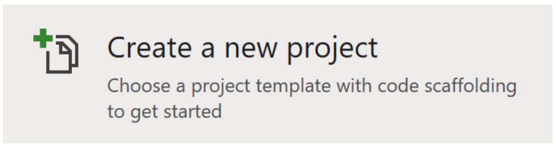
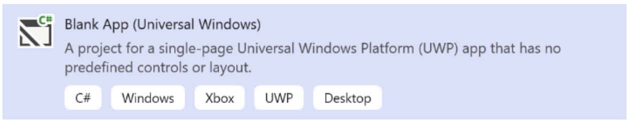
Universal Windows Platform – Lucky Dice

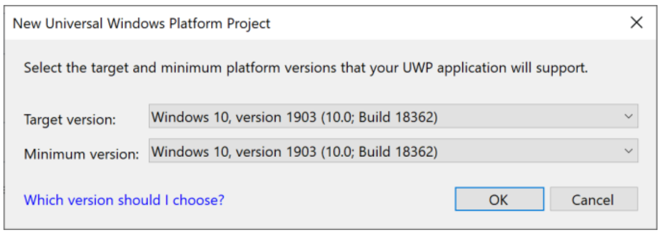
Step 1


Create a new project
Choose a project template with code scaffolding to get started

Follow **Setup and Start** on how to Install and/or Get Started with **Visual Studio 2019** if not already or in **Windows 10** choose **Start**, find and select **Visual Studio 2019** then from the **Get started** screen select **Create a new project**


Blank App (Universal Windows)
A project for a single-page Universal Windows Platform (UWP) app that has no predefined controls or layout.
C#
Windows
Xbox
UWP
Desktop

Then choose **Blank App (Universal Windows)** and select **Next** and then in **Configure your new project** enter the **Project name** as **LuckyDice** and select **Create**



New Universal Windows Platform Project

Select the target and minimum platform versions that your UWP application will support.

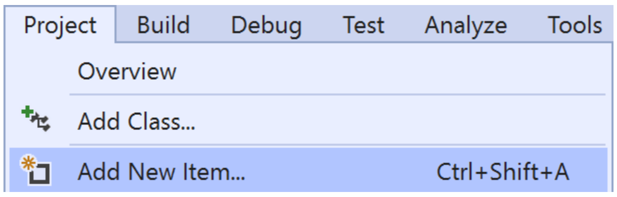
Target version: Windows 10, version 1903 (10.0; Build 18362)

Minimum version: Windows 10, version 1903 (10.0; Build 18362)

Finally, in **New Universal Windows Platform Project** pick the **Target version** and **Minimum version** to be at least **Windows 10, version 1903 (10.0; Build 18362)** and then select **OK**

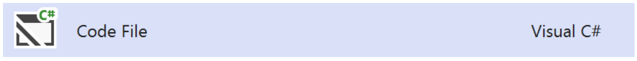
Target Version will control the most recent features of Windows 10 your application can use. To make sure you always have the most recent version, check for any Notifications or Updates in Visual Studio 2019

Step 2



Choose **Project** then **Add New Item...** from the **Menu** in **Visual Studio 2019**

Step 3



Then choose **Code File** from **Add New Item** in **Visual Studio 2019**, enter the **Name** as **Library.cs** and select **Add**

Universal Windows Platform – Lucky Dice

Step 4

In the **Code** View of **Library.cs** will be displayed and in this the following should be entered:

```
using System;
using System.Linq;
using Windows.UI;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Shapes;

public class Library
{
    private const int size = 3;

    private static readonly byte[][] layout =
    {
                   // a, b, c, d, e, f, g, h, i
        new byte[] { 0, 0, 0, 0, 0, 0, 0, 0, 0 }, // 0
        new byte[] { 0, 0, 0, 0, 1, 0, 0, 0, 0 }, // 1
        new byte[] { 1, 0, 0, 0, 0, 0, 0, 0, 1 }, // 2
        new byte[] { 1, 0, 0, 0, 1, 0, 0, 0, 1 }, // 3
        new byte[] { 1, 0, 1, 0, 0, 0, 1, 0, 1 }, // 4
        new byte[] { 1, 0, 1, 0, 1, 0, 1, 0, 1 }, // 5
        new byte[] { 1, 0, 1, 1, 0, 1, 1, 0, 1 }, // 6
    };
    private readonly Color _accent =
        (Color)Application.Current.Resources["SystemAccentColor"];

    private Random _random = new Random((int)DateTime.UtcNow.Ticks);


}
```

There are `using` statements to include necessary functionality. `layout` is a `byte[][]` is a two-dimensional array of values that will represent which row and column of pips will be displayed on the dice or die. `_accent` is a `Color` that will be used to set the look of the dice and `Random` is used to create the numbers for the dice

