

[](https://www.tutorialr.com/tutorials/)

Windows App SDK

Emoji Game





# Emoji Game

**Emoji Game** shows how you can create a game where you can pick from a set of **Emoji** to identify which

one is the correct one using emoji and 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. | Text  Description automatically generated |
| Once **Visual Studio 2022** has started select **Create a new project**. | **Graphical user interface, text  Description automatically generated** |
| Then choose the **Blank App, Packages (WinUI in Desktop)** and then select **Next**. | **Graphical user interface, text  Description automatically generated** |
| After that in **Configure your new project** type in the **Project name** as *EmojiGame*, then select a Location and then select **Create** to start a new **Solution**. | **Graphical user interface, text, application, email  Description automatically generated** |

## 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…**

Graphical user interface, application

Description automatically generated

## 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**

Graphical user interface, text, application, email

Description automatically generated

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.

## Step 4

Then while still in the **NuGet Package Manager** from the **Browse** tab search for **Comentsys.Assets.FluentEmoji** and then select **Comentsys.Assets.FluentEmoji by Comentsys** as indicated and select **Install**

Graphical user interface, text, application, email

Description automatically generated

This will add the package for **Comentsys.Assets.FluentEmoji** 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: EmojiGame** by selecting the **x** next to it.

## Step 5

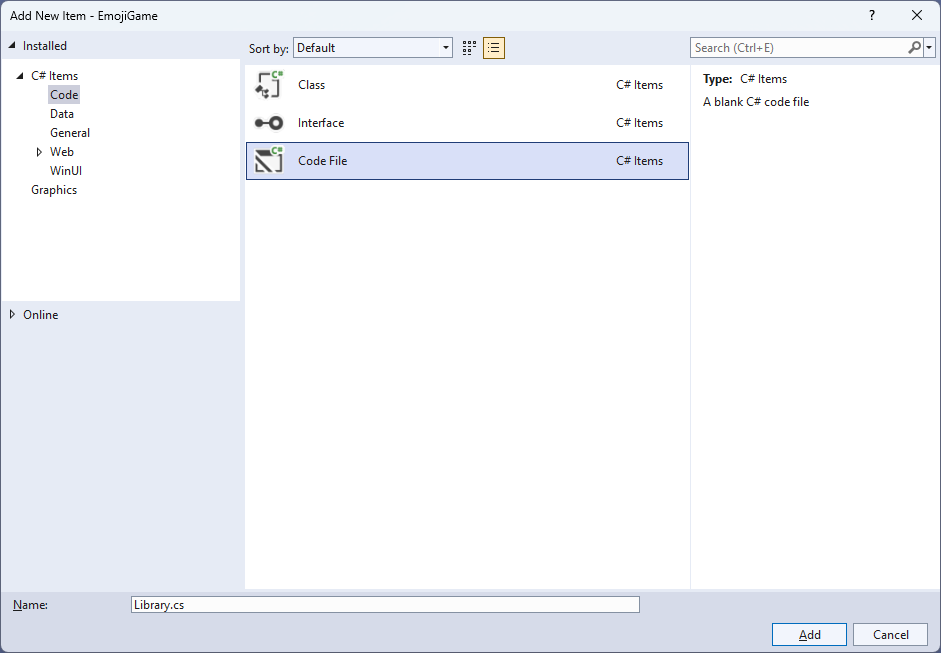
Then in **Visual Studio** within **Solution** **Explorer** for the **Solution**, right click on the **Project** shown below the **Solution** and then select **Add** then **New Item…**

Table

Description automatically generated with low confidence

## Step 6

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 7

You will now be in the **View** for the **Code** of *Library.cs* to define a **namespace** allowing classes to be defined together, usually each is separate but will be defined in *Library.cs* by typing the following **Code** for **using** for **Comentsys.Toolkit.WindowsAppSdk** and others plus an **enum** for **State** and more.

using Comentsys.Assets.FluentEmoji;

using Comentsys.Toolkit.Binding;

using Comentsys.Toolkit.WindowsAppSdk;

using Microsoft.UI;

using Microsoft.UI.Xaml;

