tutorialr.com

tutorial

Windows App SDK

Tiles Game

tutorialr.com

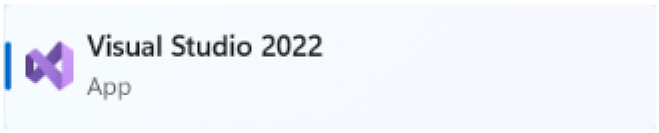
Tiles Game

Tiles Game shows how you can create the game where you must tap all the **Tiles** that are **Black** in the quickest time using a control in a toolkit from **NuGet** using the **Windows App SDK**.

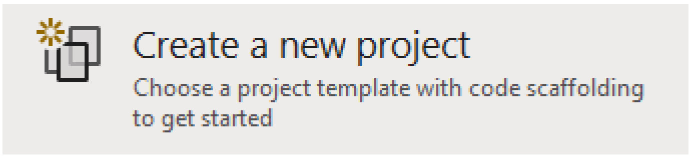
Step 1

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

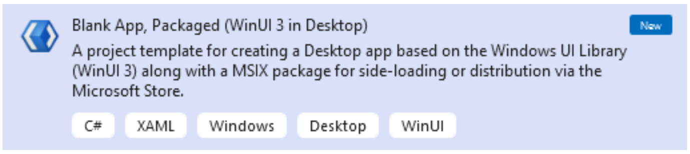
In **Windows 11** choose **Start** and then find or search for **Visual Studio 2022** and then select it.



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

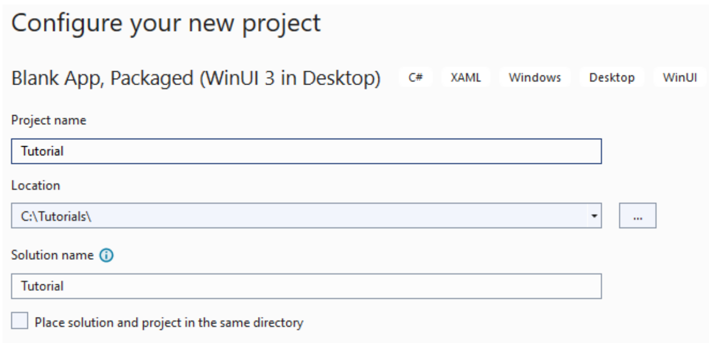

Create a new project
Choose a project template with code scaffolding to get started

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



A project template for creating a Desktop app based on the Windows UI Library (WinUI 3) along with a MSIX package for side-loading or distribution via the Microsoft Store.

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


Configure your new project
Blank App, Packaged (WinUI 3 in Desktop)
C#
XAML
Windows
Desktop
WinUI

