C# 2 – Les 3: Grafische User Interfaces (GUIs)

In deze les gaan we onze eerste grafische Windows applicatie maken! Er zijn 3 verschillende manieren om een grafische applicatie te maken:

1. **Windows Forms**: dit is een wat verouderde manier om Windows desktop applicaties te maken. Grafisch is dit wat beperkter met bijvoorbeeld beperkte ondersteuning voor hoge dpi schermen. Dit framework draait enkel op de CPU waardoor de CPU zwaarder belast wordt.

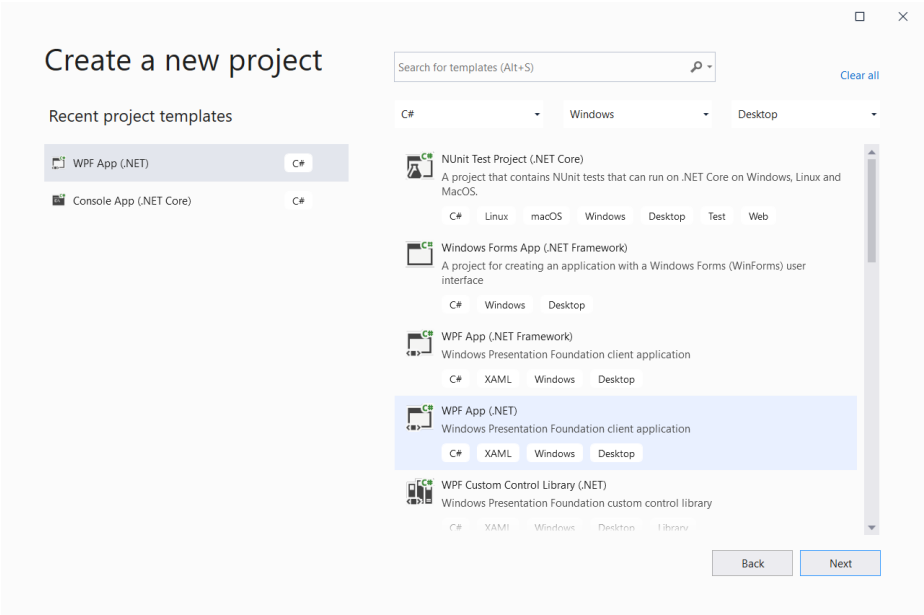
2. **Windows Presentation Foundation (WPF)**: een modern framework om Windows desktop applicaties te maken. Hoge dpi schermen worden ondersteund en de grafische kaart wordt gebruikt om de CPU te ontlasten.

3. **Universal Windows Platform (UWP)**: een modern framework om applicaties te maken die bedoeld zijn om via de Windows store te worden verdeeld en applicaties de mogelijkheid bood om op zowel de Windows Phone (RIP) als op desktop te kunnen draaien. Door de sterke koppeling met de Windows store is dit framework wat minder interessant. Mogelijk wordt het in de toekomst gemakkelijker om hiermee vrij aan de slag te gaan.

