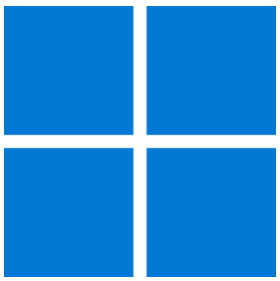
tutorial

Windows App SDK



Match Game

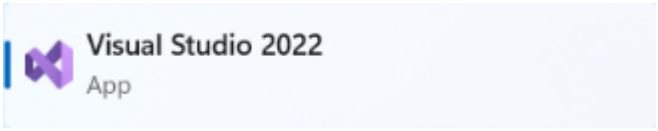
Match Game

Match Game shows how to create a game where you will see a set of **Squares** that are **Black** and **White** then remember where the **White** ones are, then you will see a set of **Grey** ones before seeing a set of **Black** ones and then just need to simply **Match** the correct ones to **White** to proceed using a toolkit from **NuGet** using the **Windows App SDK**.

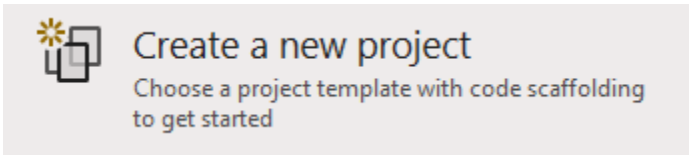
Step 1

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

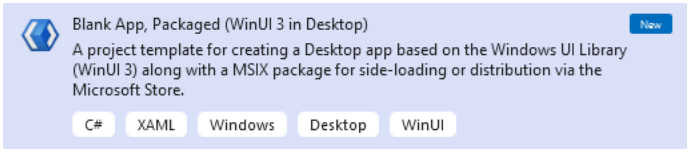
In **Windows 11** choose **Start** and then find or search for **Visual Studio 2022** and then select it.



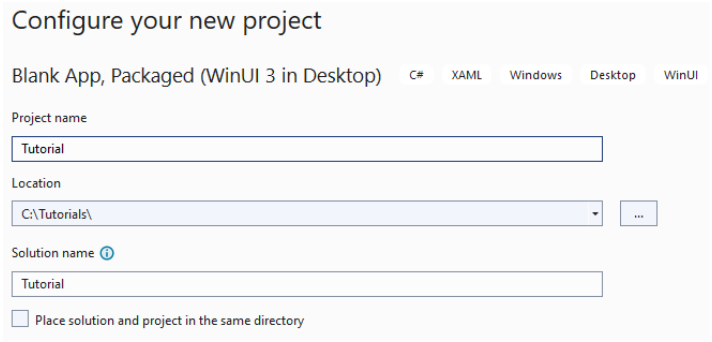
Once **Visual Studio 2022** has started select **Create a new project**.



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

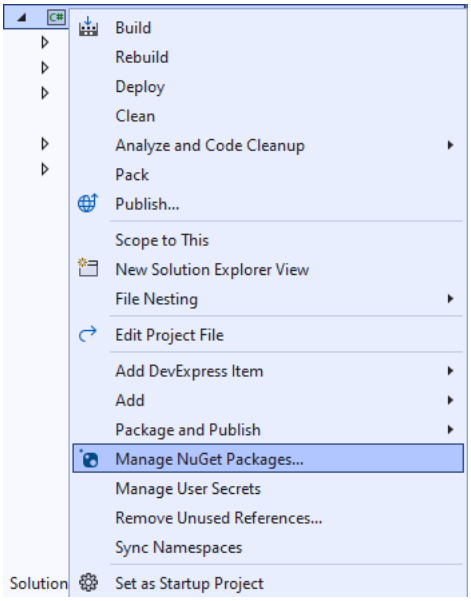


After that in **Configure your new project** type in the **Project name** as *MatchGame*, then select a Location and then select **Create** to start a new **Solution**.



