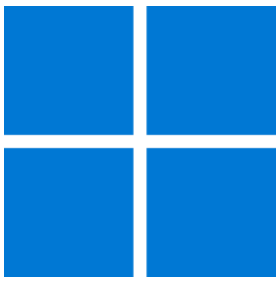
tutorial

Windows App SDK



Mahjong

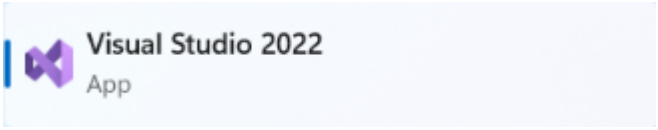
Mahjong

Mahjong shows how you can create the game of **Mahjong** based on the work by cry-inc using game assets and a toolkit from **NuGet** using the **Windows App SDK**.

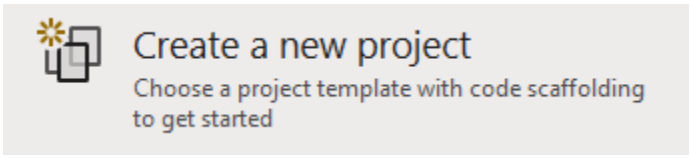
Step 1

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

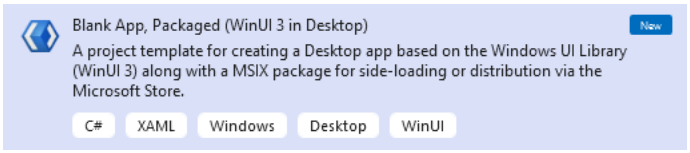
In **Windows 11** choose **Start** and then find or search for **Visual Studio 2022** and then select it.



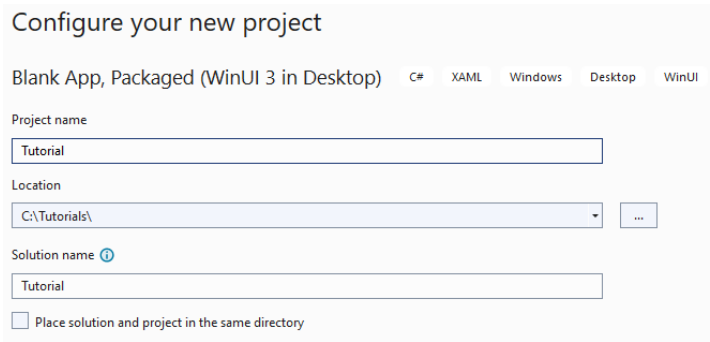
Once **Visual Studio 2022** has started select **Create a new project**.



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

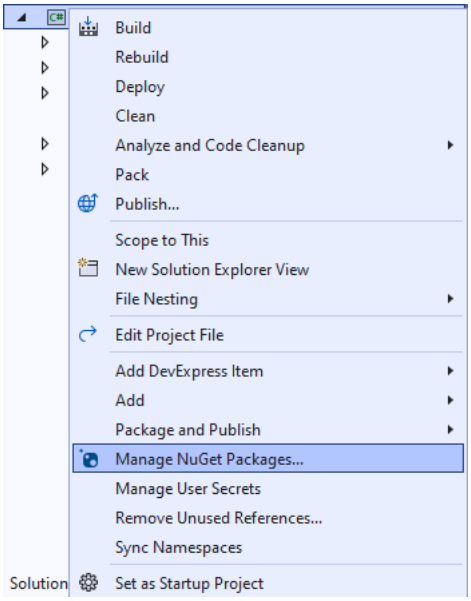


After that in **Configure your new project** type in the **Project name** as *Mahjong*, then select a Location and then select **Create** to start a new **Solution**.



