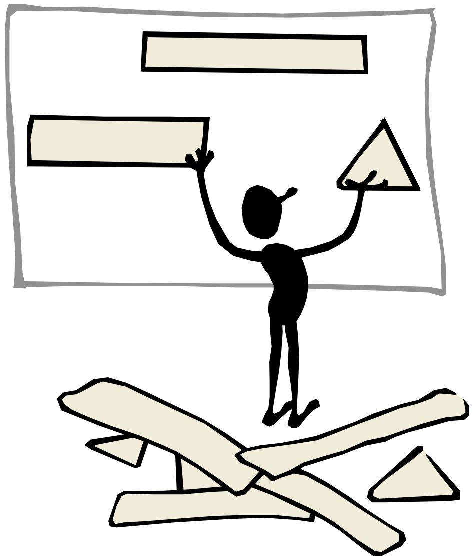
Layout in WPF



AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Agenda

- Layout Basics
- StackPanel
- Canvas
  - *Attached Properties*
- ScrollViewer
- Expander
- DockPanel
- WrapPanel
- Grid
- UniformGrid

Layout Basics

- A window can only have one child element!
  - So how can we put 2 buttons and a couple of textboxes on a window?
- We use a panel to put several controls on one window.
- Panels are responsible for the layout of the controls it contains.
- Panels comes in different flavors, each offers a straightforward and easily understood layout mechanism.
- Panel types:
  - StackPanel
  - WrapPanel
  - DockPanel
  - Grid
  - UniformGrid
  - Canvas
- By default, panels have no appearance of their own.

The Layout Contract

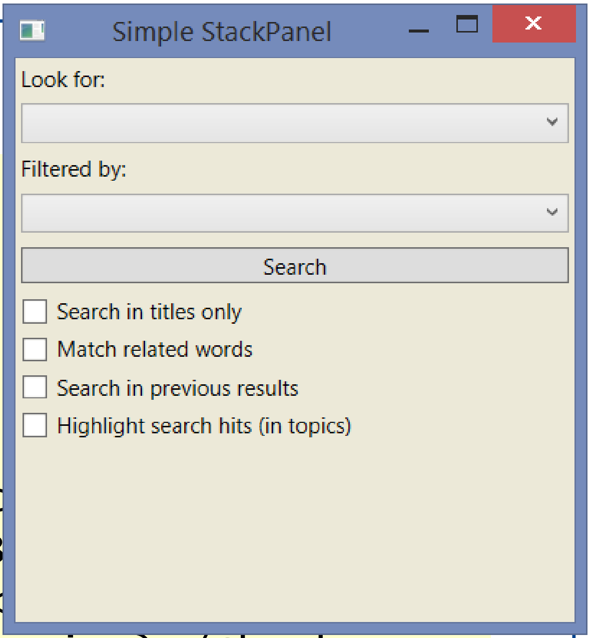
- The layout contract defines the way a panel and child controls communicate.

- Size to content
  - Is the idea that each control is expected to determine its size on the basic of its content.

- Two-Phase Layout
  - A control’s size is determined in two distinct phases: **measure** and **arrange**.
  - The **measure phase** consist of walking the entire display and asking each element to determine its desired size, given a constraint.
  - In the **arrange phase** each element is told by its parent to size itself to a specific size and location.

StackPanel

```
<StackPanel Background="#ECE9D8">
  <TextBlock Margin="3">Look for:</TextBlock>
  <ComboBox Margin="3"/>
  <TextBlock Margin="3">Filtered by:</TextBlock>
  <ComboBox Margin="3"/>
  <Button Margin="3,5">Search</Button>
  <CheckBox Margin="3">Search in titles only</Chec
  <CheckBox Margin="3">Match related words</CheckB
  <CheckBox Margin="3">Search in previous results<
  <CheckBox Margin="3">Highlight search hits (in topics)</CheckBox>
</StackPanel>
```

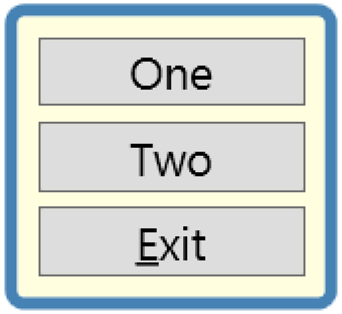


- StackPanel is a very simple panel that arranges its children in a row or a column.
- **It is most useful for arranging small subsections of the UI**.
- The default behavior of a vertical StackPanel is to make all of the controls the same width as the panel.
  - This is not always what we want.
  - When an element has been given a fixed amount of space that is greater than required by its content, the way in which the extra space gets used is determined by the HorizontalAlignment and VerticalAlignment properties.
  - We can prevent the button from being stretched across the panel's whole width by setting its HorizontalAlignment property to Left:
    <Button Margin="3,5" **HorizontalAlignment="Left"**>Search</Button>