Then below the **`private Random _random = new Random((int)DateTime.UtcNow.Ticks);`** line the following **method** should be entered:

```
private void Add(ref Grid grid, int row, int column)
{
    Ellipse element = new Ellipse()
    {
        Fill = new SolidColorBrush(_accent),
        Margin = new Thickness(5),
        Opacity = 0
    };
    element.SetValue(Grid.ColumnProperty, column);
    element.SetValue(Grid.RowProperty, row);
    grid.Children.Add(element);
}
```

`Add(...)` is used to create the `Ellipse` to represent the pips of the dice

Next below the `private void Add(...) { }` **method** the following **method** should be entered:

```
private void Set(ref Grid grid, int row, int column, byte opacity)
{
    Grid element = (Grid)((Viewbox)grid.Children
        .FirstOrDefault()).Child;
    Ellipse ellipse = element.Children.Cast<Ellipse>()
        .FirstOrDefault(f =>
    Grid.GetRow(f) == row && Grid.GetColumn(f) == column);
    if (ellipse != null) ellipse.Opacity = opacity;
}
```

`Set(...)` is used to get the first with `FirstOrDefault()` existing `Grid` item within a `Viewbox` and then from this get the first `Ellipse` item and set the `Opacity` to the passed in value

After the **`private void Set(...)`** **method** the following **method** should be entered:

```
private void Update(ref Grid grid, int value)
{
    int count = 0;
    for (int row = 0; row < size; row++)
    {
        for (int column = 0; column < size; column++)
        {
            Set(ref grid, row, column, layout[value][count]);
            count++;
        }
    }
}
```

`Update(...)` will for a given `Grid` loop through all the rows and columns of this and use the `Set(...)` method to update pips of the dice using the `value` passed in

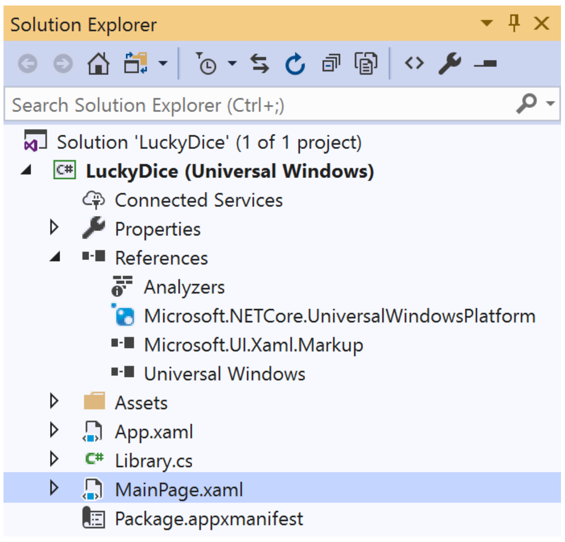
Finally, after the **`private void Update(...)` method** the following `public` **methods** should be entered:

```
public void New(ref Grid grid)
{
    grid.Children.Clear();
    Grid element = new Grid()
    {
        Width = 100,
        Height = 100,
        Padding = new Thickness(5)
    };
    // Setup Grid
    for (int index = 0; index < size; index++)
    {
        element.RowDefinitions.Add(new RowDefinition());
        element.ColumnDefinitions.Add(new ColumnDefinition());
    }
    for (int row = 0; row < size; row++)
    {
        for (int column = 0; column < size; column++)
        {
            Add(ref element, row, column);
        }
    }
    Viewbox viewbox = new Viewbox()
    {
        Child = element
    };
    grid.Children.Add(viewbox);
    Update(ref grid, 0);
}

public void Get(ref Grid grid)
{
    if(!grid.Children.Any()) New(ref grid);
    Update(ref grid, _random.Next(1, 7));
}
```

`New(...)` will setup the layout of the `Grid` passed in with the `RowDefinition` and `ColumnDefinition` and use the `Add(...)` method to create the layout of the dice and then place this within a `ViewBox` and add this to the `Grid`. `Get(...)` will call the `New` method if nothing has been added to the `Grid` then calls the `Update(...)` method to set the dice value using the randomised number from `Random`

Universal Windows Platform – Lucky Dice

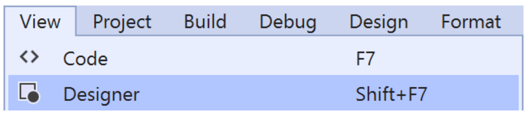
Step 5



Solution 'LuckyDice' (1 of 1 project)
LuckyDice (Universal Windows)
Connected Services
Properties
References
Analyzers
Microsoft.NETCore.UniversalWindowsPlatform
Microsoft.UI.Xaml.Markup
Universal Windows
Assets
App.xaml
Library.cs
MainPage.xaml
Package.appxmanifest

