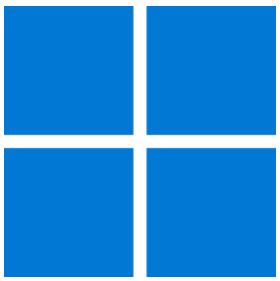
tutorial

Windows App SDK



Tetrominos

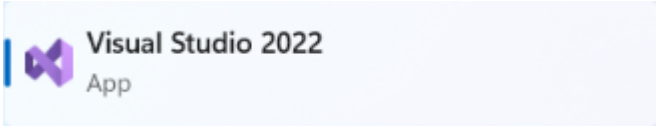
Tetrominos

Tetrominos shows how you can create the game of **Tetrominos** or **Tetris** based on the work by OttoBotCode using a control in a toolkit from **NuGet** using the **Windows App SDK**.

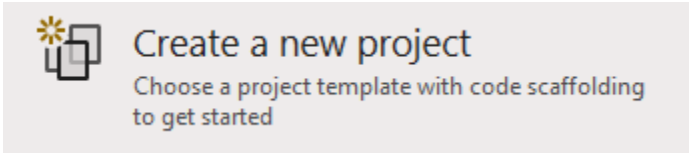
Step 1

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

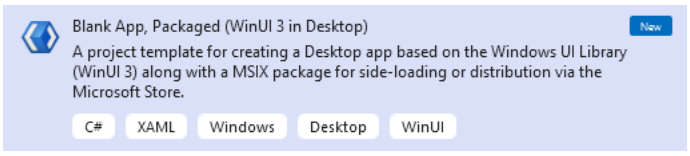
In **Windows 11** choose **Start** and then find or search for **Visual Studio 2022** and then select it.



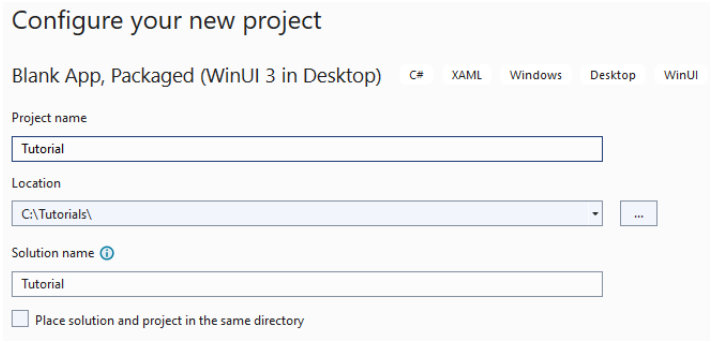
Once **Visual Studio 2022** has started select **Create a new project**.



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

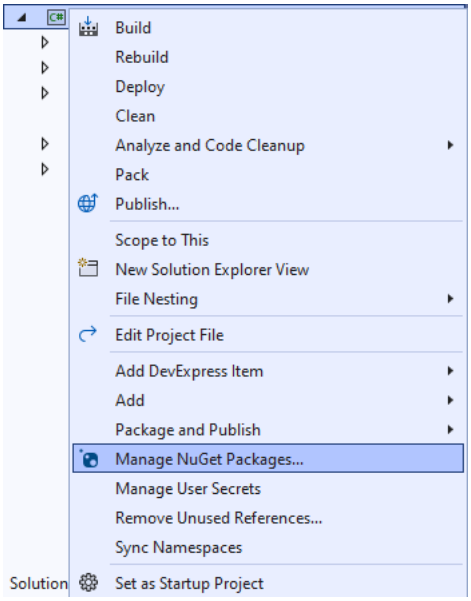


After that in **Configure your new project** type in the **Project name** as *Tetronimos*, then select a Location and then select **Create** to start a new **Solution**.