Border

- The Border isn’t a layout panels, but a control.
- But the Border is a handy element that is often used together with a panel.

```
<Border Margin="5" Padding="5"
        Background="LightYellow"
        BorderBrush="SteelBlue"
        BorderThickness="5"
        CornerRadius="5"
        VerticalAlignment="Top">
    <StackPanel>
        <Button Margin="3">One</Button>
        <Button Margin="3" Content="Two"/>
        <Button Margin="3" Content="_Exit"
                Name="btnExit"
                Click="btnExit_Click"/>
    </StackPanel>
</Border>
```

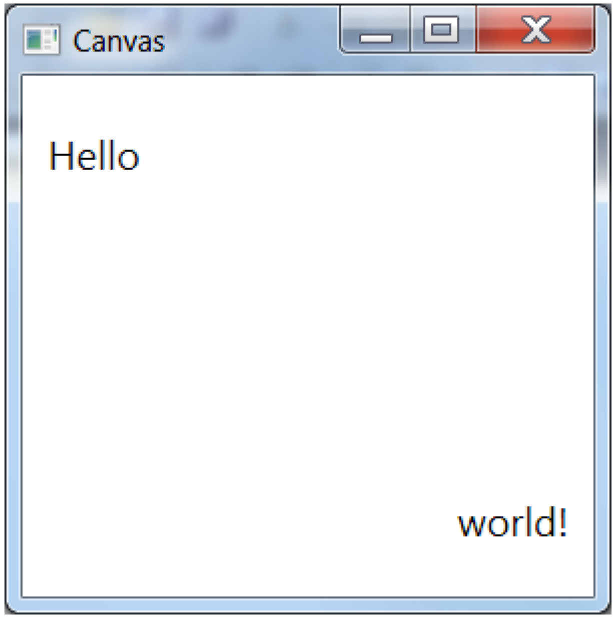


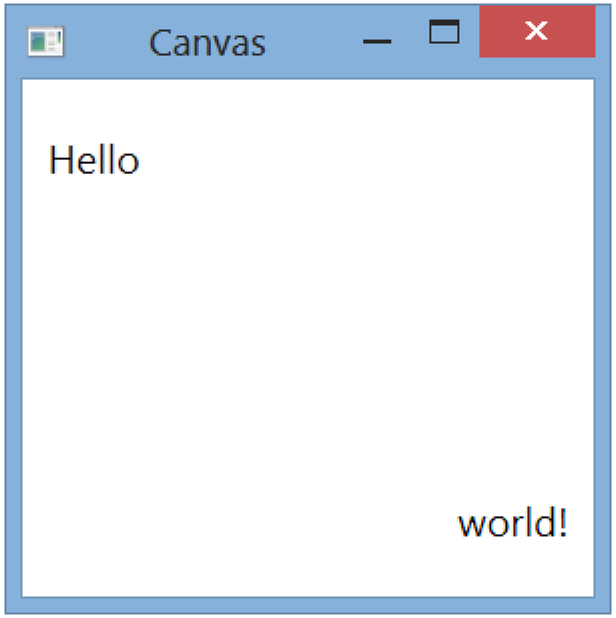
Canvas

- Canvas is the simplest of the panels.
- It allows the location of child elements to be specified precisely relative to the edges of the canvas.
  - **The Canvas doesn't really do any layout at all!**
  - it simply puts things where you tell it to.
- Also, Canvas will not size elements to fill the available space
  - all its children are sized to content.
- Use a Canvas to take complete control of the precise positioning of every element
  - when you want to build an image out of graphical elements, the positioning of the elements is dictated by the picture you are creating, not by any set of automated layout rules!
  - Note:
    You are much more likely to have code that adds items to a canvas.

Positioning on a Canvas

```
<Canvas Background="LemonChiffon">
  <TextBlock Canvas.Left="10" Canvas.Top="20">Hello</TextBlock>
  <TextBlock Canvas.Right="10" Canvas.Bottom="20">world!</TextBlock>
</Canvas>
```





Canvas – Warning!