Project name: Tutorial

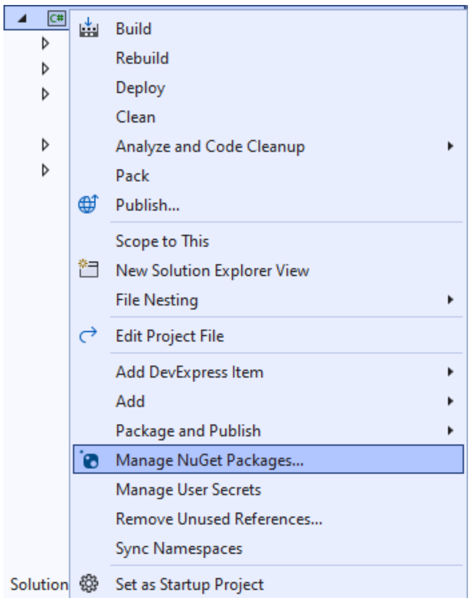
Location: C:\Tutorials\

Solution name: Tutorial

☐ Place solution and project in the same directory

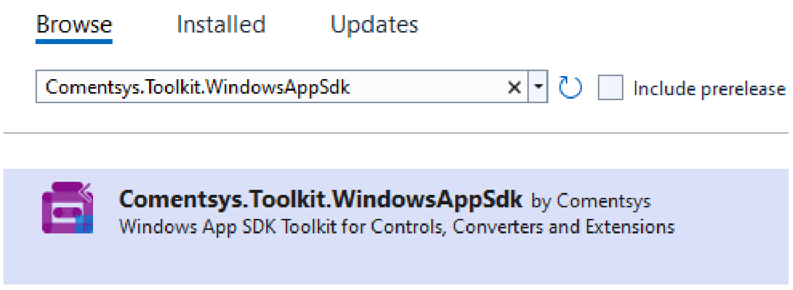
Step 2

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

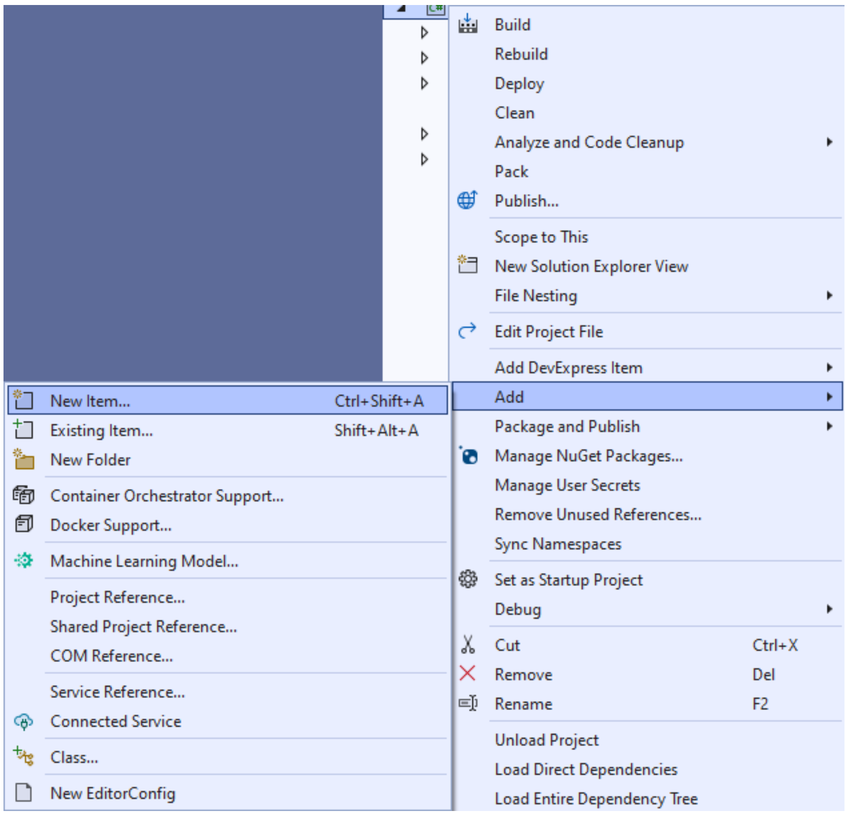
Then in the **NuGet Package Manager** from the **Browse** tab search for **Comentsys.Toolkit.WindowsAppSdk** and then select **Comentsys.Toolkit.WindowsAppSdk by Comentsys** as indicated and select **Install**


Browse
Installed
Updates
Comentsys.Toolkit.WindowsAppSdk
Include prerelease
Comentsys.Toolkit.WindowsAppSdk by Comentsys
Windows App SDK Toolkit for Controls, Converters and Extensions