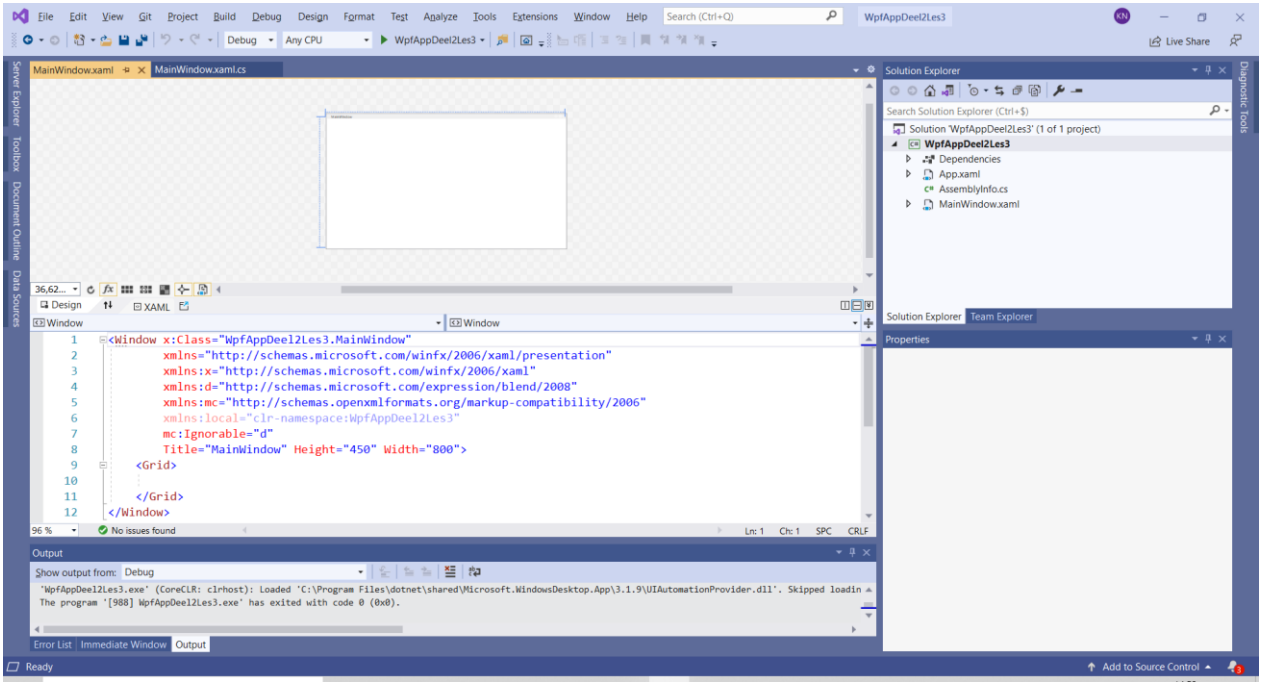
We gaan in deze module focussen op WPF-applicaties.

Een eerste WPF-applicatie maken

Om een WPF App te maken kies je de optie “WPF App (.NET)” bij het maken van een nieuw project.



Na het maken van een project wordt je begroet door een grafische weergave van een Windows venster en bijhorende XAML-code onderaan. De code die getoond wordt zit in het bestand MainWindow.xaml.



Je kan dit programma al uitvoeren en dan zie je een leeg venster van 450px hoog op 800px breed.

In de *Solution Explorer* zie je dat er een aantal bestanden zijn aangemaakt. Bij het bestand MainWindow.xaml zit een bijhorend .cs bestand (onder het kleine driehoekje naast MainWindow.xaml). In dit bestand zit de C#-code die bij de layout hoort. Dit bestand wordt ook wel eens de *"code behind"* genoemd (de "achterliggende code"). Daar gaan we straks nog een paar dingen aan toevoegen.

XAML

XAML is de code die bepaalt welke elementen er in het programma worden getoond. XAML is een XML-achtige taal die wat lijkt op HTML. Als je al wat HTML kan is dat mooi meegenomen. Bij de start-code staat deze code in het XAML-bestand dat gekoppeld is aan de applicatie:

```
<Window x:Class="WpfAppDeel2Les3.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:local="clr-namespace:WpfAppDeel2Les3"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
```

```
    <Grid>

    </Grid>
</Window>
```

XAML bestaat, net als XML, uit zogenaamde tags. In de code hierboven zie je een Grid-tag (bestaande uit een open-tag: <Grid> en sluit-tag: </Grid>). Alles wat tussen de Grid-tag komt te staan gaat te maken hebben met het Grid-element.

Grid

Een Grid is een manier om elementen netjes te gaan plaatsen in een raster van rijen en kolommen. De Grid-tag is eigenlijk optioneel, maar wordt door Visual Studio al geplaatst omdat dit één van de handigste manieren is om elementen te gaan rangschikken.

De Grid-tag staat in de Window-tag. De Window-tag is niet optioneel, deze moet altijd aanwezig zijn om het programma uit te kunnen voeren.

Om een grid of raster van 3 rijen en 3 kolommen te maken schrijven we dit:

```
<Grid ShowGridLines="True">
    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>
</Grid>
```

Om de grid zichtbaar te maken zetten we tijdelijk de ShowGridLines optie op True. Als je nu het programma start zie je hoe de grid het venster verdeelt in 9 delen van gelijke grootte. Versleep ook het venster eens van grootte, de grid past zich automatisch aan.