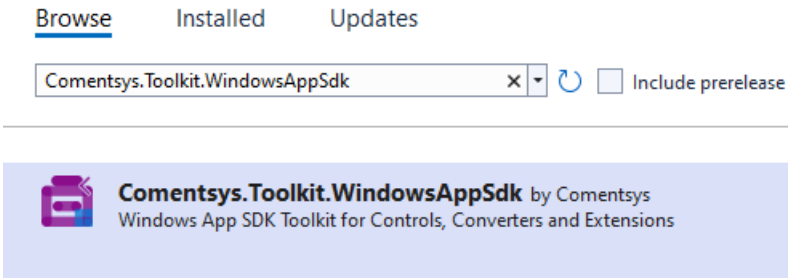
Step 2

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

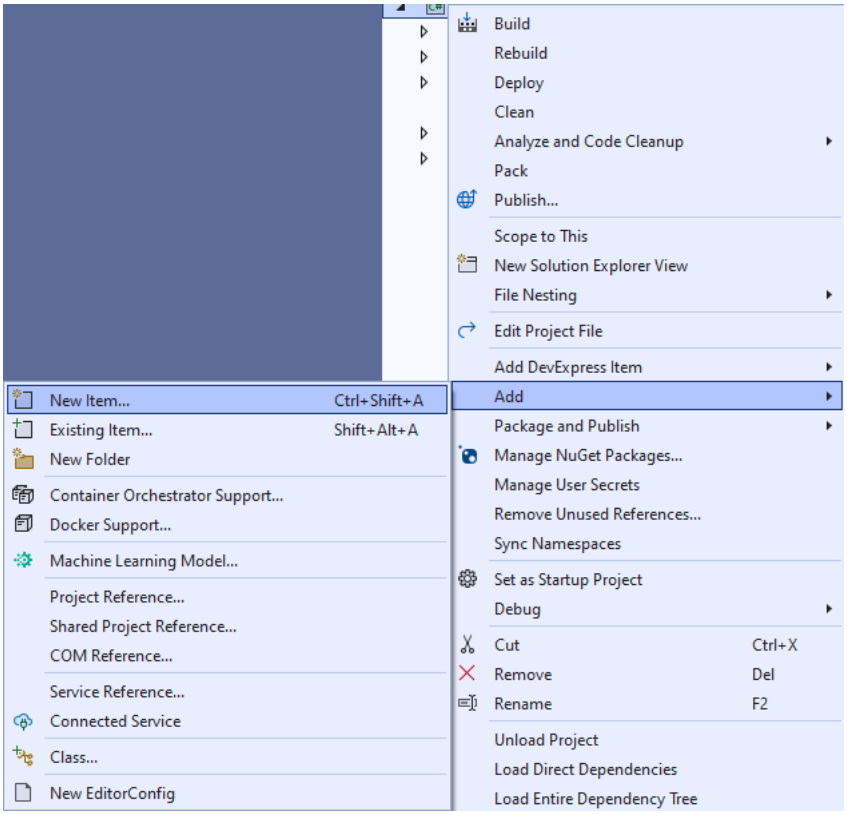
Then in the **NuGet Package Manager** from the **Browse** tab search for **Comentsys.Toolkit.WindowsAppSdk** and then select **Comentsys.Toolkit.WindowsAppSdk by Comentsys** as indicated and select **Install**



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

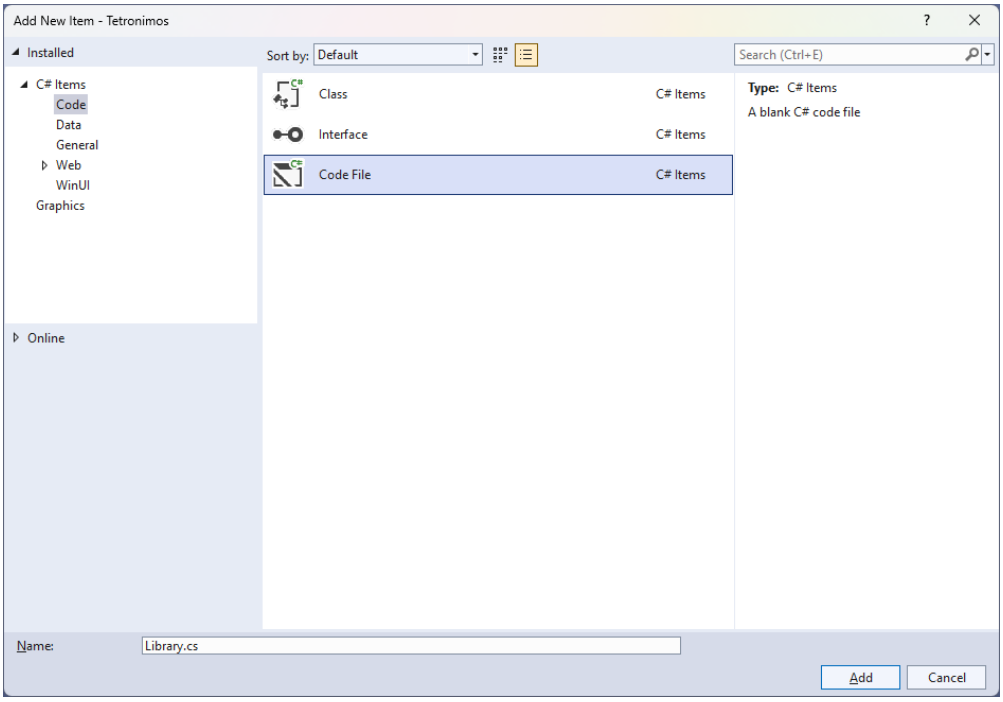
Step 4

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* 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 Statements

namespace Tetrominos;