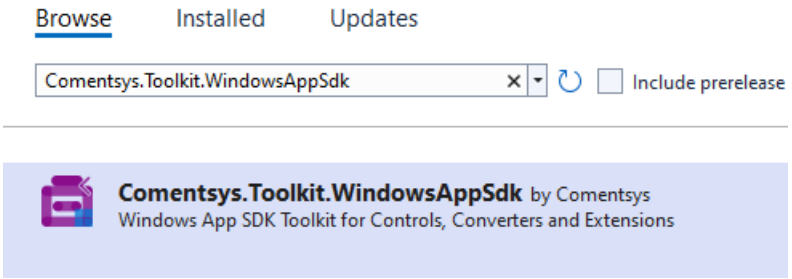
Step 2

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

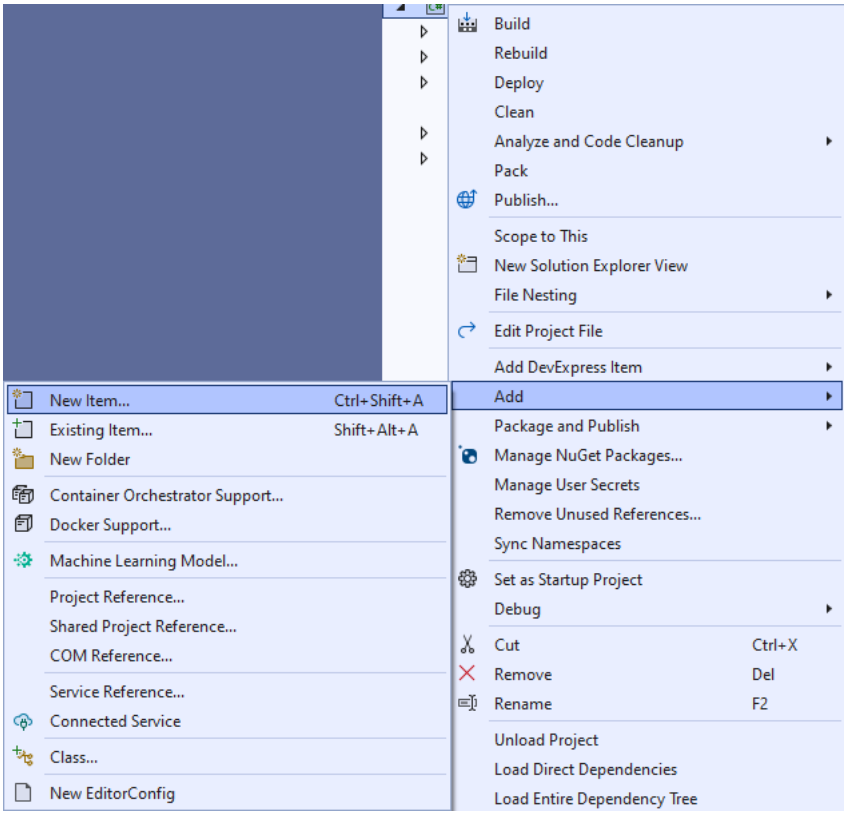
Then in the **NuGet Package Manager** from the **Browse** tab search for **Comentsys.Toolkit.WindowsAppSdk** and then select **Comentsys.Toolkit.WindowsAppSdk by Comentsys** as indicated and select **Install**



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: MatchGame** by selecting the **x** next to it.