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

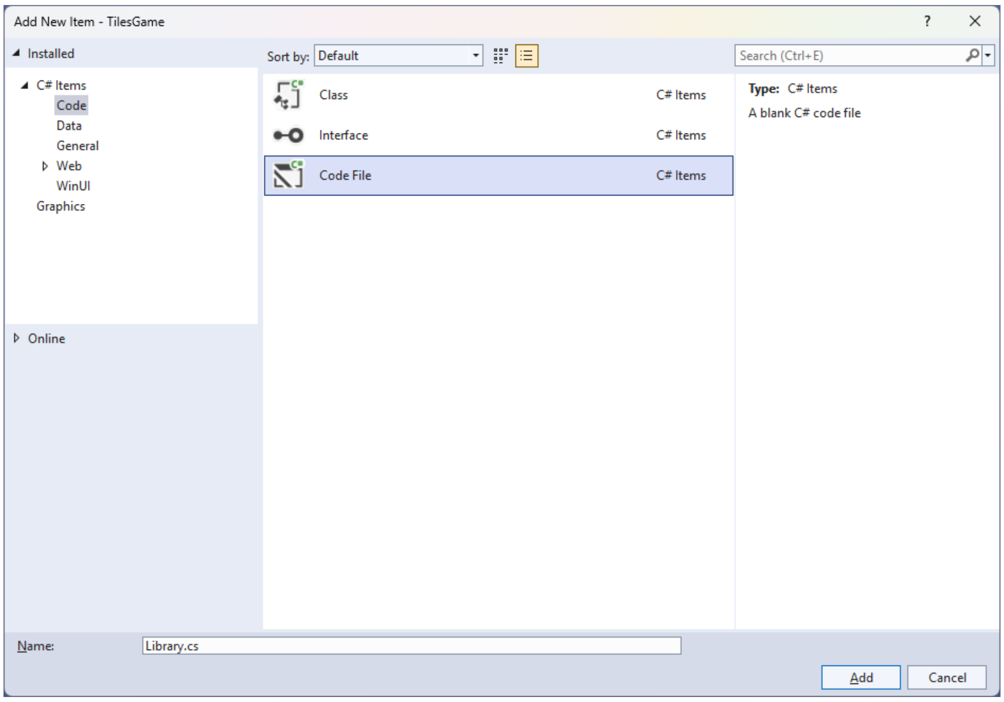
Step 4

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**.



Type: C# Items
A blank C# code file

Name: Library.cs

Step 6

You will now be in the **View** for the **Code** of *Library.cs* then 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**:

```
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.Input;
using Microsoft.UI.Xaml.Media;
using System;
using System.Collections.Generic;
using System.Linq;

namespace TilesGame;

public enum State
{
    White,
    Black,
    Start,
    Finish,
    Correct,
    Incorrect
}

// Item Class

public class Board : ObservableBase
{
    // Board Constants, Members & Properties

    // Board Choose, Set & Start Methods

    // Board Play Method

    // Board Get & New Methods and Constructor
}

public class Library
{
    // Library Constants, Variables & Play Method

    // Library SetPieces Method

    // Library GetBoundText Method

    // Library Layout & New Methods

}
```

Step 7

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

```
public class Item
{
    public int Row { get; set; }

    public int Column { get; set; }

    public State State { get; set; }

    public Item(int row, int column, State state) =>
        (Row, Column, State) = (row, column, state);
}
```

Item has **Properties** for **Row**, **Column** and **State** along with a **Constructor** to set them which will represent an item in the game.

Step 8

Still in *Library.cs* for the **namespace** of **TilesGame** in *Library.cs* in the **class** for **Board** after the **Comment** of **// Board Constants, Members & Properties** type the following **Constants**, **Members** and **Properties**:

```
private const int bound = 1;
private const int timer = 100;
private readonly State[,] _board;
private readonly Random _random = new((int)DateTime.UtcNow.Ticks);
private readonly int _rows;
private readonly int _columns;
private readonly int _levels;
private readonly int _start;
private readonly int _finish;

private int _index;
private int _offset;
private bool _over;
private bool _started;
private TimeSpan _time;
private TimeSpan _best;
private DateTime _when;
private string _message;
private DispatcherTimer _timer;

public TimeSpan Time { get => _time; set => SetProperty(ref _time, value); }
public TimeSpan Best { get => _best; set => SetProperty(ref _best, value); }
public string Message { get => _message; set => SetProperty(ref _message, value); }
```

Step 9

Still in *Library.cs* for the **namespace** of **TilesGame** in *Library.cs* in the **class** for **Board** after the **Comment** of **// Board Choose, Set & Start Methods** type the following **Methods** for **Choose** to pick random numbers. **Set** will update the **State** of game elements and **Start** which will begin the **Timer** for the game.

```
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;
}

private void Set(int columns, int levels)
{
    var values = Choose(0, columns, levels - bound);
    for (int level = 0; level < levels; level++)
    {
        for (int column = 0; column < columns; column++)
        {
            State tile;
            if (level < _finish)
                tile = State.Finish;
            else if (level >= _start)
                tile = State.Start;
            else
                tile = values[level] == column ? State.Black : State.White;
            _board[level, column] = tile;
        }
    }
}

private void Start()
{
    _when = DateTime.UtcNow;
    if (_timer != null)
        _timer.Stop();
    _timer = new DispatcherTimer()
    {
        Interval = TimeSpan.FromMilliseconds(timer)
    };
    _timer.Tick += (object sender, object e) =>
    {
        if(_started && !_over)
            Time = DateTime.UtcNow - _when;
        else
            _timer.Stop();
    };
    _started = true;
    _timer.Start();
}
```