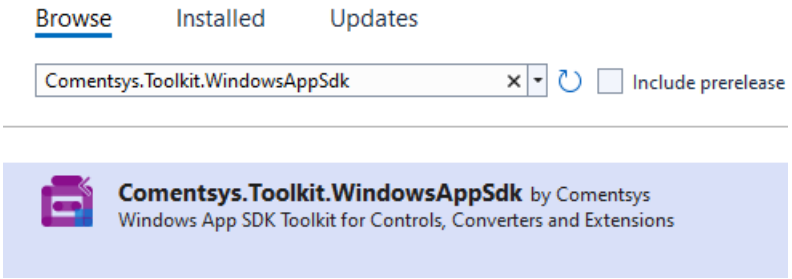
Step 2

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

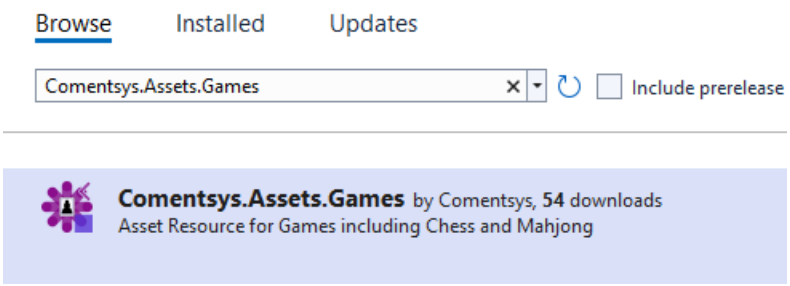
Then in the **NuGet Package Manager** from the **Browse** tab search for **Comentsys.Toolkit.WindowsAppSdk** and then select **Comentsys.Toolkit.WindowsAppSdk by Comentsys** as indicated and select **Install**



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.

Step 4

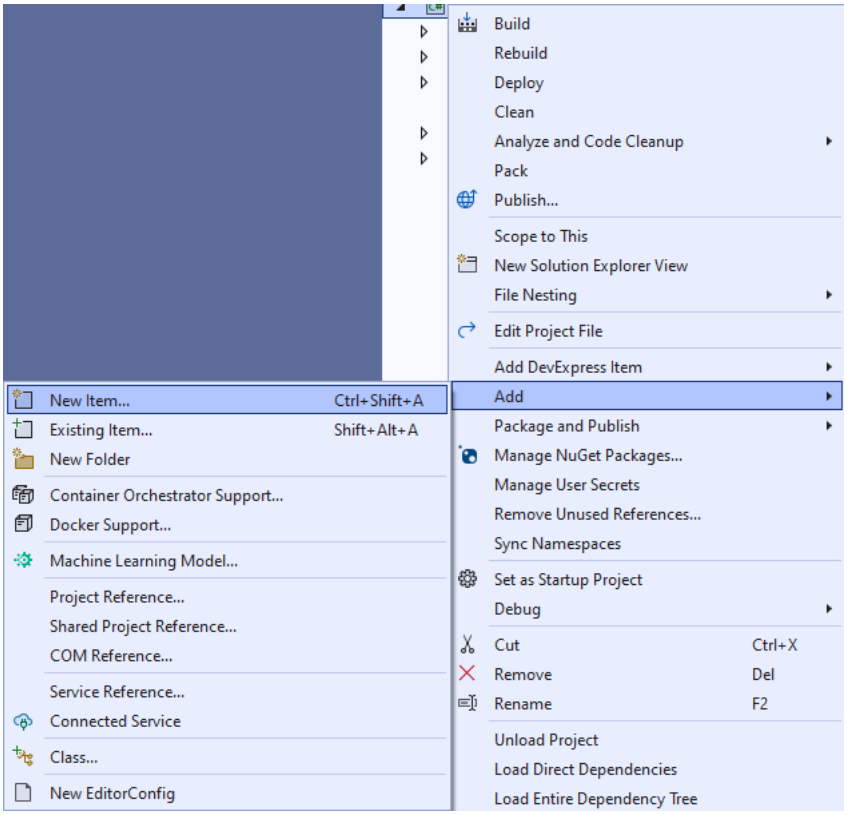
Then while still in the **NuGet Package Manager** from the **Browse** tab search for **Comentsys.Assets.Games** and then select **Comentsys.Assets.Games by Comentsys** as indicated and select **Install**