Een TextBlock op de grid zetten

Een tekst zetten in 1 van de delen van de grid kan met een TextBlock, hieronder in het geel getoond:

```
<Grid ShowGridLines="True">
    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>
    <TextBlock>Eerste tekst</TextBlock>
</Grid>
```

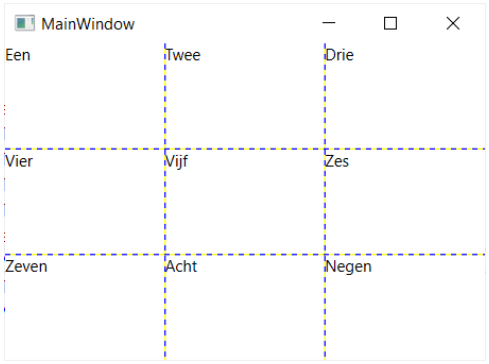
De TextBlock komt automatisch in de eerste “cel” van de grid. Je kan de TextBlock verplaatsen met Grid.Column en Grid.Row. Probeer dit eens (waar komt de tekst in de grid te staan?):

```
<TextBlock Grid.Row="2" Grid.Column="1">Eerste tekst</TextBlock>
```

Je kan natuurlijk ook meerdere TextBlocken gaan zetten. (De shortcut om een regel te dupliceren is ctrl-d.)

Oefening 1: Grid-layouts

1. Maak 9 TextBlocken en zet ze zo:



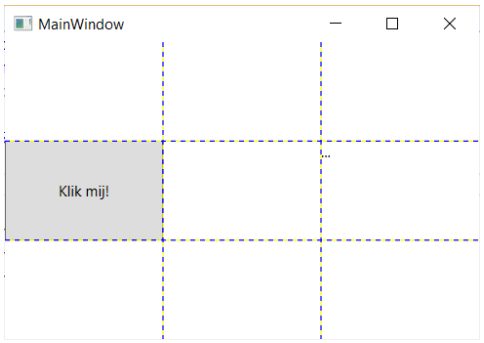
2. Plaats de code van de grid van vorige oefening even in commentaar (ctrl-k, ctrl-c) en maak een nieuwe grid met deze layout:

Button toevoegen

Het Button-element is een knop-achtig element waaraan normaalgezien code aanhangt die wordt geactiveerd bij het klikken op de knop. De code gaan we in een methode zetten, de methode zal automatisch worden uitgevoerd als we op de knop klikken.

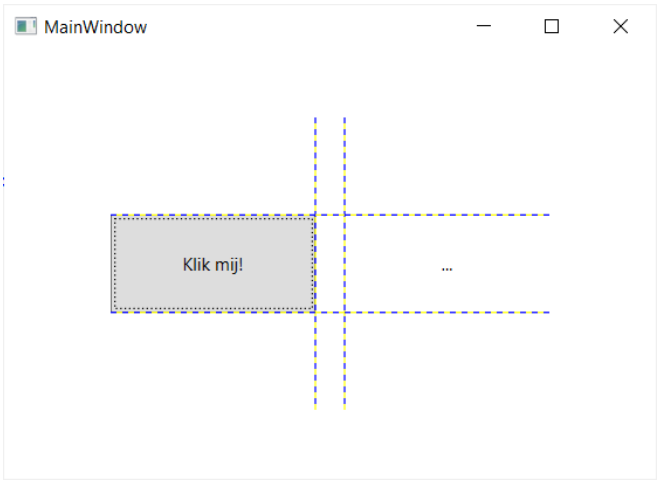
We starten met eerst gewoon een Button in een Grid te zetten. We gebruiken een extra rij en kolom om wat plaats te voorzien tussen de elementen.