Step 10

Still in *Library.cs* for the **namespace** of **TilesGame** in *Library.cs* in the **class** for **Board** after the **Comment** of **// Board Play Method** type the following **Method:**

```
public void Play(Item item)
{
    if (!_over)
    {
        if (item.State == State.Black)
        {
            if(item.Row == _index)
            {
                if(!_started)
                    Start();
                _board[item.Row, item.Column] = State.Correct;
                _index--;
                if (_offset > 0)
                    _offset--;
                if (_index < _finish)
                {
                    _started = false;
                    Time = DateTime.UtcNow - _when;
                    if (Best == TimeSpan.Zero || Time < Best)
                        Best = Time;
                    Message = $"Completed in {Time:ss\\.fff}!";
                }
            }
            else
            {
                _board[item.Row, item.Column] = State.Incorrect;
                Message = $"Game Over, Hit Wrong Tile!";
                _over = true;
            }
        }
        else if (item.State == State.White)
        {
            _board[item.Row, item.Column] = State.Incorrect;
            Message = $"Game Over, Hit White Tile!";
            _over = true;
        }
    }
    else
        Message = $"Game Over!";
}
```

Play is used when an element is interacted with in the game, the **State** will be checked and if the game is completed the finish time will be displayed, if anything other than the correct **Tile** is interacted with then the game is over, and a message will be displayed or if the game is over that message will be shown.

Step 11

Still in *Library.cs* for the **namespace** of **TilesGame** in *Library.cs* in the **class** for **Board** after the **Comment** of **// Board Get & New Methods and Constructor** type the following **Methods** and **Constructor:**

```
public Item Get(int row, int column) =>
    new(row + _offset, column, _board[row + _offset, column]);

public void New()
{
    _over = false;
    _started = false;
    Time = TimeSpan.Zero;
    Set(_columns, _levels);
    _offset = _levels - _rows;
    _index = _levels - (bound * 2);
    Message = "Don't Hit White Tiles!";
}

public Board(int rows, int columns, int levels)
{
    _rows = rows;
    _columns = columns;
    _levels = levels;
    _start = levels - bound;
    _finish = rows - bound;
    _board = new State[levels, columns];
    New();
}
```

Get will be used to determine what element is at the given position, **New** will start a new game and the **Constructor** for **Board** is used to setup the game.

Step 12

While still in the **`namespace`** of **`TilesGame`** in *Library.cs* and in the **`class`** of **`Library`** after the **Comment** of **`// Library Constants, Variables & Play Method`** type the following **Constants**, **Variables** and **Method**:

```
private const int rows = 6;
private const int columns = 4;
private const int levels = 32;
private const int size = 36;
private const int font = 10;

private readonly Dictionary<State, SolidColorBrush> _brushes = new()
{
    { State.White, new SolidColorBrush(Colors.White) },
    { State.Black, new SolidColorBrush(Colors.Black) },
    { State.Start, new SolidColorBrush(Colors.Gold) },
    { State.Finish, new SolidColorBrush(Colors.ForestGreen) },
    { State.Correct, new SolidColorBrush(Colors.Gray) },
    { State.Incorrect, new SolidColorBrush(Colors.IndianRed) }
};
private readonly Board _board = new(rows, columns, levels);

private Piece[,] _pieces;
private Grid _grid;

private void Play(Item selected)
{
    _board.Play(selected);
    for (int row = 0; row < rows; row++)
    {
        for (int column = 0; column < columns; column++)
        {
            var item = _board.Get(row, column);
            var piece = _pieces[row, column];
            piece.Tag = item;
            piece.Fill = _brushes[item.State];
        }
    }
}
```

Constants are values that are used in the game that will not change and **Variables** are used to store various values for the game, there is also the **Method** of **`Play`** which will be used to perform an action in the game for a given **Tile** or **`Item`**.