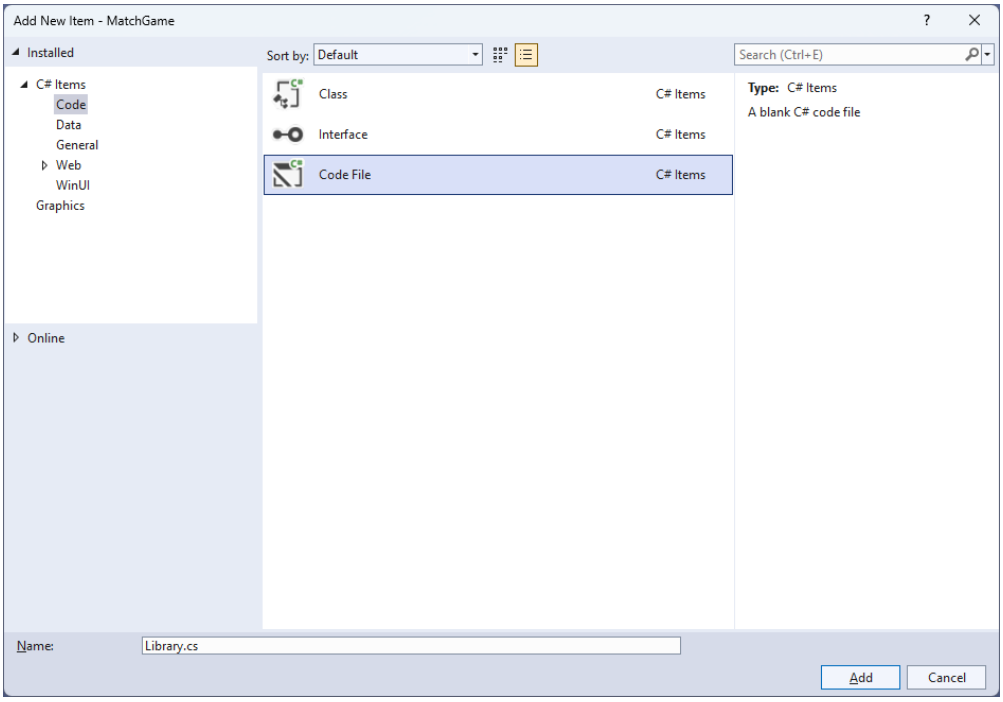
Step 4

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;
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 MatchGame;

public enum State
{
    Wait,
    Off,
    On
}

public enum Match
{
    Memorise,
    Waiting,
    Remember,
    Complete
}

// Item Class

// StateToBrushConverter Class

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

    // Library Set, Change, Update & Pattern Method

    // Library Tick Method

    // Library Play, Layout & 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 **`MatchGame`** in *Library.cs* you will define a **`class`** after the **Comment** of **`// Item Class`** by typing the following:

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

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

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

    public int Index
    {
        get => _index;
        set => SetProperty(ref _index, value);
    }

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

`Item` uses the **`class`** from the toolkit of **`ObservableBase`** which will be used for **Data Binding** the **Properties** which include the **`State`** and **`Index`** 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 **`MatchGame`** 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.On => Colors.White,
                State.Off => Colors.Black,
                _ => Colors.Gray
            });
        }
        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 *White*, *Black* or *Grey* as a **`SolidColorBrush`**.

Step 9

While still in the **namespace** of **MatchGame** 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 = "Match Game";
private const int interval = 1;
private const int total = 16;
private const int delay = 4;
private const int size = 4;

private readonly List<int> _hits = new();
private readonly List<int> _miss = new();
private readonly Dictionary<int, State> _states = new();
private readonly ObservableCollection<Item> _items = new();
private readonly Random _random = new((int)DateTime.UtcNow.Ticks);

private DispatcherTimer _timer;
private Dialog _dialog;
private Match _match;
private int _count;
private int _turns;

private List<int> Choose(int minimum, int maximum, int total) =>
    Enumerable.Range(minimum, maximum)
        .OrderBy(r => _random.Next(minimum, maximum))
            .Take(total).ToList();
```

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 unique randomised numbers.

Step 10

While still in the **namespace** of **MatchGame** in *Library.cs* and in the **class** of **Library** after the **Comment** of **// Library Set, Change, Update & Pattern Method** type the following **Methods**:

```
private void Set(int index, State state) =>
    _items.FirstOrDefault(w => w.Index == index)
        .State = state;

private void Change(Match match) =>
    (_count, _match) = (delay, match);

private void Update(State state)
{
    foreach(var item in _items)
        item.State = state;
}

private void Pattern()
{
    _hits.Clear();
    _miss.Clear();
    _states.Clear();
    Update(State.Off);
    var positions = Choose(0, total, size);
    for (int index = 0; index < total; index++)
    {
        State state = State.Off;
        if (positions.Contains(index))
        {
            state = State.On;
            _states.Add(index, state);
        }
        Set(index, state);
    }
}
```

Set is used to update the **State** for an **Item** and **Change** uses tuple-syntax to update the values being passed in. **Update** is used to set the **State** for all items and **Pattern** is used to display the items that are to be matched using **Set**.

Step 11

While still in the **`namespace`** of **`MatchGame`** in *Library.cs* and in the **`class`** of **`Library`** after the **Comment** of **`// Library Tick Method`** type the following **Method**:

```
private void Tick()
{
    switch (_match)
    {
        case Match.Complete:
            _dialog.Show($"Game Over with {_turns} Matches!");
            _timer?.Stop();
            break;
        case Match.Memorise:
            if (_count == delay)
                Pattern();
            _count--;
            if (_count == 0)
                Change(Match.Waiting);
            break;
        case Match.Waiting:
            if (_count == delay)
                Update(State.Wait);
            _count--;
            if (_count == 0)
                Change(Match.Remember);
            break;
        case Match.Remember:
            if (_count == delay)
                Update(State.Off);
            _count--;
            if (_count == 0)
            {
                if (_hits.Count == size)
                {
                    _turns++;
                    Change(Match.Memorise);
                }
                else
                    Change(Match.Complete);
            }
            break;
    }
}
```

`Tick` will be used by the **Timer** and will perform the actions in the game depending on the value of the **`enum`** of **`Match`** which includes displaying a message when the game is finished using a **`Dialog`** and using the **Methods** of **`Change`** or **`Update`** to display elements in the game as needed.

Step 12

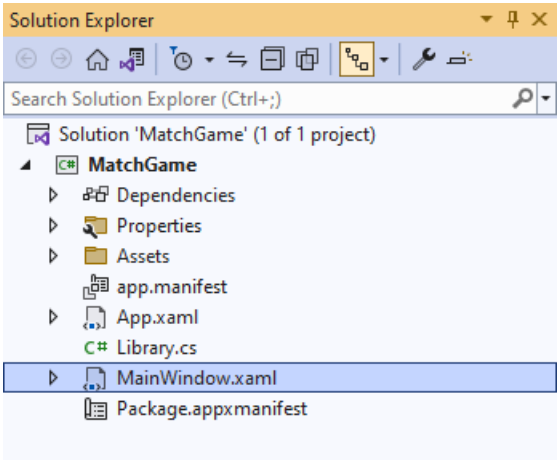
While still in the **`namespace`** of **`CodesGame`** in *Library.cs* and in the **`class`** of **`Library`** after the **Comment** of **`// Library Play, Layout & New Method`** type the following **Methods**:

```
private void Play(int index)
{
    if (_match == Match.Remember &&
        _hits.Count + _miss.Count < size)
    {
        if (_states.ContainsKey(index) &&
            !_hits.Contains(index))
            _hits.Add(index);
        else if(!_states.ContainsKey(index) &&
            !_miss.Contains(index))
            _miss.Add(index);
        Set(index, State.On);
    }
}

private void Layout(ItemsControl display)
{
    for (int index = 0; index < total; index++)
    {
        _items.Add(new Item(index, State.Wait, (int i) => Play(i)));
    }
    display.ItemsSource = _items;
}

public void New(ItemsControl display)
{
    _turns = 1;
    _count = delay;
    _hits.Clear();
    _miss.Clear();
    _items.Clear();
    Layout(display);
    _match = Match.Memorise;
    _dialog = new Dialog(display.XamlRoot, title);
    _timer?.Stop();
    _timer = new DispatcherTimer()
    {
        Interval = TimeSpan.FromSeconds(interval)
    };
    _timer.Tick += (object sender, object e) =>
        Tick();
    _timer.Start();
}
```

`Play` will be used when recalling a pattern of elements and will set them accordingly, **`Layout`** will use **`Play`** and create set of elements and **`New`** will be used to start a new game and setup the **Timer**.

Step 13

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



Step 14

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 15

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="MatchGame.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MatchGame"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">
```

Step 16

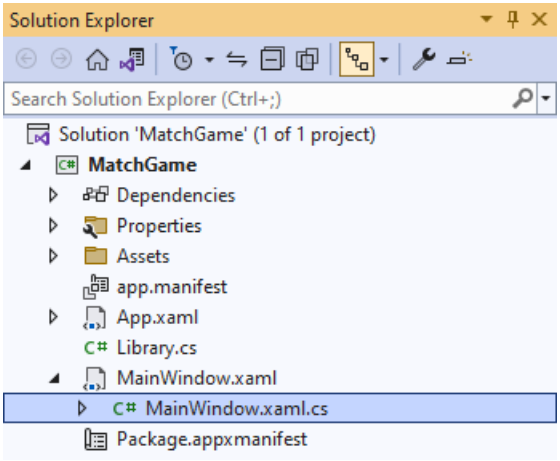
While still in the **XAML** for **MainWindow.xaml** above **`</Window>`**, type in the following **XAML**:

```
<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 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>
                    <VariableSizedWrapGrid MaximumRowsOrColumns="4"/>
                </ItemsPanelTemplate>
            </ItemsControl.ItemsPanel>
        </ItemsControl>
    </Viewbox>
    <CommandBar VerticalAlignment="Bottom">
        <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`**.

Step 17

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 18

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 19

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);
```