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.Text.RegularExpressions;

using System.Threading.Tasks;

using System.Windows.Input;

namespace EmojiGame;

public enum State

{

None,

Correct,

Incorrect

}

// Item Class

public class Board : ObservableBase

{

// Board Constants, Variables, Properties and GetSourceAsync Method

// Board SetSourcesAsync Method

// Board ChooseValues, ChooseUnique, Name, GetQuestion & Indexes Method

// Board Next Method

// Board SetupAsync, Correct & Play Method

}

// StateToBrushConverter Class

public class Library

{

// Library Constants and GetBoundText Method

// Library Layout & New Methods

}

## Step 8

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

public class Item : ActionCommandObservableBase

{

private State \_state = State.None;

public int Index { get; }

public FluentEmojiType Type { get; }

public bool Correct { get; }

public ImageSource Source { get; }

public State State

{

get => \_state;

set => SetProperty(ref \_state, value);

}

public Item(int index, FluentEmojiType type,

bool correct, ImageSource source, Action<int> action) :

base(new ActionCommandHandler((param) => action(index))) =>

(Index, Type, Correct, Source) =

(index, type, correct, source);

}

**Item** represents the elements for the **Emoji** in the game with various **Properties** and uses **ActionCommandObservableBase** from the package of **Comentsys.Toolkit.WindowsAppSdk.**

## Step 9

While still in the **namespace** of **EmojiGame** in *Library.cs* in the **class** of **Board** after the **Comment** of **// Board Constants, Variables, Properties and GetSourceAsync Method** type the following **Constants**, **Variables**, **Properties** and **Method**:

private const string space = " ";

private const int rounds = 12;

private const int options = 2;

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

private Dictionary<FluentEmojiType, ImageSource> \_sources;

private ObservableCollection<Item> \_items = new();

private List<int> \_selected = new();

private List<int> \_options = new();

private List<int> \_indexes = new();

private string \_question;

private string \_message;

private int \_round;

private bool \_over;

public ObservableCollection<Item> Items

{

get => \_items;

set => SetProperty(ref \_items, value);

}

public string Question

{

get => \_question;

set => SetProperty(ref \_question, value);

}

public string Message

{

get => \_message;

set => SetProperty(ref \_message, value);

}

private async Task<ImageSource> GetSourceAsync(FluentEmojiType type) =>

await FlatFluentEmoji.Get(type)

.AsImageSourceAsync();

**Constants** are values that are used for the **Board** that will not change and **Variables** are used to store various values that will be set or changed some of which are exposed using the **Properties** and then there is a **Method** of **GetSourceAsync** which will be used to get the assets for the **Emoji**.

## Step 10

While still in the **namespace** of **EmojiGame** in *Library.cs* in the **class** of **Board** after the **Comment** of **// Board SetSourcesAsync Method** type the following first part of the **Method**:

private async Task SetSourcesAsync() =>

\_sources ??= new Dictionary<FluentEmojiType, ImageSource>()