- If you are accustomed to working with fixed layout systems such as those offered by Windows Forms, the Canvas will seem familiar and natural.
- However, it is strongly recommended that you avoid it!
  - unless you really need this absolute control.
- The automatic layout provided by the other panels will make your life much easier because:
  - they can adapt to changes in text and font.
  - They also make it far simpler to produce resizable user interfaces.
  - Localization tends to be much easier with resizable user interfaces, because different languages tend to produce strings with substantially different lengths.
- Don't opt for the Canvas simply because it seems familiar!
- **The main use for Canvas is to arrange drawings**.

Attached Properties

- An attached property is one where the property is defined by a different type than the element to which the property is applied.
- WPF uses this to allow properties specific to the layout panel to be set for any UIElement.
  - *The advantage by using attached properties is that they don't need to be implemented and consume memory on elements that don't use them!*
- Syntax
  - **XAML:**

```
<Canvas Name=canvas1>
   <Button Canvas.Left="119" Canvas.Top="24">
    Button</Button>
```

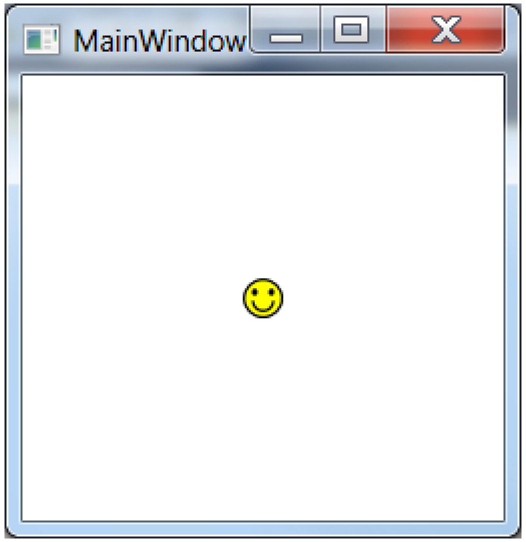
  - **Code:**

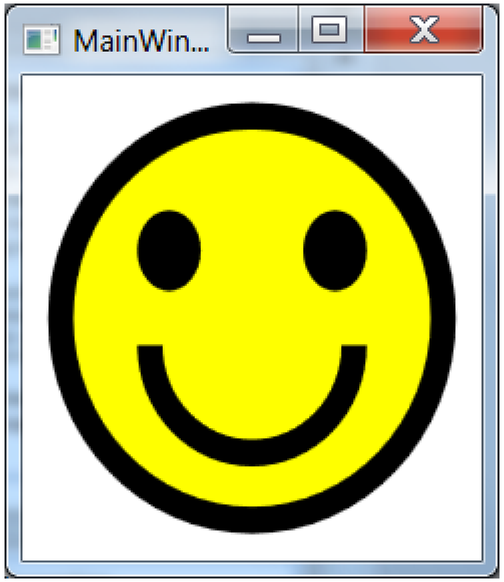
```
button1 = new Button { Content = "Button", Width = 70,
          Height = 23 };
Canvas.SetLeft(button1, 119);
Canvas.SetTop(button1, 24);
canvas1.Children.Add(button1);
```

Obs:
type name of layout panel
– NOT instance!

Viewbox

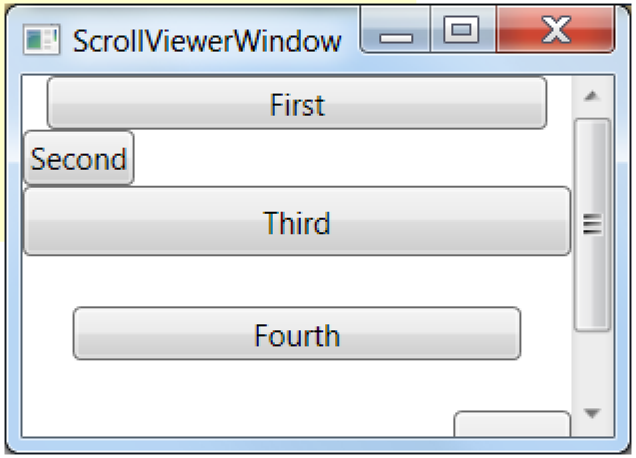
- Automatically scales its content to fill the space available.
- Viewbox is not a panel
  - it derives from Decorator.
  - This means that unlike most panels, it can have only one child.
- Its capability to adjust the size of its content in order to adapt to its surroundings makes it a useful layout tool.
- The Viewbox can scale any child element—it's not just for Canvas.
  - But you would rarely use it to size anything other than a drawing.
  - If you were to use a Viewbox to resize some nongraphical part of your UI, it would resize any text in there as well, making it look inconsistent with the rest of your UI.