In the **Solution Explorer** of **Visual Studio 2019** select **MainPage.xaml**

Step 6



Choose **View** then **Designer** from the **Menu** in **Visual Studio 2019**

Universal Windows Platform – Lucky Dice

Step 7

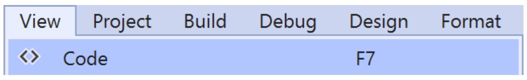
In the **Design** View and **XAML** View of **Visual Studio 2019** will be displayed, and in this between the **`Grid`** and **`/Grid`** elements enter the following **XAML**:

```
<Viewbox>
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="*"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>
        <Grid Margin="50" Grid.Column="1" Grid.Row="1"
        Name="DiceOne" CornerRadius="10" Background="WhiteSmoke"
        Height="100" Width="100" Tapped="DiceOne_Tapped"/>
        <Grid Margin="50" Grid.Column="3" Grid.Row="1"
        Name="DiceTwo" CornerRadius="10" Background="WhiteSmoke"
        Height="100" Width="100" Tapped="DiceTwo_Tapped"/>
    </Grid>
</Viewbox>
<CommandBar VerticalAlignment="Bottom">
    <AppBarButton Icon="Page2" Label="New" Click="New_Click"/>
</CommandBar>
```

The first block of XAML the main user interface features a Grid with two Grid Controls within to represent the dice. The second block of XAML is the CommandBar which contains New to reset the game

Universal Windows Platform – Lucky Dice

Step 8



Choose **View** then **Code** from the **Menu** in **Visual Studio 2019**

Step 9

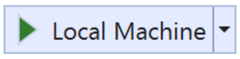
Once in the **Code** View, below the end of **`public MainPage() { ... }`** the following Code should be entered:

```
Library library = new Library();

private void New_Click(object sender, RoutedEventArgs e)
{
    library.New(ref DiceOne);
    library.New(ref DiceTwo);
}

private void DiceOne_Tapped(object sender, RoutedEventArgs e)
{
    library.Get(ref DiceOne);
}

private void DiceTwo_Tapped(object sender, RoutedEventArgs e)
{
    library.Get(ref DiceTwo);
}
```

Below the MainPage(...) method an instance of the `Library` Class is created. In the `New_Click(...)` Event handler will setup the two Grid Controls, `DiceOne_Tapped(...)` and `DiceTwo_Tapped(...)` will call the `Get` method in the `Library` Class

Universal Windows Platform – Lucky Dice

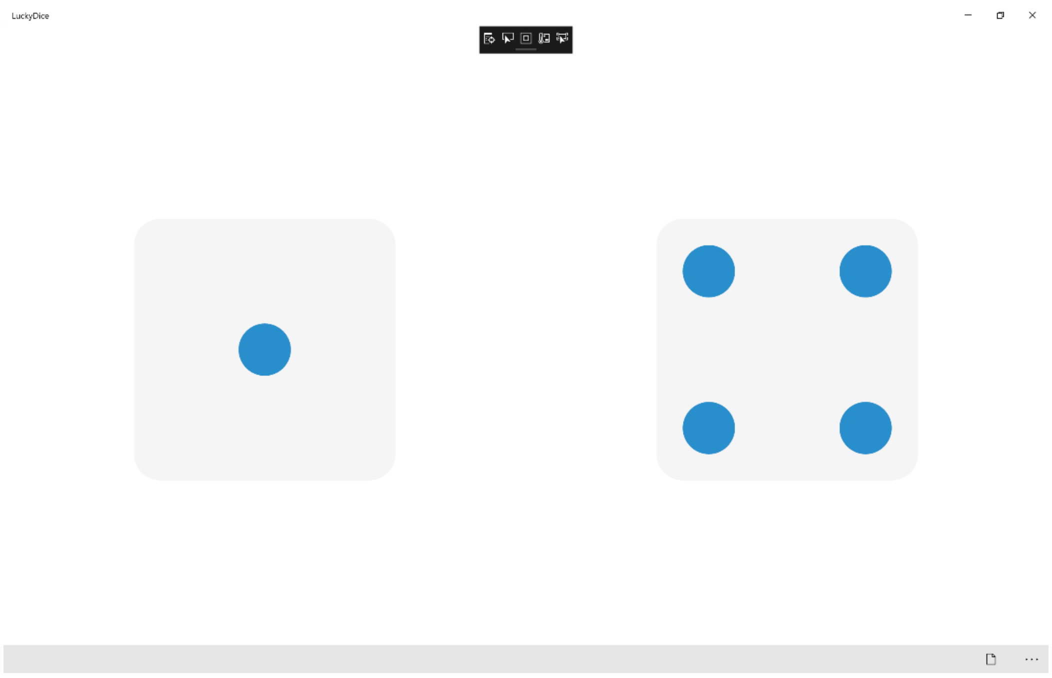
Step 10



That completes the **Universal Windows Platform** Application, in **Visual Studio 2019** select **Local Machine** to run the Application

Step 11

Once the Application is running you can then click on either of the **Grid** controls to randomly show the value of the dice or use **New** to reset them



Step 12

To Exit the Application, select the **Close** button in the top right of the Application