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

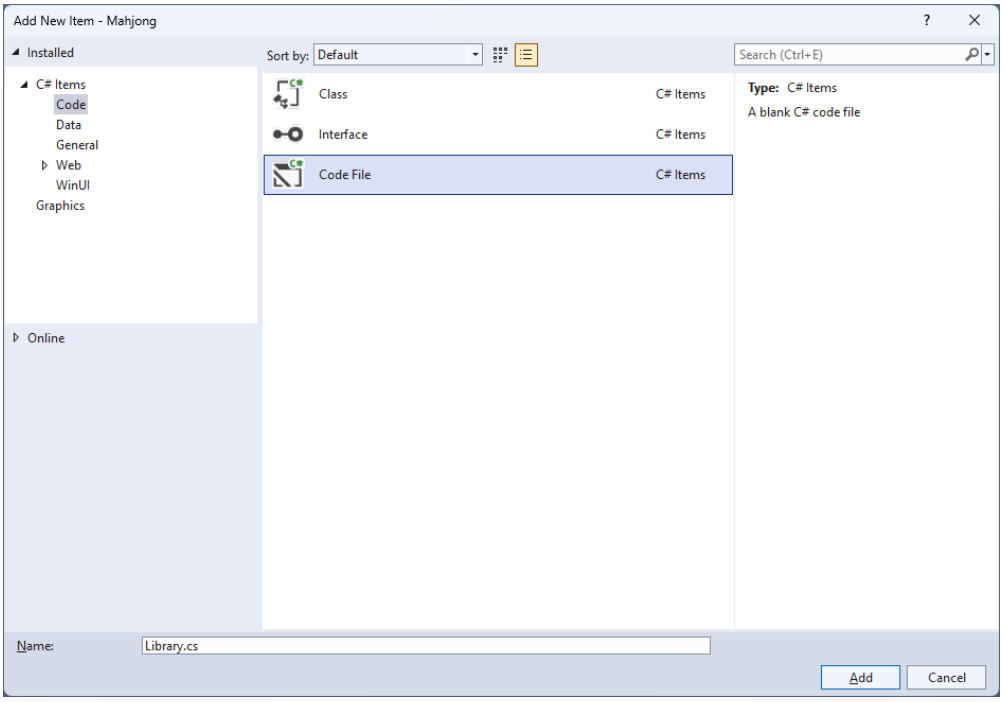
Step 5

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...**



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* 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.Assets.Games;
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 Microsoft.UI.Xaml.Shapes;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Mahjong;

// State & Result Enums and Position & Tile Class

// Pair Class

public class Board
{
    // Board Constants, Variables, Event & Get Methods

    // Can Move, Can Move Up, Can Move Right, Can Move Left, Can Move & Next Move

    // Add, Remove, Removable & Structure

    // Get Hint & Scramble

    // Constructor, Play, Set Hint & Set Disabled

}

// State to Brush Converter

public class Library
{
    // Constants, Variables, Get Source, Set Sources, Get Tile & Shuffle

    // Play

    // Add

    // Layout, Remove, New & Hint

}
```

Step 8

Still in *Library.cs* for the **`namespace`** of **`Mahjong`** you can define an **`enum`** for **`State`** and **`Result`** along with a **`Class`** for **`Position`** and **`Tile`** to represent the **Mahjong** after the **Comment** of **`// State & Result Enums and Position & Tile Class`** by typing the following:

```
public enum State
{
    None,
    Selected,
    Disabled,
    Hint
}

public enum Result
{
    DifferentTypes = 1,
    UnableToMove = 2,
    InvalidMove = 4,
    ValidMove = 8,
    NoMoves = 16,
    Winner = 32,
}

public class Position
{
    public int Row { get; set; }

    public int Column { get; set; }

    public int Index { get; set; }

    public Position(int row, int column, int index) =>
        (Row, Column, Index) = (row, column, index);
}

public class Tile : ObservableBase
{
    private State _state;

    public Position Position { get; }

    public MahjongTileType? Type { get; set; }

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

    public Tile(MahjongTileType type, Position position) =>
        (Type, Position) = (type, Position = position);

    public Tile(Position position) =>
        (Type, Position) = (null, Position = position);
}
```

Step 9

Still in the **`namespace`** of **`Mahjong`** in *Library.cs* after the **Comment** of **`// Pair Class`** type the following:

```
public class Pair
{
    private static readonly Random _random = new((int)DateTime.UtcNow.Ticks);

    public Tile TileOne { get; set; }

    public Tile TileTwo { get; set; }

    public Pair(Tile tileOne, Tile tileTwo) =>
        (TileOne, TileTwo) = (tileOne, tileTwo);

    public static Pair Get(List<Tile> tiles)
    {
        if(tiles.Count < 2)
            throw new Exception();
        var index = _random.Next() % tiles.Count;
        var tileOne = tiles[index];
        tiles.RemoveAt(index);
        index = _random.Next() % tiles.Count;
        var tileTwo = tiles[index];
        tiles.RemoveAt(index);
        return new Pair(tileOne, tileTwo);
    }
}
```

`Pair` will represent a couple of **`Tile`** classes with a **Property** for each one it will also from a **`List`** of them, get a randomly selected set of them.

Step 10

While still in the **namespace** of **Mahjong** in *Library.cs* and the **class** of **Board** and after the **Comment** of **// Board Constants, Variables, Event & Get Methods** type the following:

```
private const int rows = 8;
private const int columns = 10;
private const int indexes = 5;

private static readonly byte[] _layout =
{
    0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
    1, 1, 2, 2, 2, 2, 2, 2, 1, 1,
    1, 1, 2, 3, 4, 4, 3, 2, 1, 1,
    1, 1, 2, 4, 5, 5, 4, 2, 1, 1,
    1, 1, 2, 4, 5, 5, 4, 2, 1, 1,
    1, 1, 2, 3, 4, 4, 3, 2, 1, 1,
    1, 1, 2, 2, 2, 2, 2, 2, 1, 1,
    0, 1, 1, 1, 1, 1, 1, 1, 1, 0
};
private static readonly Random _random = new((int)DateTime.UtcNow.Ticks);

private static readonly List<MahjongTileType> _types =
    Enum.GetValues(typeof(MahjongTileType))
    .Cast<MahjongTileType>()
    .Where(w => w != MahjongTileType.Back)
    .ToList();

private readonly List<Tile> _tiles;

public delegate void RemovedEventHandler(Tile tile);

public event RemovedEventHandler Removed;

public IEnumerable<Tile> Get(int row, int column) =>
    _tiles.Where(w => w.Position.Row == row
    && w.Position.Column == column)
        .OrderBy(o => o.Position.Index);

private Tile Get(int row, int column, int index) =>
    _tiles.FirstOrDefault(
        f => f.Position.Row == row
    && f.Position.Column == column
    && f.Position.Index == index);

private Tile Get(Tile tile) =>
    Get(tile.Position.Row, tile.Position.Column, tile.Position.Index);
```

Constants define the layout and configuration of the board for the game, there are **Variables** for both the types and tiles themselves along with an **Event Handler** which will be used when playing the game when a tile is removed from the board. Then there are **Methods** that are used to obtain a **Tile** by **row** & **column** along with those plus **index** as well as by **Tile**.

Step 11

While still in the **namespace** of **Mahjong** in *Library.cs* and the **class** of **Board** and after the **Comment** of **// Can Move, Can Move Up, Can Move Right, Can Move Left, Can Move & Next Move** type the following **Methods**:

```
private bool CanMove(Tile tile, int rowOffset, int columnOffset, int indexOffset)
{
    var found = Get(
        tile.Position.Row + rowOffset,
        tile.Position.Column + columnOffset,
        tile.Position.Index + indexOffset
    );
    return found == null || tile == found;
}

private bool CanMoveUp(Tile tile) =>
    CanMove(tile, 0, 0, 1);

private bool CanMoveRight(Tile tile) =>
    CanMove(tile, 1, 0, 0);

private bool CanMoveLeft(Tile tile) =>
    CanMove(tile, -1, 0, 0);

public bool CanMove(Tile tile)
{
    bool up = CanMoveUp(tile);
    bool upLeft = up && CanMoveLeft(tile);
    bool upRight = up && CanMoveRight(tile);
    return upLeft || upRight;
}

private bool NextMove()
{
    var removable = new List<Tile>();
    foreach (var tile in _tiles)
        if (CanMove(tile))
            removable.Add(tile);
    for (int i = 0; i < removable.Count; i++)
        for (int j = 0; j < removable.Count; j++)
            if (j != i && removable[i].Type == removable[j].Type)
                return true;
    return false;
}
```

CanMove, **CanMoveUp**, **CanMoveRight** and **CanMoveLeft** will be used to determine if a **Tile** can be moved in those directions and **CanMove** will be used by **NextMove** to determine the next available move.

Step 12

While still in the **`namespace`** of **`Mahjong`** in *Library.cs* and the **`class`** of **`Board`** and after the **Comment** of **`// Add, Remove, Removable & Structure`** type the following **Methods**:

```
private void Add(Tile tile) =>
    _tiles.Add(tile);

private void Remove(Tile tile)
{
    if (tile == Get(tile))
    {
        _tiles.Remove(tile);
        Removed?.Invoke(tile);
    }
}

private List<Tile> Removable()
{
    List<Tile> removable = new();
    foreach (var tile in _tiles)
        if (CanMove(tile))
            removable.Add(tile);
    foreach (Tile tile in removable)
        Remove(tile);
    return removable;
}

private void Structure()
{
    for (int index = 0; index < indexes; index++)
    {
        for (int row = 0; row < rows; row++)
        {
            for (int column = 0; column < columns; column++)
            {
                var current = _layout[row * columns + column];
                if (current > 0 && index < current)
                    Add(new Tile(new Position(row, column, index)));
            }
        }
    }
}
```

`Add` will be used to add a **`Tile`** to the **`List`** of them and **`Remove`** will not only remove it from the **`List`** but it will also **`Invoke`** the **Event Handler**. **`Removable`** will determine which tiles can be removed and **`Structure`** will create the structure for the tiles in the game.

Step 13

While still in the **`namespace`** of **`Mahjong`** in *Library.cs* and the **`class`** of **`Board`** and after the **Comment** of **`// Get Hint & Scramble`** type the following **Methods**:

```
private Pair GetHint()
{
    var tiles = new List<Tile>();
    foreach (var tile in _tiles)
        if (CanMove(tile))
            tiles.Add(tile);
    for (int i = 0; i < tiles.Count; i++)
    {
        for (int j = 0; j < tiles.Count; j++)
        {
            if (i == j)
                continue;
            if (tiles[i].Type == tiles[j].Type)
                return new Pair(tiles[i], tiles[j]);
        }
    }
    return null;
}

public void Scramble()
{
    List<Pair> reversed = new();
    while (_tiles.Count > 0)
    {
        List<Tile> removable = new();
        removable.AddRange(Removable());
        while (removable.Count > 1)
            reversed.Add(Pair.Get(removable));
        foreach (var tile in removable)
            Add(tile);
    }
    for (int i = reversed.Count - 1; i >= 0; i--)
    {
        int index = _random.Next() % _types.Count;
        reversed[i].TileOne.Type = _types[index];
        reversed[i].TileTwo.Type = _types[index];
        Add(reversed[i].TileOne);
        Add(reversed[i].TileTwo);
    }
}
```

`GetHint` will be used to determine which are the next set of tiles that can be moved anywhere on the gameboard to give a hint to the player of the game by checking which ones can be moved with **`CanMove`** and **`Scramble`** will be used to randomise the tiles used in the game.

Step 14

While still in the **namespace** of **Mahjong** in *Library.cs* and the **class** of **Board** and after the **Comment** of **// Constructor, Play, Set Hint & Set Disabled** type the following **Methods**:

```
public Board()
{
    _tiles = new List<Tile>();
    Structure();
    Scramble();
}

public Result Play(Tile tileOne, Tile tileTwo)
{
    if (tileOne == tileTwo)
        return Result.InvalidMove;
    if (tileOne.Type != tileTwo.Type)
        return Result.DifferentTypes;
    if(!CanMove(tileOne) || !CanMove(tileTwo))
        return Result.UnableToMove;
    Remove(tileOne);
    Remove(tileTwo);
    if(_tiles.Count == 0)
        return Result.Winner;
    var result = Result.ValidMove;
    if(!NextMove())
        result |= Result.NoMoves;
    return result;
}

public void SetHint()
{
    if (_tiles.Count > 0)
    {
        var hint = GetHint();
        if (hint != null)
        {
            hint.TileOne.State = State.Hint;
            hint.TileTwo.State = State.Hint;
        }
    }
}

public void SetDisabled()
{
    if(_tiles.Count > 0)
        foreach(var tile in _tiles)
            tile.State = CanMove(tile) ?
                State.None : State.Disabled;
}
```

Play will be used to check if the action is valid and produce the necessary outcome as well as checking if the action is valid and if the game has been won or no further actions are valid, **SetHint** will be used to indicate which tiles are the hint ones and **SetDisabled** will show which tiles are not valid.

Step 15

While still in the **`namespace`** of **`Mahjong`** in *Library.cs* after the **Comment** of **`// State to Brush Converter`** type the following **Class**:

```
public class StateToBrushConverter : IValueConverter
{
    public object Convert(object value, Type targetType,
        object parameter, string language) =>
        new SolidColorBrush((State)value switch
        {
            State.None => Colors.Transparent,
            State.Selected => Colors.ForestGreen,
            State.Disabled => Colors.DarkSlateGray,
            State.Hint => Colors.CornflowerBlue,
            _ => Colors.Transparent
        });

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

`StateToBrushConverter` defines an **`IValueConverter`** that will be used with **Data Binding**, and this will return a **`SolidColorBrush`** of a given colour depending on the value of the **`State`** used.