ScrollViewer

```
<Window x:Class="_04ScrollViewer.Window1”
        …
        Title="ScrollViewerWindow" Height="180" Width="250">
   <ScrollViewer VerticalScrollBarVisibility="Auto"
                 HorizontalScrollBarVisibility="Auto">
      <StackPanel Orientation="Vertical">
         <Button Width="200">First</Button>
         <Button HorizontalAlignment="Left">Second</Button>
         <Button Padding="10 4">Third</Button>
         …
      </StackPanel>
   </ScrollViewer>
</Window>
```

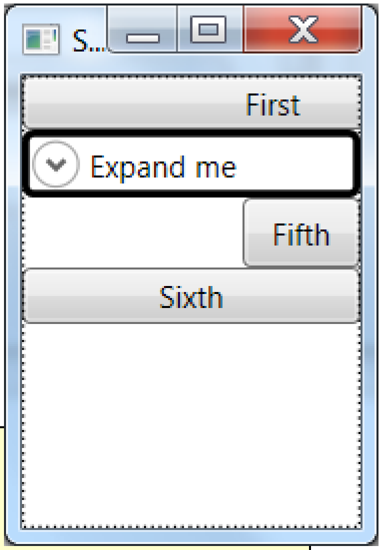


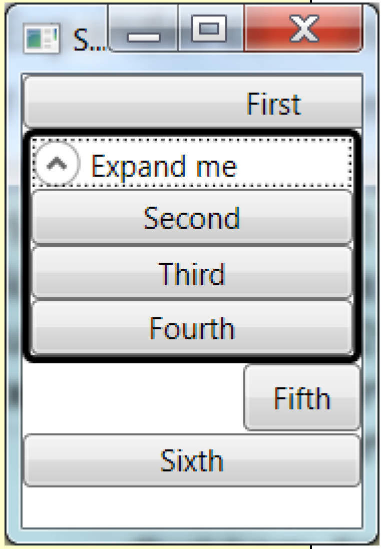
- The ScrollViewer control allows oversized content to be displayed by putting it into a scrollable area.
- A ScrollViewer element has a single child.

The Expander Control

- The Expander control allow you to show or hide different sections.
- The Expander can only contain one thing: its contents.
  - If we want to add multiple items, we have to put some sort of panel (a StackPanel) inside the Expander.

```
<ScrollViewer VerticalScrollBarVisibility="Auto">
   <StackPanel Orientation="Vertical">
      <Button Width="200" >First</Button>
      <Expander Header="Expand me" BorderThickness="3”
                BorderBrush="Black">
         <StackPanel>
            <Button>Second</Button>
            <Button>Third</Button>
            <Button>Fourth</Button>
         </StackPanel>
      </Expander>
      <Button Padding="10 4”
              HorizontalAlignment="Right">Fifth</Button>
      <Button HorizontalAlignment="Stretch">Sixth</Button>
   </StackPanel>
</ScrollViewer>
```