{

{ FluentEmojiType.GrinningFace,

await GetSourceAsync(FluentEmojiType.GrinningFace) },

{ FluentEmojiType.BeamingFaceWithSmilingEyes,

await GetSourceAsync(FluentEmojiType.BeamingFaceWithSmilingEyes) },

{ FluentEmojiType.FaceWithTearsOfJoy,

await GetSourceAsync(FluentEmojiType.FaceWithTearsOfJoy) },

{ FluentEmojiType.GrinningSquintingFace,

await GetSourceAsync(FluentEmojiType.GrinningSquintingFace) },

{ FluentEmojiType.WinkingFace,

await GetSourceAsync(FluentEmojiType.WinkingFace) },

{ FluentEmojiType.FaceSavoringFood,

await GetSourceAsync(FluentEmojiType.FaceSavoringFood) },

{ FluentEmojiType.SmilingFace,

await GetSourceAsync(FluentEmojiType.SmilingFace) },

{ FluentEmojiType.HuggingFace,

await GetSourceAsync(FluentEmojiType.HuggingFace) },

{ FluentEmojiType.ThinkingFace,

await GetSourceAsync(FluentEmojiType.ThinkingFace) },

{ FluentEmojiType.FaceWithRaisedEyebrow,

await GetSourceAsync(FluentEmojiType.FaceWithRaisedEyebrow) },

{ FluentEmojiType.NeutralFace,

await GetSourceAsync(FluentEmojiType.NeutralFace) },

{ FluentEmojiType.ExpressionlessFace,

await GetSourceAsync(FluentEmojiType.ExpressionlessFace) },

{ FluentEmojiType.FaceWithRollingEyes,

await GetSourceAsync(FluentEmojiType.FaceWithRollingEyes) },

{ FluentEmojiType.PerseveringFace,

await GetSourceAsync(FluentEmojiType.PerseveringFace) },

{ FluentEmojiType.FaceWithOpenMouth,

await GetSourceAsync(FluentEmojiType.FaceWithOpenMouth) },

{ FluentEmojiType.HushedFace,

await GetSourceAsync(FluentEmojiType.HushedFace) },

{ FluentEmojiType.SleepyFace,

await GetSourceAsync(FluentEmojiType.SleepyFace) },

{ FluentEmojiType.TiredFace,

await GetSourceAsync(FluentEmojiType.TiredFace) },

{ FluentEmojiType.SleepingFace,

await GetSourceAsync(FluentEmojiType.SleepingFace) },

You will define the rest of the **Method** of **SetSourcesAsync** in the next **Step**.

## Step 11

While still in the **namespace** of **EmojiGame** in *Library.cs* in the **class** of **Board** after the end of first part of the **Method** for **SetSourcesAsync** of **await GetSourceAsync(FluentEmojiType.SleepingFace) },** from the previous **Step** type the following last part of the **Method**:

{ FluentEmojiType.RelievedFace,

await GetSourceAsync(FluentEmojiType.RelievedFace) },

{ FluentEmojiType.UnamusedFace,

await GetSourceAsync(FluentEmojiType.UnamusedFace) },

{ FluentEmojiType.PensiveFace,

await GetSourceAsync(FluentEmojiType.PensiveFace) },

{ FluentEmojiType.ConfusedFace,

await GetSourceAsync(FluentEmojiType.ConfusedFace) },

{ FluentEmojiType.AstonishedFace,

await GetSourceAsync(FluentEmojiType.AstonishedFace) },

{ FluentEmojiType.FrowningFace,

await GetSourceAsync(FluentEmojiType.FrowningFace) },

{ FluentEmojiType.ConfoundedFace,

await GetSourceAsync(FluentEmojiType.ConfoundedFace) },

{ FluentEmojiType.DisappointedFace,

await GetSourceAsync(FluentEmojiType.DisappointedFace) },

{ FluentEmojiType.WorriedFace,

await GetSourceAsync(FluentEmojiType.WorriedFace) },

{ FluentEmojiType.FaceWithSteamFromNose,

await GetSourceAsync(FluentEmojiType.FaceWithSteamFromNose) },

{ FluentEmojiType.AnguishedFace,

await GetSourceAsync(FluentEmojiType.AnguishedFace) },

{ FluentEmojiType.FearfulFace,

await GetSourceAsync(FluentEmojiType.FearfulFace) },

{ FluentEmojiType.FlushedFace,

await GetSourceAsync(FluentEmojiType.FlushedFace) },

{ FluentEmojiType.ZanyFace,

await GetSourceAsync(FluentEmojiType.ZanyFace) },

{ FluentEmojiType.FaceExhaling,

await GetSourceAsync(FluentEmojiType.FaceExhaling) },

{ FluentEmojiType.AngryFace,

await GetSourceAsync(FluentEmojiType.AngryFace) },

{ FluentEmojiType.NerdFace,

await GetSourceAsync(FluentEmojiType.NerdFace) }

};

**SetSourcesAsync** is used to set the assets for the **Emoji** needed for the game.

## Step 12

While still in the **namespace** of **EmojiGame** in *Library.cs* in the **class** of **Board** after the **Comment** of **// Board ChooseValues, ChooseUnique, Name, GetQuestion & Indexes Method** type the following **Methods**:

private List<int> ChooseValues(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;

