyTime="0:0:12"></DiscretePointKeyFrame>
            <DiscretePointKeyFrame Value="0.7,0.3" KeyTime="0:0:14"></DiscretePointKeyFrame>
         </PointAnimationUsingKeyFrames>
      </Storyboard>
   </UserControl.Resources>
```

Animação

Exemplo usando Easing key-Frame

```
<UserControl.Resources>
        <Storyboard x:Name="ellipseStoryboard">
            <PointAnimationUsingKeyFrames Storyboard.TargetName="ellipseBrush"
                                          Storyboard.TargetProperty="GradientOrigin"
                                          RepeatBehavior="Forever">
                <LinearPointKeyFrame Value="0.7,0.3" KeyTime="0:0:0"></LinearPointKeyFrame>
                <EasingPointKeyFrame Value="0.3,0.7" KeyTime="0:0:5">
                    <EasingPointKeyFrame.EasingFunction>
                        <CircleEase></CircleEase>
                    </EasingPointKeyFrame.EasingFunction>
                </EasingPointKeyFrame>
                <LinearPointKeyFrame Value="0.5,0.9" KeyTime="0:0:8"></LinearPointKeyFrame>
                <LinearPointKeyFrame Value="0.9,0.6" KeyTime="0:0:10"></LinearPointKeyFrame>
                <LinearPointKeyFrame Value="0.8,0.2" KeyTime="0:0:12"></LinearPointKeyFrame>
                <LinearPointKeyFrame Value="0.7,0.3" KeyTime="0:0:14"></LinearPointKeyFrame>
            </PointAnimationUsingKeyFrames>
        </Storyboard>
    </UserControl.Resources>
```

Animação

Spline key-Frame

Como os Linear Key Frames, Spline key frames usam interpolação para mover suavemente de um valor chave para o outro. A diferença é que cada spline key frame esporte uma propriedade KeySpline. Usando a propriedade KeySpline, você define uma curva de Bézier cúbica que influencia como modo de interpolação é executada.

Animação

Exemplo usando Spline key-Frame

```
<Grid x:Name="LayoutRoot" Background="White" Loaded="LayoutRoot_Loaded">
        <Canvas>
            <Canvas.Resources>
                <Storyboard x:Name="ellipseSpline">
                    <DoubleAnimationUsingKeyFrames Storyboard.TargetName="Elipse1"
                                            Storyboard.TargetProperty="(Canvas.Left)">
                        <SplineDoubleKeyFrame KeyTime="0:0:5" Value="250"
                                    KeySpline="0.25,0 0.5,0.7"></SplineDoubleKeyFrame>
                        <SplineDoubleKeyFrame KeyTime="0:0:10" Value="500"
                                    KeySpline="0.25,0.8 0.2,0.3"></SplineDoubleKeyFrame>
                    </DoubleAnimationUsingKeyFrames>
                </Storyboard>
            </Canvas.Resources>
            <Ellipse x:Name="Elipse1" Width="50" Height="50" Fill="Black" Canvas.Top="1"
Canvas.Left="1"></Ellipse>
        </Canvas>
    </Grid>
```