Step 16

While still in the **`namespace`** of **`Mahjong`** in *Library.cs* and in the **`class`** of **`Library`** after the **Comment** of **`// Constants, Variables, Get Source, Set Sources, Get Tile & Shuffle`** type the following **Constants**, **Variables** and **Methods**:

```
private const string title = "Mahjong";
private const int rows = 8;
private const int columns = 10;
private const int tile_width = 74;
private const int tile_height = 95;
private const int square_height = 120;
private const int square_width = 90;
private readonly Dictionary<MahjongTileType, ImageSource> _sources = new();

private Board _board = new();
private Dialog _dialog;
private Grid _grid;
private Tile _selected;
private bool _gameOver;

private static async Task<ImageSource> GetSourceAsync(MahjongTileType type) =>
    await MahjongTile.Get(type)
    .AsImageSourceAsync();

private async Task SetSourcesAsync()
{
    if (_sources.Count == 0)
        foreach (var mahjongTileType in Enum.GetValues<MahjongTileType>())
            _sources.Add(mahjongTileType, await GetSourceAsync(mahjongTileType));
}

private ImageSource GetTile(MahjongTileType? type) =>
    type == null ? null : _sources[type.Value];

private void Shuffle()
{
    _board.Scramble();
    _grid.Children.Clear();
    for (int column = 0; column < columns; column++)
        for (int row = 0; row < rows; row++)
            Add(row, column);
}
```

Constants are values that are used in the game that will not change and **Variables** are used to store various values and controls needed for the game. **`GetSourceAsync`**, **`SetSourcesAsync`** and **`GetTile`** are used for the assets for the **Mahjong** tiles and **`Shuffle`** is used to randomise the tiles displayed in the game the **Method** of **`Add`** will be defined later.

Step 17

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

```
private async void Play(Tile tile)
{
    if(!_gameOver)
    {
        if (!_board.CanMove(tile))
            return;
        if (_selected == null || tile == _selected)
        {
            if (_selected == tile)
            {
                tile.State = State.None;
                _selected = null;
            }
            else
            {
                tile.State = State.Selected;
                _selected = tile;
            }
        }
        else
        {
            var state = _board.Play(_selected, tile);
            _board.SetDisabled();
            if (state == Result.Winner)
                _gameOver = true;
            else if ((state & Result.NoMoves) != 0)
            {
                if (await _dialog.ConfirmAsync(
                    "No further moves. Shuffle?", "Yes", "No"))
                    Shuffle();
            }
            _selected = null;
        }
    }
    if(_gameOver)
        _dialog.Show("You Won, Game Over!");
}
```

`Play` will check if the game is over, then will check if there is a valid move then will update the selected **`Tile`** and then once there are two the turn will be checked to see if it is valid then if the game is not over or there are moves still available the game will continue otherwise the game will be over or if no more moves then there will be the option to shuffle the tiles.

Step 18

While still in the **`namespace`** of **`Mahjong`** in *Library.cs* and in the **`class`** of **`Library`** to create the **Method** of **`Add`** to create a *Mahjong* tile and set up **Data Binding** after the **Comment** of **`// Add`** type the following**:**

```
private void Add(int row, int column)
{
    Canvas square = new()
    {
        Width = square_width, Height = square_height
    };
    var tiles = _board.Get(row, column);
    foreach (var tile in tiles)
    {
        Canvas canvas = new()
        {
            Tag = tile, Width = tile_width, Height = tile_height,
        };
        Image image = new()
        {
            Tag = tile,
            Width = tile_width,
            Height = tile_height,
            Source = GetTile(tile.Type)
        };
        image.Tapped += (object sender, TappedRoutedEventArgs e) =>
            Play((sender as Image).Tag as Tile);
        canvas.Children.Add(image);
        var rectangle = new Rectangle()
        {
            Tag = tile,
            Opacity = 0.25,
            Width = tile_width,
            Height = tile_height,
            IsHitTestVisible = false
        };
        var binding = new Binding()
        {
            Source = tile,
            Mode = BindingMode.OneWay,
            Converter = new StateToBrushConverter(),
            Path = new PropertyPath(nameof(tile.State)),
            UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged
        };
        BindingOperations.SetBinding(rectangle, Shape.FillProperty, binding);
        if (!_board.CanMove(tile))
            tile.State = State.Disabled;
        canvas.Children.Add(rectangle);
        Canvas.SetTop(canvas, -(tile.Position.Index * 5));
        square.Children.Add(canvas);
    }
    square.SetValue(Grid.RowProperty, row);
    square.SetValue(Grid.ColumnProperty, column);
    _grid.Children.Add(square);
}
```