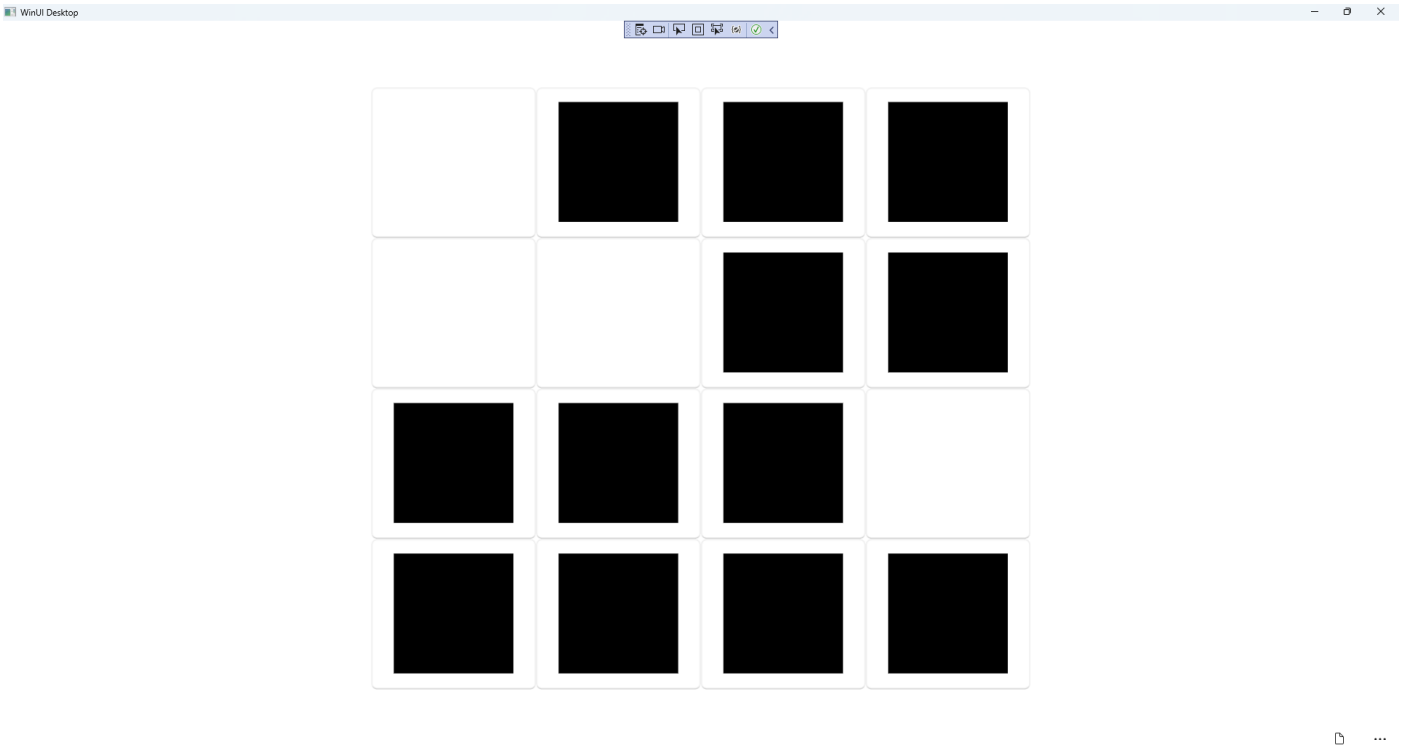
Here an **Instance** of the **Class** of **`Library`** is created then below this is the **Method** of and **`New`** that will be used with **Event Handler** from the **XAML**, this **Method** uses Arrow Syntax with the **`=>`** for an Expression Body which is useful when a **Method** only has one line.

Step 20

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

Step 21

Once running you progress by remembering the positions of the **Squares** that are **White** that are shown before a set of **Squares** that are **Grey** when you see a set of **Squares** that are **Black** if you get them right you proceed to the next ones to **Match** but if you get any wrong you lose the game, or you can select *New* to start a new game.



Step 22

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!