}

private List<int> ChooseUnique(int minimum, int maximum, int total) =>

Enumerable.Range(minimum, maximum)

.OrderBy(r => \_random.Next(minimum, maximum))

.Take(total).ToList();

private string Name(FluentEmojiType item) =>

Enum.GetName(typeof(FluentEmojiType), item);

private string GetQuestion(FluentEmojiType item) =>

string.Join(space, new Regex(@"\p{Lu}\p{Ll}\*")

.Matches(Name(item))

.Select(s => s.Value));

private List<int> Indexes(IEnumerable<FluentEmojiType> items) =>

items.Select(item => Array.IndexOf(items.ToArray(), item))

.ToList();

**ChooseValues** is used to get a list of randomised non-unique numbers and **ChooseUnique** is used to get a list of randomised unique numbers. **Name** is used to get the name of an **Emoji** and **GetQuestion** will be used to get the displayed **Emoji** to be guessed with **Indexes** returning the positions of a given **FluentEmojiType**.

## Step 13

While still in the **namespace** of **EmojiGame** in *Library.cs* in the **class** of **Board** after the **Comment** of **// Board Next Method** type the following **Method**:

public bool Next()

{

if (\_round < rounds)

{

Items.Clear();

var emoji = \_sources.Keys.ToArray();

var correct = emoji[\_selected[\_round]];

Question = GetQuestion(correct);

var incorrect = ChooseUnique(0, \_options.Count - 1, options);

var indexOne = \_options[incorrect.First()];

var indexTwo = \_options[incorrect.Last()];

var one = emoji[indexOne];

var two = emoji[indexTwo];

\_options.Remove(indexOne);

\_options.Remove(indexTwo);

var indexes = ChooseUnique(0, options + 1, options + 1);

var items = new List<Item>()

{

new Item(indexes[0], correct, true, \_sources[correct], Play),

new Item(indexes[1], one, false, \_sources[one], Play),

new Item(indexes[2], two, false, \_sources[two], Play)

}.OrderBy(o => o.Index);

foreach (var item in items)

{

Items.Add(item);

}

\_round++;

return true;

}

return false;

}

**Next** is used to proceed through the game to get the next **Emoji** to guess and uses the **Methods** of **GetQuestion** and **ChooseUnique** and will use the **Method** of **Play** which will be defined in the next **Step**.

## Step 14

While still in the **namespace** of **EmojiGame** in *Library.cs* in the **class** of **Board** after the **Comment** of **// Board SetupAsync, Correct & Play Method** type the following **Methods** of **SetupAsync** which will configure the game, **Correct** which will set the **State** accordingly and then the **Method** of **Play**.

public async Task SetupAsync()

{

\_round = 0;

\_over = false;

Question = string.Empty;

Message = string.Empty;

await SetSourcesAsync();

\_indexes = Indexes(\_sources.Keys);

\_selected = ChooseValues(0, \_indexes.Count, rounds);

\_options = \_indexes.Where(index => !\_selected

.Any(selected => selected == index)).ToList();

Next();

}

public bool Correct(Item selected)

{

foreach(var item in Items)

{

item.State = item.Correct ?

State.Correct : State.Incorrect;

}

return selected.Correct;

}

public void Play(int index)

{

if(!\_over)

{

if (Correct(\_items[index]))

{

if(!Next())

{

Message = "Game Over, You Won";

\_over = true;

}

}

else

{

Message = "Incorrect, You Lost!";

\_over = true;

}

}

else

Message = "Game Over";

}

## Step 15

Still in *Library.cs* for the **namespace** of **EmojiGame** 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)

{

return new SolidColorBrush(value switch

{

State.Correct => Colors.Green,

State.Incorrect => Colors.Red,

\_ => Colors.Transparent

});

}

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 **Item** in the game to be represented from either *Green*, *Red* or *Transparent* as a **SolidColorBrush**.