Step 19

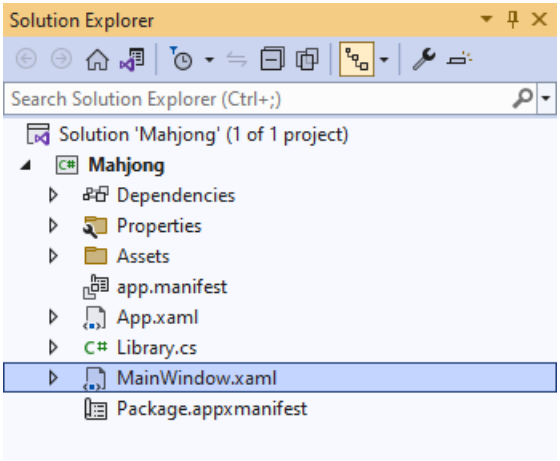
While still in the **namespace** of **Mahjong** in *Library.cs* and in the **class** of **Library** after the **Comment** of **// Layout, Remove, New & Hint** type in the following **Methods**:

```
private void Layout(Grid grid)
{
    grid.Children.Clear();
    _grid = new Grid();
    for (int column = 0; column < columns; column++)
    {
        _grid.RowDefinitions.Add(new RowDefinition());
        for (int row = 0; row < rows; row++)
        {
            if (row == 0)
                _grid.ColumnDefinitions.Add(new ColumnDefinition());
            Add(row, column);
        }
    }
    grid.Children.Add(_grid);
}

private void Remove(Tile tile)
{
    foreach (var item in _grid.Children.Cast<Canvas>()
                .Where(w => w.Children.Any()))
    {
        var canvas = item.Children
        .Cast<Canvas>()
        .FirstOrDefault(w => w.Tag as Tile == tile);
        if (canvas != null)
            canvas.Children.Clear();
    }
}

public async void New(Grid grid)
{
    _gameOver = false;
    _board = new Board();
    _board.Removed += (Tile tile) =>
        Remove(tile);
    await SetSourcesAsync();
    Layout(grid);
    _dialog = new Dialog(grid.XamlRoot, title);
}

public void Hint() =>
    _board.SetHint();
```

Layout will create the look-and-feel of the game by setting up all the elements, **Remove** will be set to use the **Event Handler**, **New** will setup and start a new game and assign the **Event Handler** and **Hint** will be used to display the hint for the player.

Step 20

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



Step 21

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 22

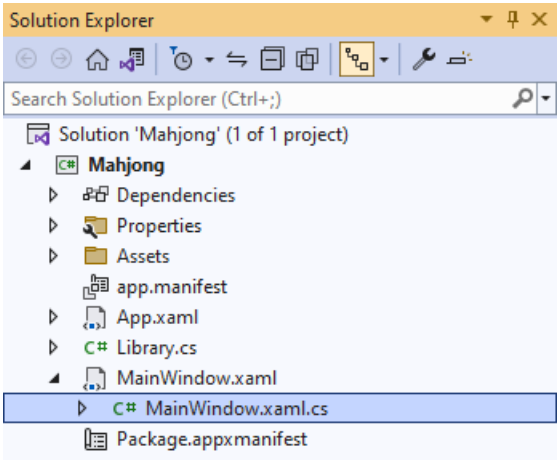
While still in the **XAML** for **MainWindow.xaml** above **`</Window>`**, type in the following **XAML**:

```
<Grid>
    <Viewbox>
        <Grid Margin="50" Name="Display"
        HorizontalAlignment="Center"
        VerticalAlignment="Center" Loaded="New">
            <ProgressRing/>
        </Grid>
    </Viewbox>
    <CommandBar VerticalAlignment="Bottom">
        <AppBarButton Icon="Page2" Label="New" Click="New"/>
        <AppBarButton Icon="Help" Label="Hint" Click="Hint"/>
    </CommandBar>
</Grid>
```

This **XAML** contains a **`Grid`** with a **`Viewbox`** which will **Scale** a **`Grid`** which contains a **`ProgressRing`** which will display until all assets have been set and it has a **`Loaded`** event handler for **`New`** which is also shared by an **`AppBarButton`** along with another for **`Hint`**.