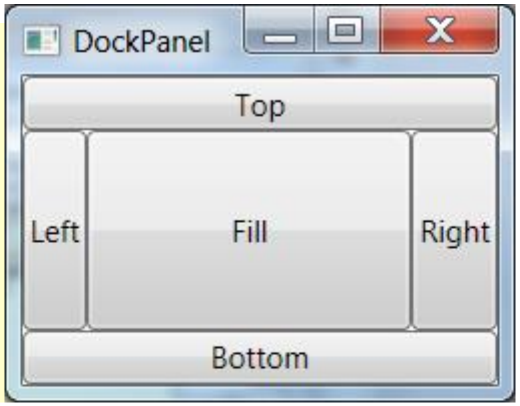




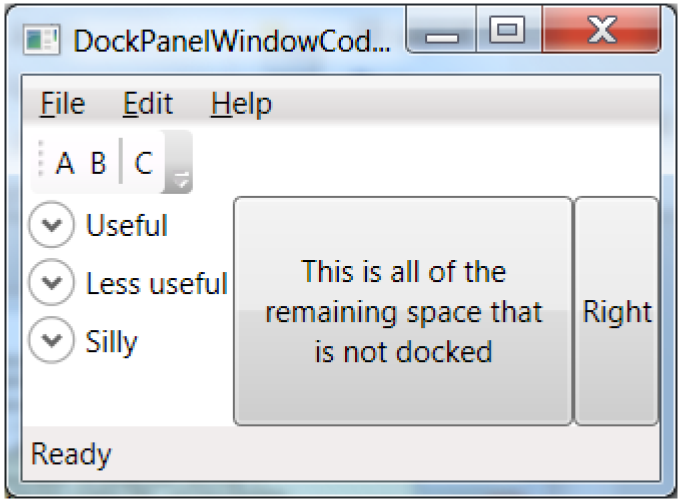
AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

DockPanel

```
<DockPanel>
  <Button DockPanel.Dock="Top">Top</Button>
  <Button DockPanel.Dock="Bottom">Bottom</Button>
  <Button DockPanel.Dock="Left">Left</Button>
  <Button DockPanel.Dock="Right">Right</Button>
  <Button>Fill</Button>
</DockPanel>
```



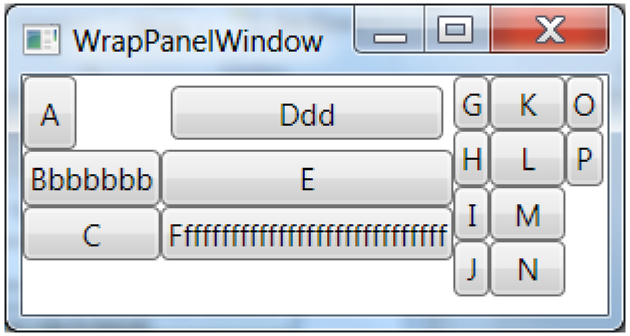
- Is useful for describing the overall layout of a simple user interface.
- Use it to position a menu and a toolbar at the top of the window.
  - Or to position a statusbar at the bottom.
- Elements never overlap in a DockPanel!



This is all of the remaining space that is not docked

WrapPanel

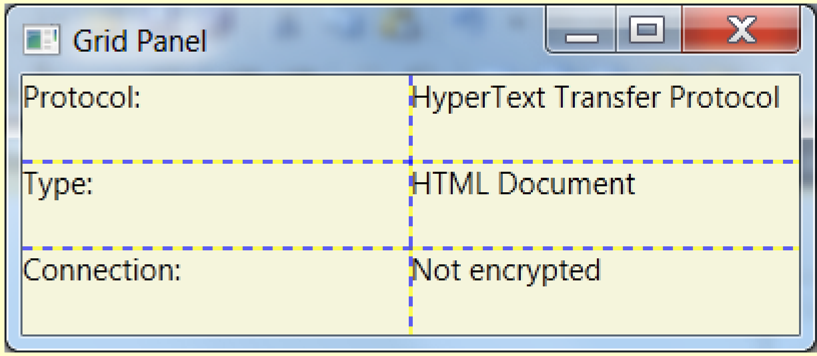
```
<WrapPanel Orientation="Vertical">
   <Button HorizontalAlignment="Left"
           Padding="5">A</Button>
   <Button >Bbbbbbb</Button>
   <Button >C</Button>
   <Button Margin="4">Ddd</Button>
   <Button >E</Button>
   …
</WrapPanel>
```



- Works just like a StackPanel until it runs out of space.
- Puts the children in a row from left to right until it runs out of space, at which point it starts on the next line.
- It is most useful for arranging small subsections of the UI.
- Has an Orientation property.
  - Setting this to Vertical will arrange the children in a sequence of vertical stacks, a layout style very similar to Windows Explorer's "List" view.

Grid

```
<Grid Background="Beige“ ShowGridLines="True">
  <Grid.ColumnDefinitions>
    <ColumnDefinition />
    <ColumnDefinition />
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition />
    <RowDefinition />
    <RowDefinition />
  </Grid.RowDefinitions>
  <TextBlock Grid.Column="0" Grid.Row="0">Protocol:</TextBlock>
  <TextBlock Grid.Column="1" Grid.Row="0">HyperText Transfer Protocol</TextBlock>
  <TextBlock Grid.Column="0" Grid.Row="1">Type:</TextBlock>
  <TextBlock Grid.Column="1" Grid.Row="1">HTML Document</TextBlock>
  <TextBlock Grid.Column="0" Grid.Row="2">Connection:</TextBlock>
  <TextBlock Grid.Column="1" Grid.Row="2">Not encrypted</TextBlock>
</Grid>
```



Protocol:	HyperText Transfer Protocol
Type:	HTML Document
Connection:	Not encrypted

- Aligns all elements into a grid that covers the whole area of the panel.
- The Grid needs to know how many columns and rows we require, and we indicate this by specifying a series of ColumnDefinition and RowDefinition elements at the start.
- Each element in the grid has its column and row specified explicitly using ***attached properties***.