Step 13

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

```
private Piece[,] SetPieces(Grid grid)
{
    grid.Children.Clear();
    var pieces = new Piece[rows, columns];
    for (int row = 0; row < rows; row++)
    {
        grid.RowDefinitions.Add(new RowDefinition());
        for (int column = 0; column < columns; column++)
        {
            if (row == 0)
                grid.ColumnDefinitions.Add(new ColumnDefinition());
            var item = _board.Get(row, column);
            var piece = new Piece()
            {
                Tag = item,
                Width = size,
                Height = size,
                IsSquare = true,
                Fill = _brushes[item.State],
                Stroke = new SolidColorBrush(Colors.WhiteSmoke)
            };
            piece.Tapped += (object sender, TappedRoutedEventArgs e) =>
                Play((Item)(sender as Piece).Tag);
            Grid.SetColumn(piece, column);
            Grid.SetRow(piece, row);
            grid.Children.Add(piece);
            pieces[row, column] = piece;
        }
    }
    return pieces;
}
```

`SetPieces` is where the elements that make up the visuals for the game for a **Tile** using **`Piece`**.

Step 14

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

```
private TextBlock GetBoundText(string property, string format = null)
{
    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,
        Converter = new StringFormatConverter(),
        ConverterParameter = format
    };
    BindingOperations.SetBinding(text, TextBlock.TextProperty, binding);
    return text;
}
```

`GetBoundText` is used to get a **`TextBlock`** that will be used with **Data Binding** to output the messages from the game such as to indicate the current time, whether the game is competed or if the game is over.

Step 15

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

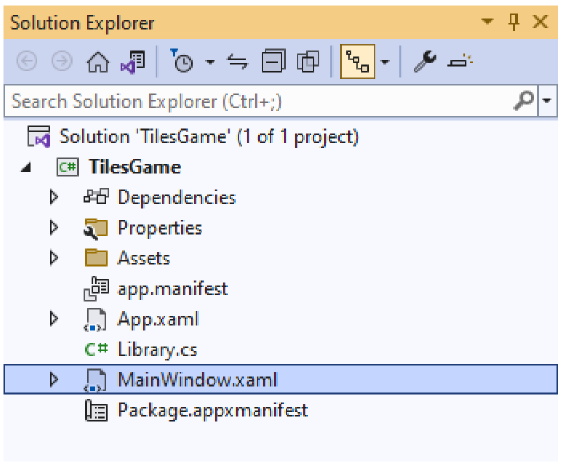
```
private void Layout(Grid grid)
{
    grid.Children.Clear();
    var panel = new StackPanel()
    {
        Orientation = Orientation.Horizontal
    };
    var time = GetBoundText(nameof(_board.Time), "Time: {0:ss\\.fff}");
    panel.Children.Add(time);
    var inner = new StackPanel()
    {
        Orientation = Orientation.Vertical
    };
    var message = GetBoundText(nameof(_board.Message));
    inner.Children.Add(message);
    _grid = new()
    {
        VerticalAlignment = VerticalAlignment.Top,
        HorizontalAlignment = HorizontalAlignment.Center
    };
    _pieces = SetPieces(_grid);
    inner.Children.Add(_grid);
    panel.Children.Add(inner);
    var best = GetBoundText(nameof(_board.Best), "Best: {0:ss\\.fff}");
    panel.Children.Add(best);
    grid.Children.Add(panel);
}

public void New(Grid grid)
{
    _board.New();
    Layout(grid);
}
```

`Layout` will create the look-and-feel of the game including to display the messages using **`GetBoundText`** and **`New`** will start a new game.

Step 16

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



Step 17

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 18

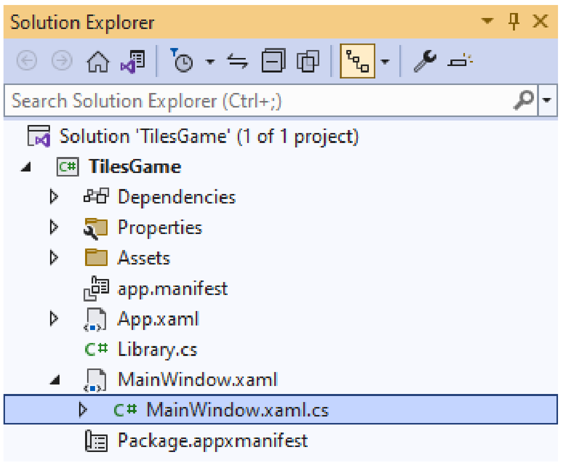
While still in the **XAML** for **MainWindow.xaml** above **`</Window>`**, type in the following **XAML**:

```
<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 19

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 20

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 21

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

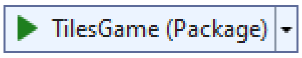
```
private readonly Library _library = new();

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 Body which is useful when a **Method** only has one line.

