

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

Windows App SDK

Memory Game





# Memory Game

**Memory Game** shows how you can create a simple moon phase pairing game with emoji and with 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 *MemoryGame*, 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**

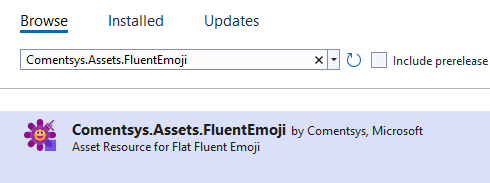
Graphical user interface, text, application, email

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**



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

## Step 5

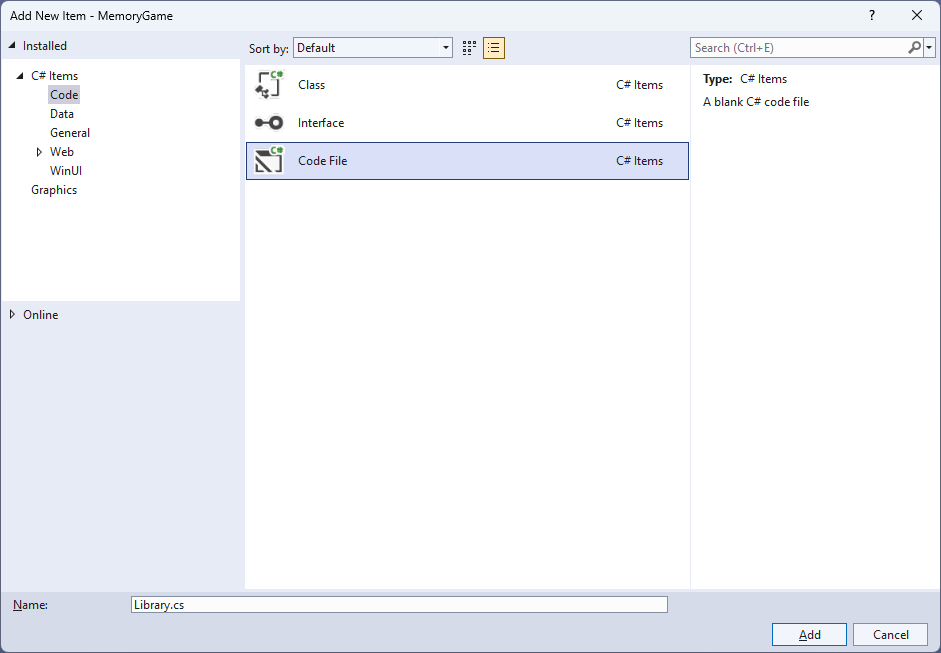
Then in **Visual Studio** within **Solution** **Explorer** for the **Solution**, right click on the **Project** shown below the **Solution** and then select **Add** then **New Item…**

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*, within this first type the following **Code**:

using Comentsys.Assets.FluentEmoji;

using Comentsys.Toolkit.WindowsAppSdk;

using Microsoft.UI.Xaml;

using Microsoft.UI.Xaml.Controls;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

public class Library

{

private const string title = "Memory Game";

private const int size = 4;

private Dialog \_dialog;

private int \_moves = 0;

private int \_row = 0;

private int \_column = 0;

private int \_clicks = 0;

private int \_firstId = 0;

private int \_secondId = 0;

private Button \_first;

private Button \_second;

private readonly int[,] \_board = new int[size, size];

private readonly List<int> \_matches = new();

private static readonly Dictionary<int, FluentEmojiType> \_options = new()

{

{ 1, FluentEmojiType.NewMoon },

{ 2, FluentEmojiType.WaxingCrescentMoon },

{ 3, FluentEmojiType.FirstQuarterMoon },

{ 4, FluentEmojiType.WaxingGibbousMoon },

{ 5, FluentEmojiType.FullMoon },

{ 6, FluentEmojiType.WaningGibbousMoon },

{ 7, FluentEmojiType.LastQuarterMoon },

{ 8, FluentEmojiType.WaningCrescentMoon }

};

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

// Choose, Asset, Match, NoMatch & Compare

// Add

// Layout

// New

}

The **Class** that has been defined in so far *Library.cs* has **using** for the packages that were addedof **Comentsys.Assets.FluentEmoji** and **Comentsys.Toolkit.WindowsAppSdk** amongst others needed. There are also some **const** and **readonly** values for parts of the game and to represent the layout including a **Dictionary** to represent the moon phases that will be paired up.