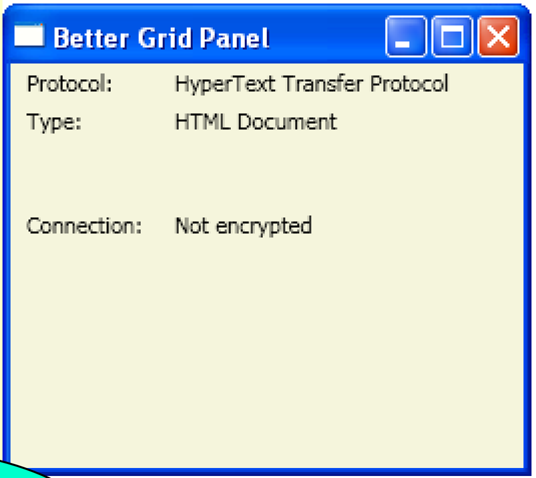
Fixed Sizes Contra Automatic Layout

- You configure the column widths and row heights in a Grid using the ColumnDefinition and RowDefinition elements.
- There are three sizing options:
  - fixed
  - automatic
  - proportional
- Fixed sizing is the simplest to understand,
  - but often requires the most effort to use!
    - If you change the text or the font, you will need to modify the sizes to match.
    - You will need to be flexible on layout if you want your application to fit in with the system look and feel, because the default font is not the same on all versions of Windows.
    - Localization of strings will also require the sizes to be changed.
- In general, you should try to avoid fixed sizes in **WPF**
  - The more you let the layout system do for you, the easier it is to adapt to:
    - localization,
    - different screen sizes, and
    - display orientations.

Proportional Grid Sizing

```
<Grid Background="Beige">

  <Grid.ColumnDefinitions>
     <ColumnDefinition Width="Auto" />
     <ColumnDefinition />
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
     <RowDefinition Height="Auto" />
     <RowDefinition Height="1*"/>
     <RowDefinition Height="2.5*"/>
  </Grid.RowDefinitions>
```



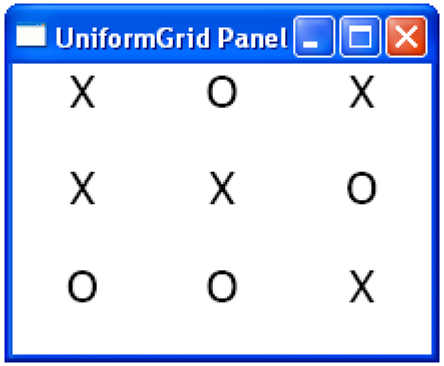
Protocol:	HyperText Transfer Protocol
Type:	HTML Document
Connection:	Not encrypted

Proportional Grid sizing

```
// Setting row height in C# code
Grid g = new Grid( );
RowDefinition r = new RowDefinition( );
r.Height = new GridLength(0, GridUnitType.Auto);
g.RowDefinitions.Add(r);
r = new RowDefinition(  );
r.Height = new GridLength(2, GridUnitType.Star);
g.RowDefinitions.Add(r);
```

UniformGrid

```
<UniformGrid TextBlock.TextAlignment="Center">
  <TextBlock Text="X" />
  <TextBlock Text="O"/>
  <TextBlock Text="X"/>
  <TextBlock Text="X"/>
  <TextBlock Text="X"/>
  <TextBlock Text="O"/>
  <TextBlock Text="O"/>
  <TextBlock Text="O"/>
  <TextBlock Text="X"/>
</UniformGrid>
```



- Is a simplified version of the Grid panel.
- All its cells are the same size,
  - so you don't need to provide collections of row and column descriptions
  - just set the Rows and Columns properties to indicate the size
    - you don't even need to set these
      - by default, it creates rows and columns automatically
    - It always keeps the number of rows and columns equal to each other, adding as many as are required to make space for the children.

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING