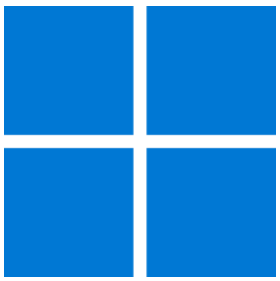
tutorial

Windows App SDK



Codes Game

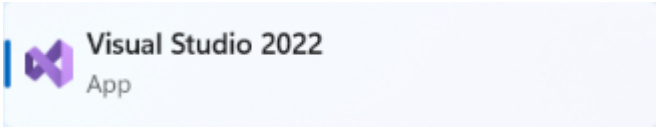
Codes Game

Codes Game shows how to create a game where you need to guess the correct combination of four numbers between *1* and *9* using a toolkit from **NuGet** using the **Windows App SDK**.

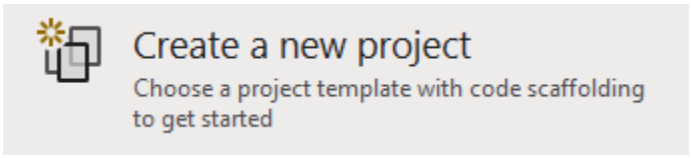
Step 1

Follow **Setup and Start** on how to get **Setup** and **Install** what you need for **Visual Studio 2022** and **Windows App SDK**.

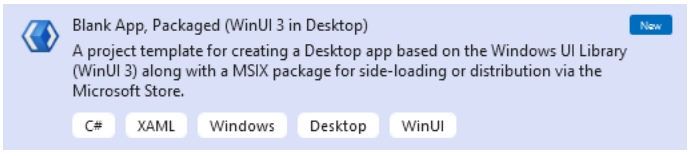
In **Windows 11** choose **Start** and then find or search for **Visual Studio 2022** and then select it.



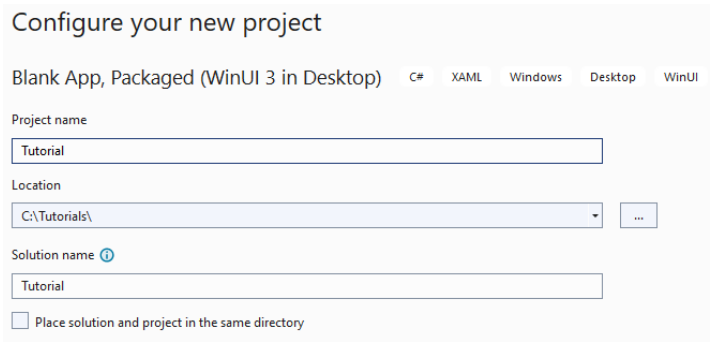
Once **Visual Studio 2022** has started select **Create a new project**.



Then choose the **Blank App, Packages (WinUI in Desktop)** and then select **Next**.

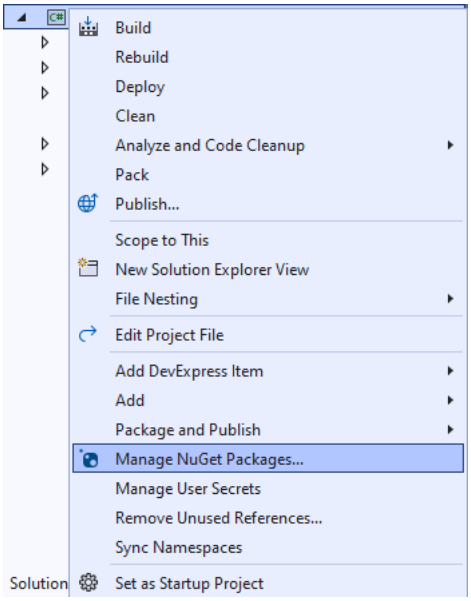


After that in **Configure your new project** type in the **Project name** as *CodesGame*, then select a Location and then select **Create** to start a new **Solution**.



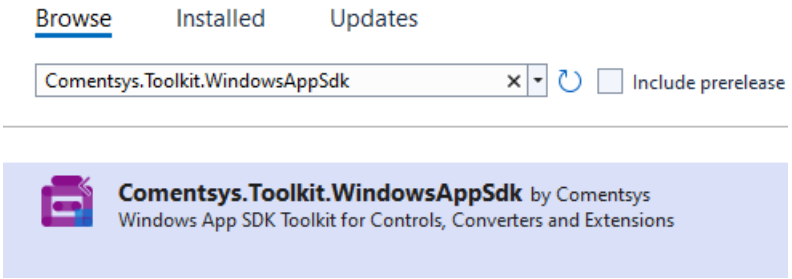
Step 2

Then in **Visual Studio** within **Solution Explorer** for the **Solution**, right click on the **Project** shown below the **Solution** and then select **Manage NuGet Packages...**



Step 3

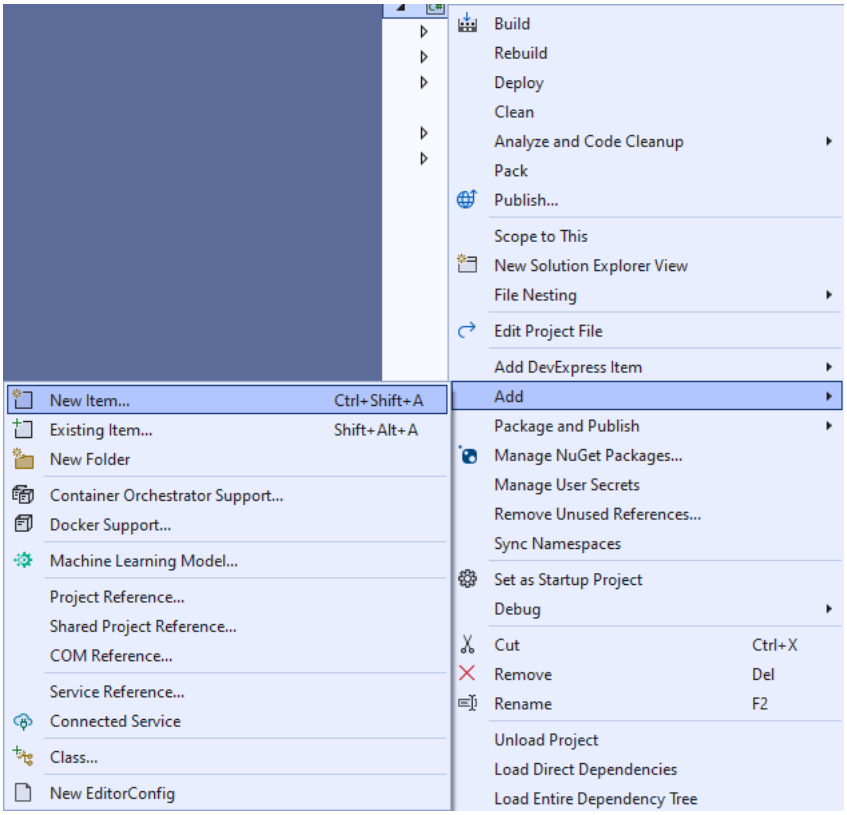
Then in the **NuGet Package Manager** from the **Browse** tab search for **Comentsys.Toolkit.WindowsAppSdk** and then select **Comentsys.Toolkit.WindowsAppSdk by Comentsys** as indicated and select **Install**



This will add the package for **Comentsys.Toolkit.WindowsAppSdk** to your **Project**. If you get the **Preview Changes** screen saying **Visual Studio is about to make changes to this solution. Click OK to proceed with the changes listed below.** You can read the message and then select **OK** to **Install** the package, then you can close the **tab** for **Nuget: CodesGame** by selecting the **x** next to it.

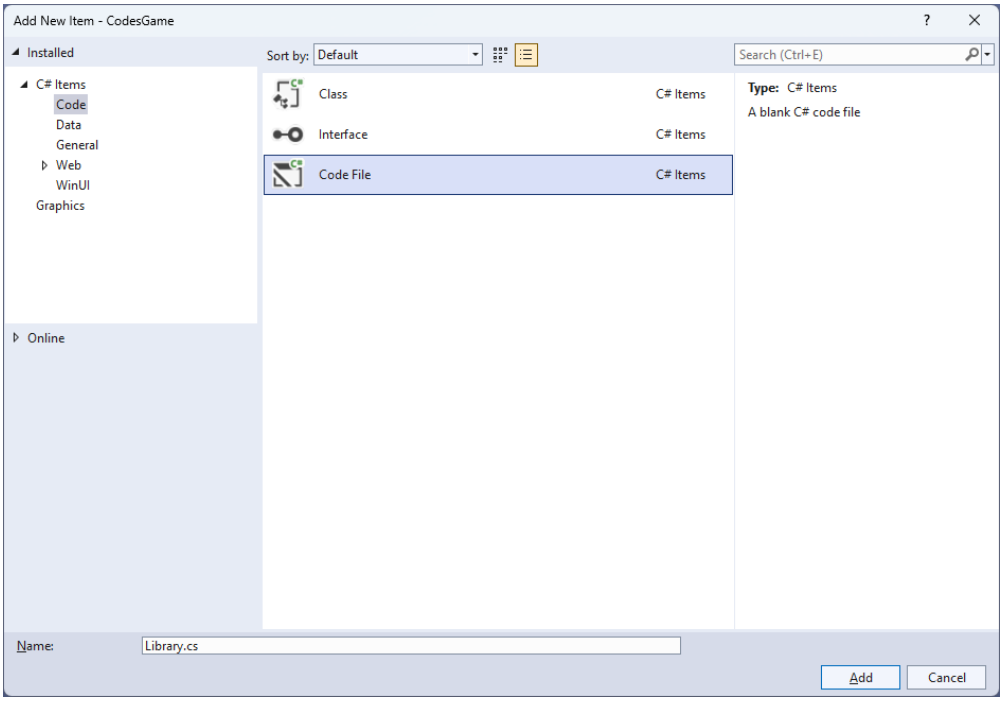
Step 4

Then in **Visual Studio** within **Solution Explorer** for the **Solution**, right click on the **Project** shown below the **Solution** and then select **Add** then **New Item...**



Step 5

Then in **Add New Item** from the **C# Items** list, select **Code** and then select **Code File** from the list next to this, then type in the name of *Library.cs* and then **Click** on **Add**.



Step 6

You will now be in the **View** for the **Code** of *Library.cs*, within this first type the following **Code**:

```
using Comentsys.Toolkit.Binding;
using Comentsys.Toolkit.WindowsAppSdk;
using Microsoft.UI;
using Microsoft.UI.Xaml.Controls;
using Microsoft.UI.Xaml.Data;
using Microsoft.UI.Xaml.Media;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Windows.Input;

namespace CodesGame;

public enum State
{
    None,
    Match
}

// Code Class

// StateToBrushConverter Class

public class Library
{
    // Library Constants, Variables and Choose Method

    // Library GetCode, IsMatch & Setup Method

    // Library Accept & New Method

}
```

Class defined so far *Library.cs* has **using** for package of **Comentsys.Toolkit.WindowsAppSdk** and others along with a **namespace** which allows many classes to be defined together, usually a **class** is defined per file but to make things easier each will be defined in *Library.cs* instead.

Step 7

Still in *Library.cs* for the **`namespace`** of **`CodesGame`** in *Library.cs* you will define a **`class`** after the **Comment** of **`// Code Class`** by typing the following:

```
public class Code : ObservableBase
{
    private int _value;
    private State _state;
    private readonly int _index;
    private readonly Action<int> _action;

    public Code(int index, int value, State state, Action<int> action) =>
        (_index, Value, State, _action) = (index, value, state, action);

    public ICommand Command =>
        new ActionCommandHandler((param) => _action(_index));

    public int Value
    {
        get => _value;
        set => SetProperty(ref _value, value);
    }

    public State State
    {
        get => _state;
        set => SetProperty(ref _state, value);
    }
}
```

`Code` uses the **`class`** from the toolkit of **`ObservableBase`** which will be used for **Data Binding** the **Properties** which include the **`State`** and **`Value`** along with the **`Command`** which will be used to allow interaction with the element using **Commanding**.

Step 8

Still in *Library.cs* for the **`namespace`** of **`CodesGame`** in *Library.cs* you will define a **`class`** after the **Comment** of **`// StateToBrushConverter Class`** by typing the following:

```
public class StateToBrushConverter : IValueConverter
{
    public object Convert(object value, Type targetType,
        object parameter, string language)
    {
        if (value is State state)
        {
            var invert = bool.Parse(parameter as string);
            var none = state == State.None;
            var color = none ^= invert;
            return new SolidColorBrush(color ? Colors.White : Colors.Black);
        }
        return null;
    }

    public object ConvertBack(object value, Type targetType,
        object parameter, string language) =>
        throw new NotImplementedException();
}
```

`StateToBrushConverter` uses the **`interface`** of **`IValueConverter`** for **Data Binding** which will allow the colours of the **Codes** in the game to be represented from either *White* or *Black* as a **`SolidColorBrush`**.

Step 9

While still in the **namespace** of **CodesGame** in *Library.cs* and in the **class** of **Library** after the **Comment** of **// Library Constants, Variables and Choose Method** type in the following **Constants**, **Variables** and **Method**:

```
private const string title = "Codes Game";
private const int max = 9;
private const int total = 4;

private readonly ObservableCollection<Code> _codes = new();
private readonly Random _random = new((int)DateTime.UtcNow.Ticks);

private List<int> _values = new();
private int _turns = 0;

private Dialog _dialog;
private ItemsControl _items;

private List<int> Choose(int minimum, int maximum, int total)
{
    var choose = new List<int>();
    var values = Enumerable.Range(minimum, maximum).ToList();
    for (int index = 0; index < total; index++)
    {
        var value = _random.Next(0, values.Count);
        choose.Add(values[value]);
    }
    return choose;
}
```

Constants are values that are used in the game that will not change and **Variables** are used to store various values for the game. The **Method** of **Choose** will be used to create a list of randomised numbers that are not unique as you can have the same number in a **Code** for the game.

Step 10

While still in the **`namespace`** of **`CodesGame`** in *Library.cs* and in the **`class`** of **`Library`** after the **Comment** of **`// Library GetCode, IsMatch & Setup Method`** type the following **Methods**:

```
private Code GetCode(int index, int value) =>
    new(index, value, State.None, (int i) =>
    {
        var code = _codes[i];
        if (code.State == State.None)
            code.Value = (code.Value == max) ? 1 : code.Value + 1;
    });

private bool IsMatch(int index, int value)
{
    var code = _codes[index];
    return value == code.Value ?
    (code.State = State.Match) == State.Match :
    (code.State = State.None) == State.Match;
}

private void Setup()
{
    _turns = 0;
    _codes.Clear();
    for (int index = 0; index < total; index++)
    {
        _codes.Add(GetCode(index, index + 1));
    }
    _values = Choose(1, max, total);
    _items.ItemsSource = _codes;
}
```

`GetCode` is used to get a part of a **Code** for the game, **`IsMatch`** will check if the selected part of a **Code** matches the part of a **Code** and **`Setup`** is used to initialise the game with a given **Code** using **`Choose`**.

Step 11

While still in the **`namespace`** of **`CodesGame`** in *Library.cs* and in the **`class`** of **`Library`** after the **Comment** of **`// Library Accept & New Method`** type the following **Methods**:

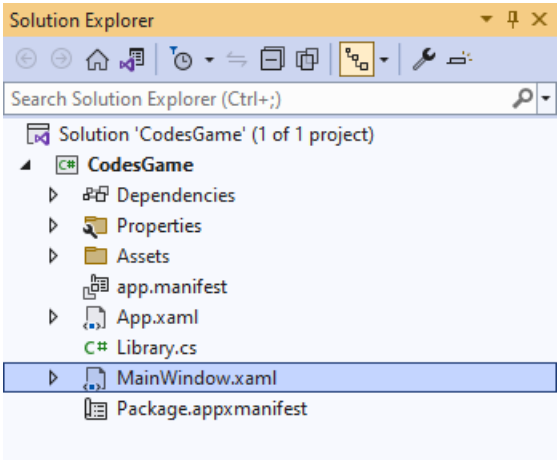
```
public void Accept()
{
    int index = 0;
    int correct = 0;
    foreach (var value in _values)
    {
        if (IsMatch(index, value))
            correct++;
        index++;
    }
    _turns++;
    if (correct == total)
    {
        string code = string.Join(string.Empty, _codes.Select(s => s.Value));
        _dialog.Show($"Matched {code} in {_turns} turns");
        Setup();
    }
}

public void New(ItemsControl items)
{
    _dialog = new Dialog(items.XamlRoot, title);
    _items = items;
    Setup();
}
```

`Accept` will check if the selected parts of the **Code** are correct using **`IsMatch`** if they are all correct then a message will be displayed using the **`Dialog`** and **`New`** will be used to begin a game and uses **`Setup`**.

Step 12

Then from **Solution Explorer** for the **Solution** double-click on **MainWindow.xaml** to see the **XAML** for the **Main Window**.



Step 13

In the **XAML** for **MainWindow.xaml** there be some **XAML** for a **`StackPanel`**, this should be **Removed** by removing the following:

```
<StackPanel Orientation="Horizontal"
HorizontalAlignment="Center" VerticalAlignment="Center">
    <Button x:Name="myButton" Click="myButton_Click">Click Me</Button>
</StackPanel>
```

Step 14

While still in the **XAML** for **MainWindow.xaml** below **`<Window`**, type in the following **XAML**:

```
xmlns:ui="using:Comentsys.Toolkit.WindowsAppSdk"
```

The **XAML** for **`<Window>`** should then look as follows:

```
<Window
    xmlns:ui="using:Comentsys.Toolkit.WindowsAppSdk"
    x:Class="CodesGame.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:CodesGame"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">
```

Step 15

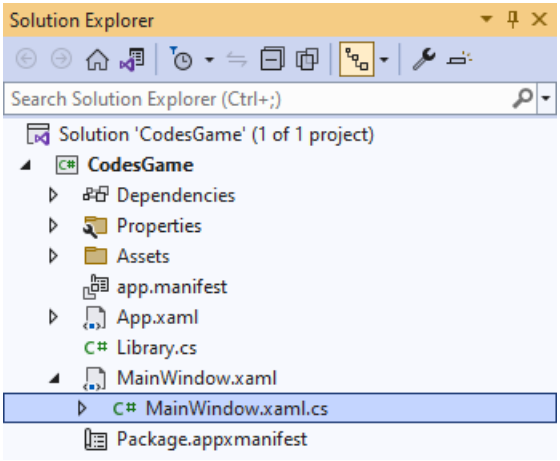
While still in the **XAML** for **MainWindow.xaml** above **</Window>**, type in the following **XAML**:

```
<Grid>
    <Grid.Resources>
        <local:StateToBrushConverter x:Key="StateToBrushConverter"/>
    </Grid.Resources>
    <Viewbox>
        <ItemsControl Margin="50" Name="Display"
            HorizontalAlignment="Center"
            VerticalAlignment="Center" Loaded="New">
            <ItemsControl.ItemTemplate>
                <DataTemplate x:Name="DataTemplate">
                    <Button Command="{Binding Command}">
                        <ui:Piece Value="{Binding Value}" IsSquare="True"
                        Fill="{Binding State, Mode=OneWay,
                        Converter={StaticResource StateToBrushConverter},
                        ConverterParameter=True}"
                        Foreground="{Binding State, Mode=OneWay,
                        Converter={StaticResource StateToBrushConverter},
                        ConverterParameter=False}" />
                    </Button>
                </DataTemplate>
            </ItemsControl.ItemTemplate>
            <ItemsControl.ItemsPanel>
                <ItemsPanelTemplate>
                    <StackPanel Orientation="Horizontal"/>
                </ItemsPanelTemplate>
            </ItemsControl.ItemsPanel>
        </ItemsControl>
    </Viewbox>
    <CommandBar VerticalAlignment="Bottom">
        <AppBarButton Icon="Accept" Label="Accept" Click="Accept"/>
        <AppBarButton Icon="Page2" Label="New" Click="New"/>
    </CommandBar>
</Grid>
```

This **XAML** contains a **Grid** with **Resources** using the **StateToBrushConverter** and also contains a **Viewbox** which will **Scale** an **ItemsControl** which has a **DataTemplate** which contains a **Button** and **Piece** which will be bound using **Data Binding**. It has a **Loaded** event handler for **New** which is also shared by the **AppBarButton** along with one for **Accept**.

Step 16

Then, within **Solution Explorer** for the **Solution** select the arrow next to **MainWindow.xaml** then double-click on **MainWindow.xaml.cs** to see the **Code** for the **Main Window**.



Step 17

In the **Code** for **MainWindow.xaml.cs** there be a **Method** of **`myButton_Click(...)`** this should be **Removed** by removing the following:

```
private void myButton_Click(object sender, RoutedEventArgs e)
{
    myButton.Content = "Clicked";
}
```

Step 18

Once **`myButton_Click(...)`** has been removed, type in the following **Code** below the end of the **Constructor** of **`public MainWindow() { ... }`**:

```
private readonly Library _library = new();

private void Accept(object sender, RoutedEventArgs e) =>
    _library.Accept();

private void New(object sender, RoutedEventArgs e) =>
    _library.New(Display);
```

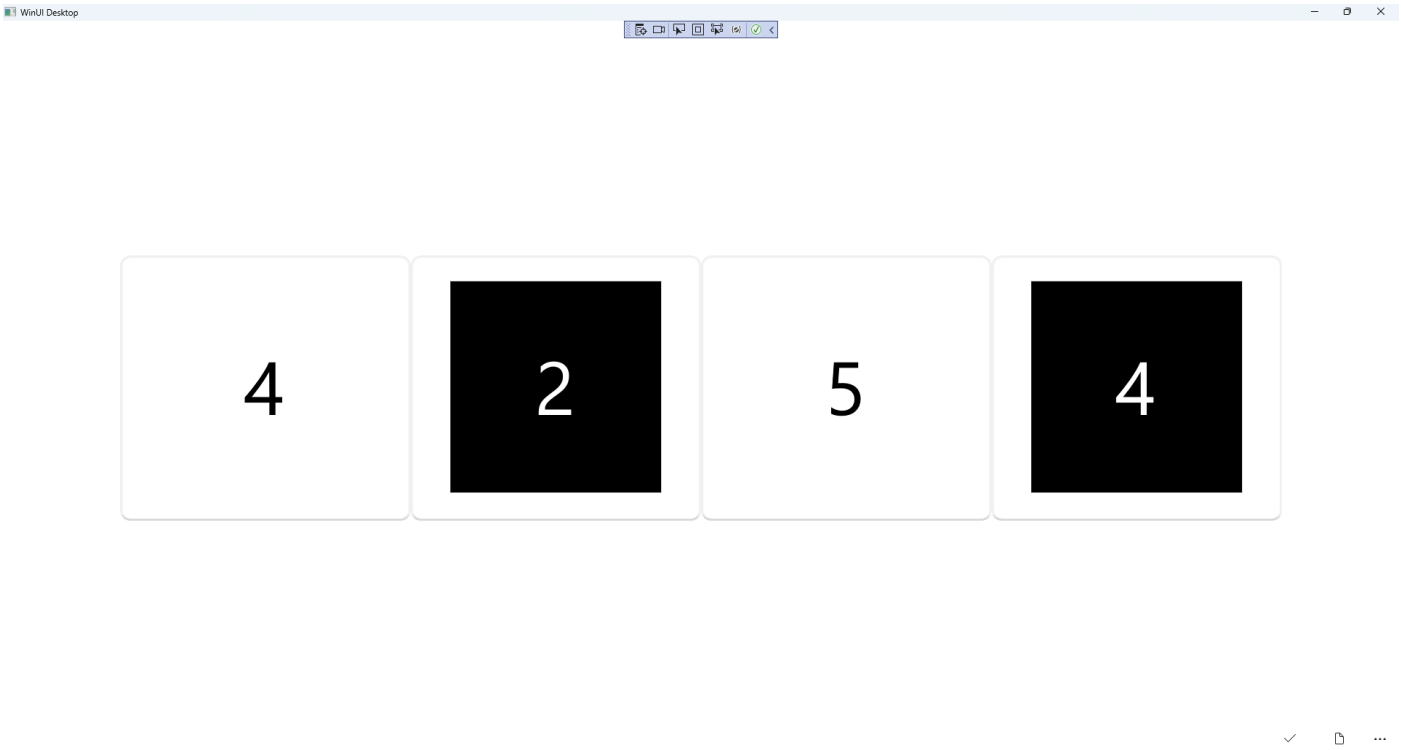
Here an **Instance** of the **Class** of **`Library`** is created then below this is the **Method** of **`Accept`** and **`New`** that will be used with **Event Handler** from the **XAML**, these **Methods** use Arrow Syntax with the **`=>`** for an Expression Body which is useful when a **Method** only has one line.

Step 19

That completes the **Windows App SDK** application. In **Visual Studio 2022** from the **Toolbar** select **CodesGame (Package)** to **Start** the application.

Step 20

Once running you win the game by selecting four numbers between *1* and *9* to guess the secret **Code** by selecting the numbers you can go through them to find the correct combination and once happy select *Accept* if you guessed a number incorrectly then it will turn **Black** with **White** text or if correct it will be **White** with **Black** text, or you can select *New* to start a new game.



Step 21

To **Exit** the **Windows App SDK** application, select the **Close** button from the top right of the application as that concludes this **Tutorial** for **Windows App SDK** from tutorialr.com!