## Step 16

While still in the **namespace** of **EmojiGame** in *Library.cs* and in the **class** of **Library** after the **Comment** of **// Library Constants and GetBoundText Method** type the following **Constants** and **Method**:

private const int font = 20;

private readonly Board \_board = new();

private TextBlock GetBoundText(string property)

{

var text = new TextBlock()

{

FontSize = font,

VerticalAlignment = VerticalAlignment.Center,

HorizontalAlignment = HorizontalAlignment.Center

};

var binding = new Binding()

{

Source = \_board,

Mode = BindingMode.OneWay,

Path = new PropertyPath(property),

UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged

};

BindingOperations.SetBinding(text, TextBlock.TextProperty, binding);

return text;

}

**Constants** are values that are used in the game that will not change and there is also a **Method** of **GetBoundText** which is used to get a **TextBlock** to be used with **Data Binding**.

## Step 17

While still in the **namespace** of **EmojiGame** in *Library.cs* and in the **class** of **Library** after the **Comment** of **// Layout & New** type in the following **Methods**:

private void Layout(Grid grid, DataTemplate itemTemplate,

ItemsPanelTemplate itemsPanel)

{

grid.Children.Clear();

var panel = new StackPanel()

{

Orientation = Orientation.Vertical

};

var question = GetBoundText(nameof(\_board.Question));

panel.Children.Add(question);

var items = new ItemsControl()

{

ItemsSource = \_board.Items,

ItemTemplate = itemTemplate,

ItemsPanel = itemsPanel

};

panel.Children.Add(items);

var message = GetBoundText(nameof(\_board.Message));

panel.Children.Add(message);

grid.Children.Add(panel);

}

public async void New(Grid grid, DataTemplate itemTemplate,

ItemsPanelTemplate itemsPanel)

{

await \_board.SetupAsync();

Layout(grid, itemTemplate, itemsPanel);

}

**Layout** will create the look-and-feel of the game by setting up all the elements including using a **DataTemplate** for the elements and **New** will use **Layout** and setup the **Emoji** used in the game.

## Step 18

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

## Step 19

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 20

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

<Grid>

<Grid.Resources>

<local:StateToBrushConverter x:Key="StateToBrushConverter"/>

<DataTemplate x:Name="DataTemplate">

<Button Command="{Binding Command}">

<Border Height="100" Width="100"

CornerRadius="5" BorderThickness="5"

BorderBrush="{Binding State,

Converter={StaticResource StateToBrushConverter}}">

<Image Source="{Binding Source}"/>

</Border>

</Button>

</DataTemplate>

<ItemsPanelTemplate x:Name="ItemsTemplate">

<StackPanel Orientation="Horizontal"/>

</ItemsPanelTemplate>

</Grid.Resources>

<Viewbox>

<Grid Margin="50" Name="Display"

HorizontalAlignment="Center"

VerticalAlignment="Center" Loaded="New">

<ProgressRing/>

</Grid>

</Viewbox>

<CommandBar VerticalAlignment="Bottom">

<AppBarButton Icon="Page2" Label="New" Click="New"/>

</CommandBar>

</Grid>

This **XAML** contains a **Grid** with a **Viewbox** which will scale a **Grid**. It has a **Loaded** event handler for **New** which is also shared by the **AppBarButton** and defines the **Templates** that will be used in the game.

## Step 21

|  |  |
| --- | --- |
| 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 22

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 23

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 New(object sender, RoutedEventArgs e) =>

\_library.New(Display, DataTemplate, ItemsTemplate);