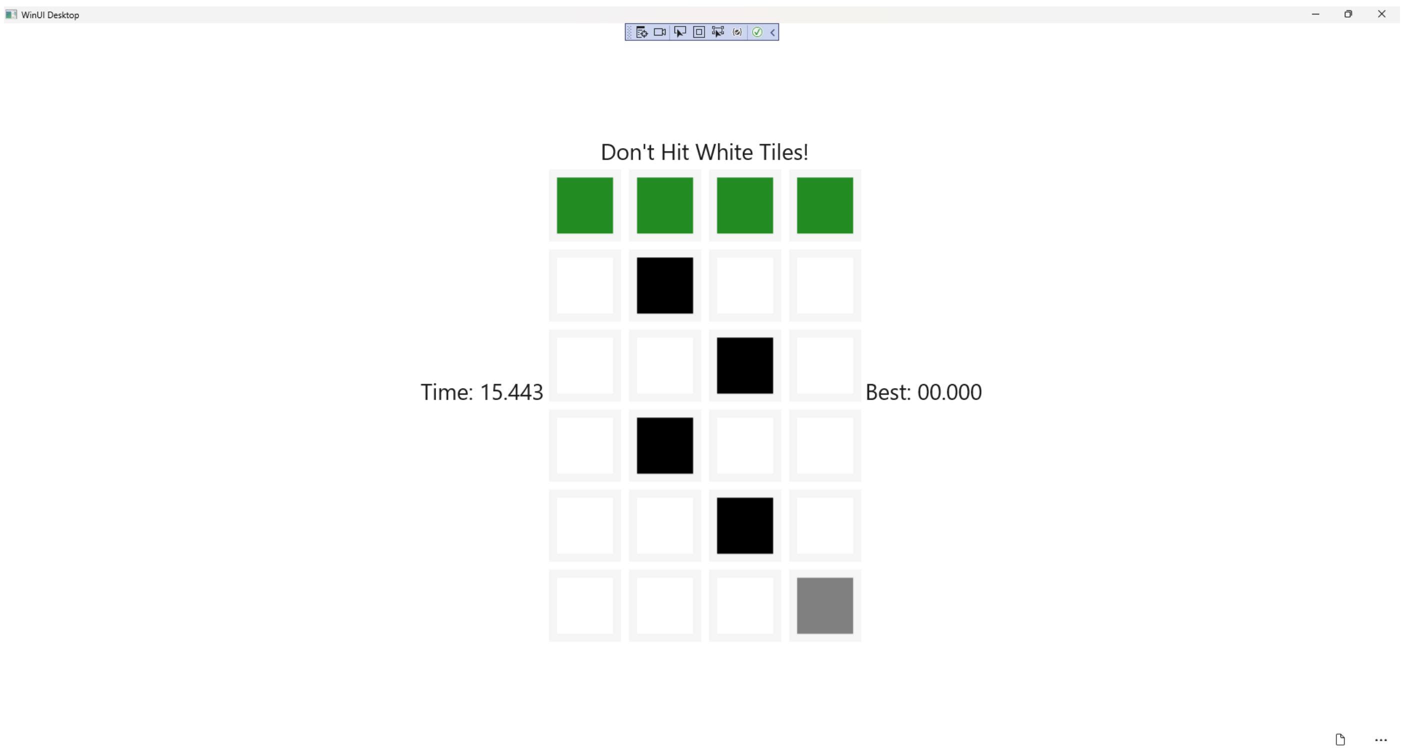
Step 22

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



Step 23

Once running you can tap on the first **Tile** that is **Black** above the **Tiles** that are **Yellow** to begin the game, tap on them all until you then you get to the **Tiles** that are **Green** to win the game, but tap on the wrong **Tile** that is **Black** or a **White** one then you lose the game or select *New* to start a new game.



Step 24

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!