## Step 8

Still in the **Class** for *Library.cs* after the **Comment** of **// Choose, Asset, Match, NoMatch & Compare** type the following **Methods**:

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

Enumerable.Range(minimum, maximum)

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

.Take(total).ToList();

private Viewbox Asset(int option) => new()

{

Child = new Asset

{

AssetResource = FlatFluentEmoji

.Get(\_options[option])

}

};

private void Match()

{

\_matches.Add(\_firstId);

\_matches.Add(\_secondId);

if (\_matches.Count == size \* size)

\_dialog.Show($"Matched in {\_moves} moves!");

}

private void NoMatch()

{

if (\_first != null)

\_first.Content = null;

if (\_second != null)

\_second.Content = null;

}

private async void Compare()

{

await Task.Delay(TimeSpan.FromSeconds(1.5));

if (\_firstId == \_secondId)

Match();

else

NoMatch();

\_first = null;

\_second = null;

\_moves++;

\_firstId = 0;

\_secondId = 0;

\_clicks = 0;

}

**Choose** will use an enumerable to select a set of numbers that will be randomly chosen. **Asset** will be used to show the **Emoji** for the moon phases. **Match** will check to see if a pair has been selected and if the game is over, **NoMatch** will reset selected items if it isn’t a pair and **Compare** is used to check for a pair.

## Step 9

While still in the **Class** for *Library.cs* after the **Comment** of **// Add** type in the following **Method**:

private void Add(Grid grid, int row, int column)

{

Button button = new()

{

Width = 75,

Height = 75

};

button.Click += (object sender, RoutedEventArgs e) =>

{

button = (Button)sender;

var row = (int)button.GetValue(Grid.RowProperty);

var column = (int)button.GetValue(Grid.ColumnProperty);

int option = \_board[row, column];

if (\_clicks <= 1 && \_matches.IndexOf(option) < 0)

{

// First Choice

if (\_row == 0 && \_column == 0)

{

\_clicks++;

\_firstId = option;

\_first = button;

\_first.Content = Asset(option);

\_row = row;

\_column = column;

}

// Second Choice

else if (!(\_row == row && \_column == column))

{

\_clicks++;

\_secondId = option;

\_second = button;

\_second.Content = Asset(option);

Compare();

\_row = 0;

\_column = 0;

}

}

};

button.SetValue(Grid.ColumnProperty, column);

button.SetValue(Grid.RowProperty, row);

grid.Children.Add(button);

}

**Add** is used to create a **Button** with an **Event Handler** for **Click** which will use **Asset** to set the **Content** and then check when it is either the first choice or the second choice and see if this is a pair with **Compare**.

## Step 10

While still in the **Class** for *Library.cs* after the **Comment** of **// Layout** type the following **Method**:

private void Layout(Grid grid)

{

grid.Children.Clear();

grid.RowDefinitions.Clear();

grid.ColumnDefinitions.Clear();

// Setup Grid

for (int index = 0; index < size; index++)

{

grid.RowDefinitions.Add(new RowDefinition());

grid.ColumnDefinitions.Add(new ColumnDefinition());

}

// Setup Board

for (int row = 0; row < size; row++)

{

for (int column = 0; column < size; column++)

{

Add(grid, row, column);

}

}

}

**Layout** will use **Add** to create the layout for the game.

## Step 11

While still in the **Class** for *Library.cs* after the **Comment** of **// New** type the following **Method**:

public void New(Grid grid)

{

\_dialog = new Dialog(grid.XamlRoot, title);

\_row = 0;

\_moves = 0;

\_column = 0;

\_clicks = 0;

Layout(grid);

int counter = 0;

\_matches.Clear();

List<int> values = new();

// Pairs : Random 1 - 8

while (values.Count <= size \* size)

{

List<int> numbers = Choose(1, size \* 2, size \* 2);

for (int number = 0; number < size \* 2; number++)

{

values.Add(numbers[number]);

}

}

// Board : Random 1 - 16

List<int> indices = Choose(1, size \* size, size \* size);

// Setup Board

for (int column = 0; column < size; column++)

{

for (int row = 0; row < size; row++)

{

\_board[column, row] = values[indices[counter] - 1];

counter++;

}

}

}

**New** will setup the **Dialog** along with initialising the values used in the game for the pairs and for the board itself by using **Layout**.

## 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** above **</Window>**, type in the following **XAML**:

<Grid>

<Viewbox>

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

HorizontalAlignment="Center"

VerticalAlignment="Center" Loaded="New"/>

</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**.

## Step 15

|  |  |
| --- | --- |
| 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 16

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 17

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);

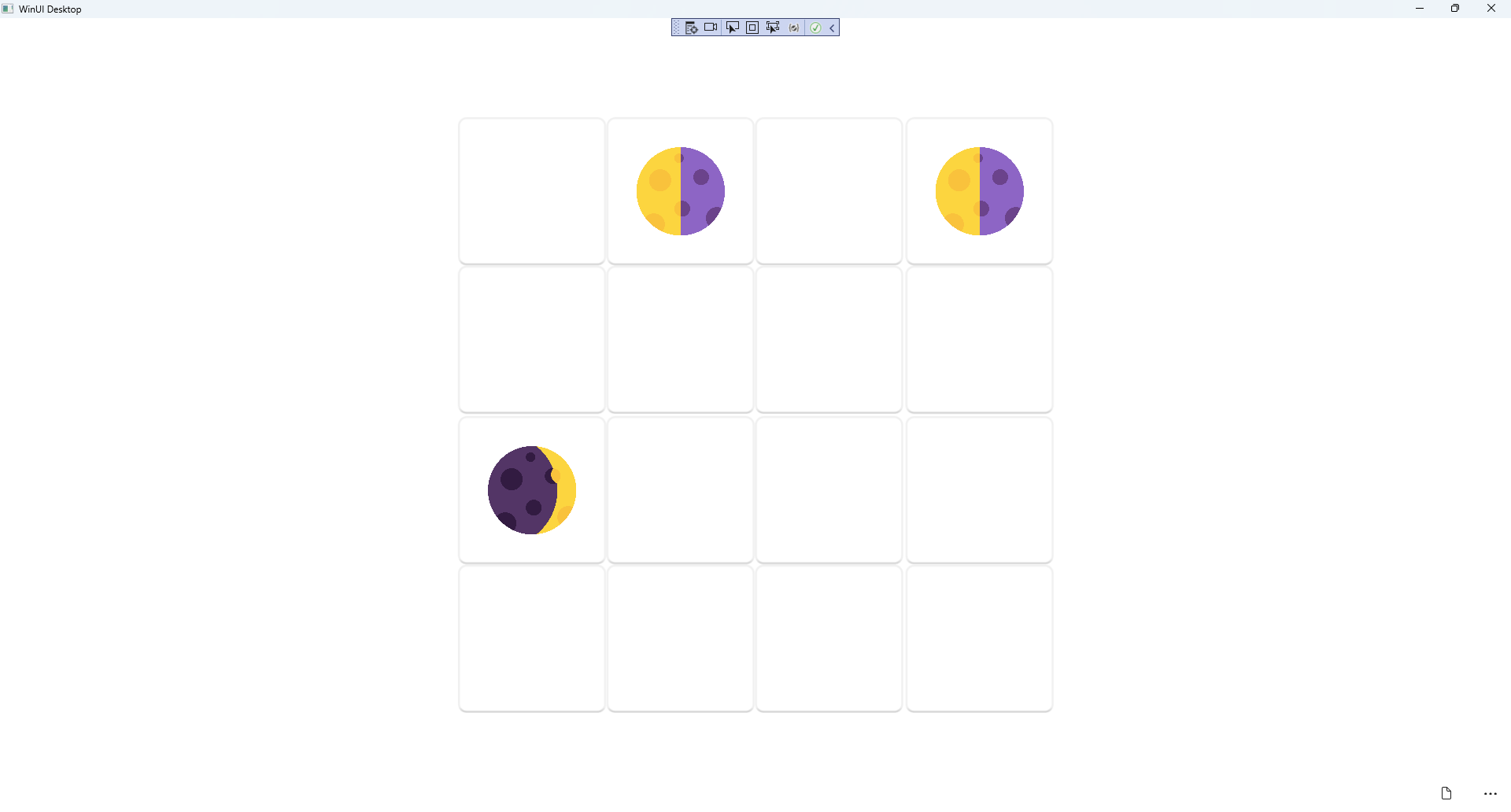
Here an **Instance** of the **Class** of **Library** is created then below this is the **Method** of **New** that will be used with **Event Handler** from the **XAML**, this **Method** uses Arrow Syntax with the **=>** for an Expression Bodywhich is useful when a **Method** only has one line.

## Step 18

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

## Step 19

Once running you can then **Click** on any two **Buttons** to display a phase of the moon, you need to match the phases to make a pair then match all pairs to win or you can restart the game by selecting *New*.

****

## Step 20

|  |  |
| --- | --- |
| 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)! |  |