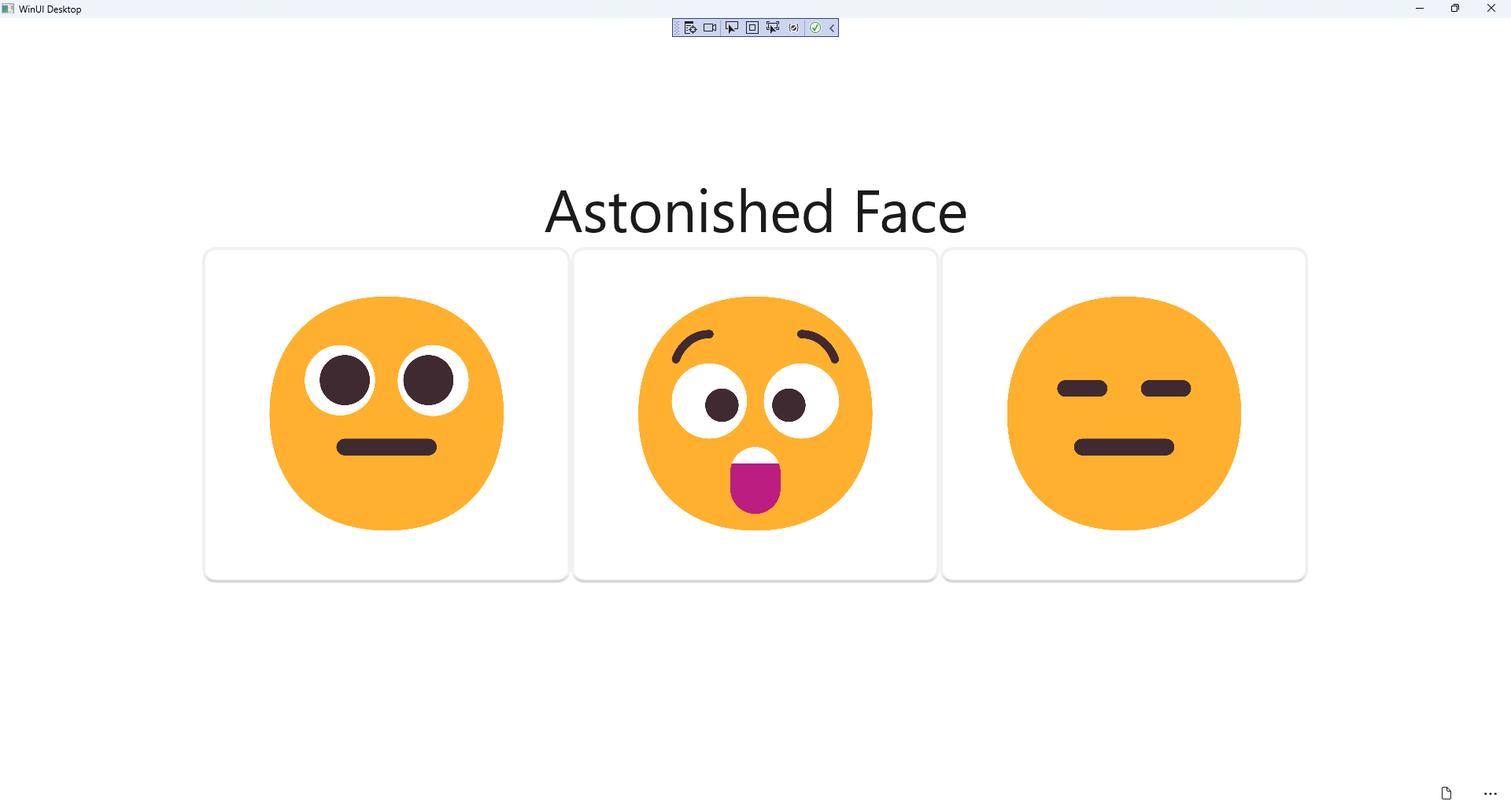
Here an **Instance** of the **Class** of **Library** is created then below this is the **Method** of **New** that will be used with **Event Handler** from the **XAML**, this **Method** uses Arrow Syntax with the **=>** for an Expression Bodywhich is useful when a **Method** only has one line and it also provides the **Templates** needed for the game.

## Step 24

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

## Step 25

Once running you can select an **Emoji** that you think is the one being asked for, if you get it right you progress to the next set of **Emoji** to pick from and if you get all *9* rounds and you win, but get any wrong and you lose or you can select *New* to start a new game.

****

## Step 26

|  |  |
| --- | --- |
| 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](https://tutorialr.com)! |  |