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition></ColumnDefinition>
        <ColumnDefinition></ColumnDefinition>
        <ColumnDefinition></ColumnDefinition>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition></RowDefinition>
        <RowDefinition></RowDefinition>
        <RowDefinition></RowDefinition>
    </Grid.RowDefinitions>
    <Button Grid.Column="0" Grid.Row="1">Klik mij!</Button>
    <TextBlock Grid.Column="2" Grid.Row="1">...</TextBlock>
</Grid>
```



De grid plaatst zich nu over heel ons scherm waardoor de knop tegen de rand plakt van het venster. Ook is de ruimte tussen de knop en het TextBlock te groot. Met enkele instellingen kunnen we de layout wat netter zetten:

```
<Grid ShowGridLines="True" MaxWidth="300" MaxHeight="200" Margin="20, 20, 20,
20">
    <Grid.ColumnDefinitions>
        <ColumnDefinition></ColumnDefinition>
        <ColumnDefinition Width="20"></ColumnDefinition>
        <ColumnDefinition></ColumnDefinition>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition></RowDefinition>
        <RowDefinition></RowDefinition>
        <RowDefinition></RowDefinition>
    </Grid.RowDefinitions>
    <Button Grid.Column="0" Grid.Row="1">Klik mij!</Button>
    <TextBlock Grid.Column="2" Grid.Row="1" HorizontalAlignment="Center"
VerticalAlignment="Center">...</TextBlock>
</Grid>
```



De instellingen die gebruikt zijn (in volgorde zoals ze voorkomen in de code hierboven):

- **MaxWidth**: het grid-element wordt nu niet breder dan 300 pixels (als het venster kleiner wordt wordt de breedte nog wel aangepast).
- **MaxHeight**: het grid-element wordt nu niet hoger dan 200 pixels.
- **Margin**: langs elke kant van het element wordt steeds 20 pixels witruimte voorzien zodat de knop niet tegen de rand van het venster zal komen (ook niet bij ene breedte kleiner dan 300 pixels).
- **Width**: de 2de kolom is exact 20 pixels breed
- **HorizontalAlignment** en **VerticalAlignment**: de TextBlock wordt gecentreerd in de grid cell.

Code aan de button hangen

Momenteel doet onze button nog niets ☹ Maar, we kunnen een stukje code (een methode) aan de button hangen. De methode zal dan uitgevoerd worden als we op de button klikken. Een methode koppelen die gaat reageren op een muisklik doen we door het woordje **Click** toe te voegen op de Button:

```
<Button Grid.Column="0" Grid.Row="1" Click="Button_Click">Klik mij!</Button>
```

In het *code behind* bestand MainWindow.cs is er nu normaalgezien een lege methode *Button_Click* bijgemaakt (als dat niet gebeurd is tip je ze even zelf):

```
namespace WpfAppDeel2Les3
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    2 references
    public partial class MainWindow : Window
    {
        0 references
        public MainWindow()
        {
            InitializeComponent();
        }

        1 reference
        private void Button_Click(object sender, RoutedEventArgs e)
        {

        }
    }
}
```

Na het woord **Click** in het xaml-bestand staat de naam van de methode die uitgevoerd zal worden bij een muisklik. De naam van de methode moet overeenkomen met wat er op het Button-element vermeld staat bij **Click** om de methode te koppelen.

In MainWindow.cs kan je code gaan toevoegen. (De constructormethode *MainWindow* wordt automatisch uitgevoerd bij het tonen van het scherm.)

In de Button_Click methode kunnen we om te beginnen een bericht sturen naar het output-venster van Visual Studio. Dat kan soms handig zijn om de waarden van variabelen te tonen als

je programma iets onverwachts doet (bug). Je kan het output-venster dus gebruiken om te debuggen. Iets printen in het output-venster doe je zo:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    Debug.WriteLine("Knop testje...");
}
```

(Om te kunnen schrijven naar het output-venster heb je nog een extra using-regel nodig.)

Als je het programma uitvoert en op de knop drukt verschijnt er iets in het output-venster in Visual Studio. Vind je het output-venster? Als het venster niet zichtbaar is kijk dan even onder het menu *View*.