Step 23

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 24

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 25

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);

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

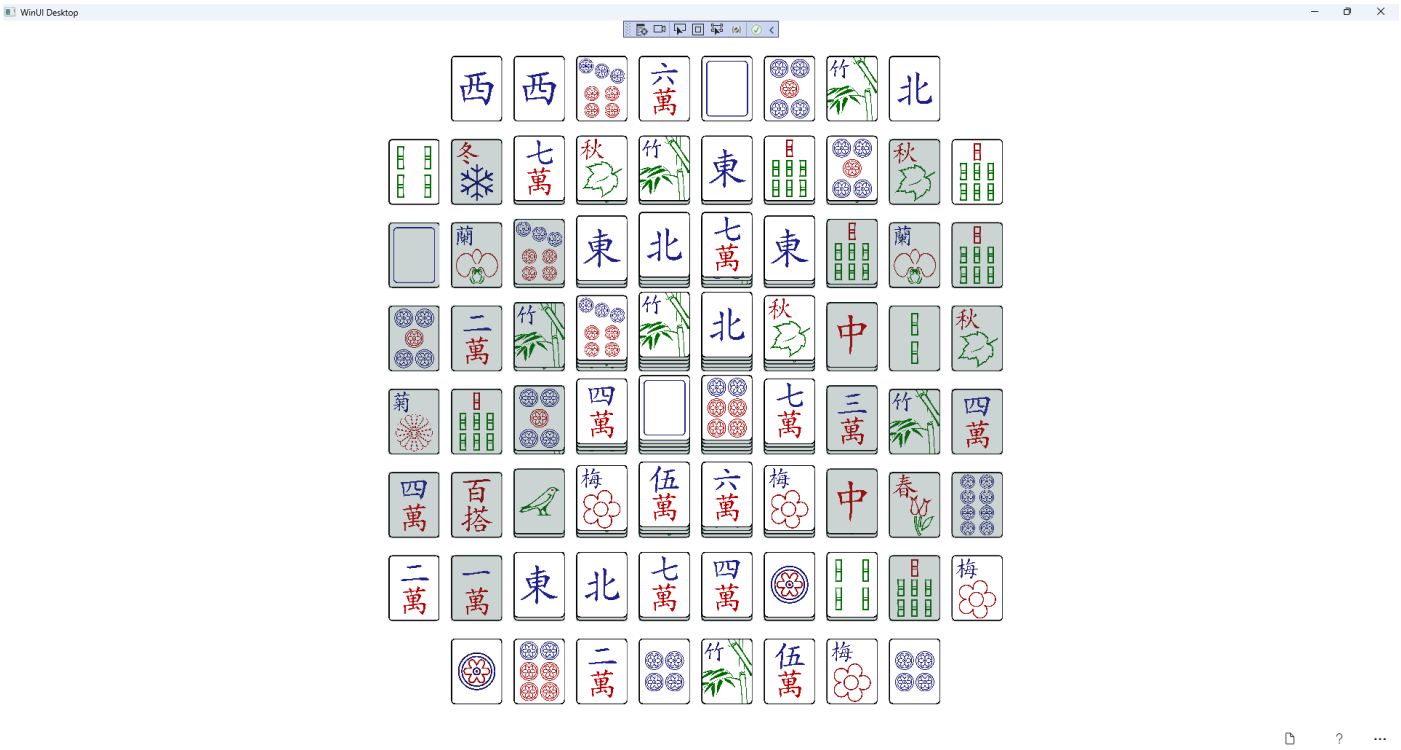
Here an **Instance** of the **Class** of **`Library`** is created then below this is the **Method** of **`New`** and **`Hint`** that will be used with **Event Handler** from the **XAML**, this **Method** uses Arrow Syntax with the **`=>`** for an Expression Body which is useful when a **Method** only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

Step 26

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

Step 27

Once running you can then tap on a **Tile** and then select another **Tile** that matches to remove it from the **Board** – if you're not sure which two to pick then use *Hint* to indicate which to choose, if no more can be moved then you'll get the option to **Shuffle** them, and you win when all the tiles have been removed or select *New* to start a new game.



Step 28

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!