De tekst van het TextBlock veranderen

In plaats van iets naar het output-venster te sturen gaan we de tekst van het TextBlock veranderen. Je moet daarvoor het TextBlock toegankelijk maken in de code, dat doe je met **x:Name**:

```
<TextBlock Grid.Column="2" Grid.Row="1"
HorizontalAlignment="Center" VerticalAlignment="Center"
x:Name="ons_textblokje">...</TextBlock>
```

Door x:Name toe te voegen kan je het TextBlock (of een ander element) bereiken via de code. Gebruik hiervoor de naam die je zonet hebt toegevoegd:

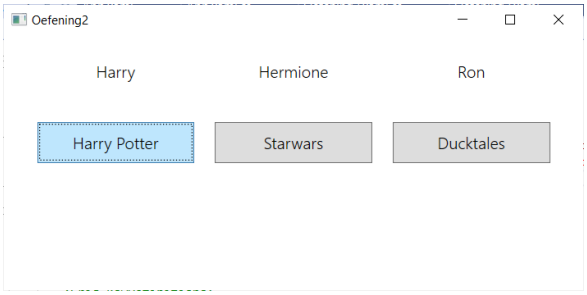
```
private void Button_Click(object sender, RoutedEventArgs e)
{
    ons_textblokje.Text = "BOEM!";
}
```

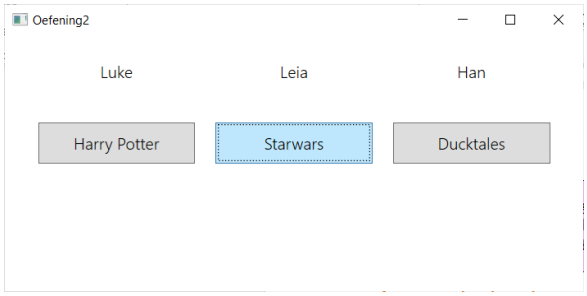
Als je op de knop drukt verandert de tekst in het TextBlock:

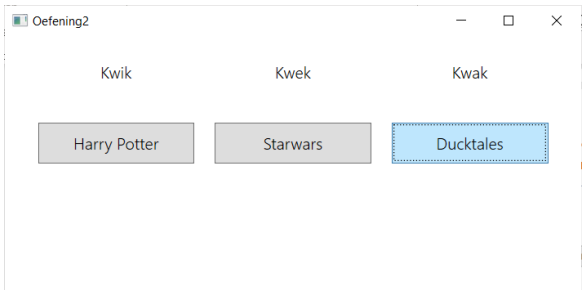
Het klikken op een knop door de gebruiker van het programma wordt een **event** (een "gebeurtenis") genoemd. De methode die aan de knop hangt en die zal uitgevoerd worden bij een klik wordt een **event-handler** genoemd.

Oefening 2: Bekende trio's

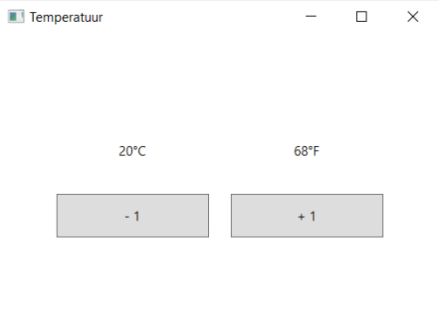
Maak een grid-layout met 3 textblocken en 3 knoppen. Hang aan de knoppen 3 methodes die de teksten van de textblocken zo zal veranderen:







Oefening 3: Celsius/Fahrenheit



Maak een programma met 2 TextBlocken en 2 Buttons. Met de knoppen kan je de temperatuur in Celsius een graad verhogen of verlagen. Zorg ervoor dat de temperatuur in Fahrenheit mee verandert.

TIP: Je kan in de MainWindow class een property toevoegen om de temperatuur bij te houden.

Oefening 4: BMI

Maak een nieuw project met 3 TextBlocken en 4 Buttons. Maak een programma waarmee dat de body mass index (BMI) berekent aan de hand van deze formule:

BMI= gewicht (kg) / (lengte (m) x lengte (m))

Met de knoppen kan je de lengte en het gewicht aanpassen. Met de Grid.ColumnSpan kan je een cel meerdere kolommen laten overspannen. In onderstaand voorbeeld is dat gebruikt om het bovenste textblock (BMI: …) over 3 kolommen te laten lopen. Probeer ook de font groter te zetten zoals in het voorbeeld.