// Position Class

// Block Class

// IBlock, JBlock & LBlock Class

// OBlock, SBlock, TBlock & ZBlock Class

// Queue Class

public class Board
{
    // Board Member, Properties & Constructor

    // Board ClearRow, MoveRowDown, IsInside & IsOutside Methods

    // Board IsRowFull, IsRowEmpty & ClearFullRow Methods

}

public class State
{
    // State Members & Properties

    // State Private Methods

    // State Constructor and Hold, RotateClockwise & RotateAntiClockwise Methods

    // State Left, Right, Down, Distance & Drop Methods

}

public class Library
{
    // Constants, Variables & Enum

    // SetPieces & Board Methods

    // Block, Preview, Next & Held Methods

    // Ghost, Score, Over & Draw Methods

    // Loop & Move Methods

    // Layout & New Methods

}
```

Step 7

Still in *Library.cs* for the **`namespace`** of **`Tetrominos`** in *Library.cs* you can define some **`using`** statements needed for the **`class`** by typing below the **Comment** of **`// Using Statements`** the following:

```
using Comentsys.Toolkit.WindowsAppSdk;
using Microsoft.UI;
using Microsoft.UI.Xaml;
using Microsoft.UI.Xaml.Controls;
using Microsoft.UI.Xaml.Media;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
```

Step 8

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

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

    public int Column { get; set; }

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

`Position` has **Properties** for **`Row`** and **`Column`** along with a **Constructor** to set them which will represent the location of an element for the game.

Step 9

Still in *Library.cs* for the **`namespace`** of **`Tetrominos`** in *Library.cs* you can define a **`class`** for **`Block`** after the **Comment** of **`// Block Class`** by typing the following:

```
public abstract class Block
{
    private readonly Position[][] _tiles;
    private readonly Position _start;
    private readonly Position _offset;
    private int _rotate;

    public int Id { get; }

    public Block(int id, int row, int column, Position[][] tiles)
    {
        Id = id;
        _start = new Position(row, column);
        _offset = new Position(_start.Row, _start.Column);
        _tiles = tiles;
    }

    public IEnumerable<Position> Positions =>
        _tiles[_rotate].Select(position =>
        new Position(
            position.Row + _offset.Row,
            position.Column + _offset.Column));

    public void RotateClockwise() =>
        _rotate = (_rotate + 1) % _tiles.Length;

    public void RotateAntiClockwise() =>
        _rotate = _rotate == 0 ?
            _rotate = _tiles.Length - 1 : _rotate--;

    public void Move(int rows, int columns)
    {
        _offset.Row += rows;
        _offset.Column += columns;
    }

    public void Reset()
    {
        _rotate = 0;
        _offset.Row = _start.Row;
        _offset.Column = _start.Column;
    }

    public IEnumerable<Position> Preview =>
        _tiles[0];
}
```

`Block` uses **`Position`** to represent the set of tiles for the elements or **Block** for the game including **Methods** such as **`RotateClockwise`** and **`RotateAntiClockwise`** to update the tiles accordingly and to get a preview to be used to represent an upcoming **Block**.

Step 10

Still in the **namespace** of **Tetrominos** in *Library.cs* after the **Comment** of **// IBlock, JBlock & LBlock Class** type the following:

```
public class IBlock : Block
{
    public IBlock() : base(1, -1, 3, new Position[][]
    {
        new Position[] { new(1,0), new(1,1), new(1,2), new(1,3) },
        new Position[] { new(0,2), new(1,2), new(2,2), new(3,2) },
        new Position[] { new(2,0), new(2,1), new(2,2), new(2,3) },
        new Position[] { new(0,1), new(1,1), new(2,1), new(3,1) }
    })
    { }
}

public class JBlock : Block
{
    public JBlock() : base(2, 0, 3, new Position[][]
    {
        new Position[] { new(0, 0), new(1, 0), new(1, 1), new(1, 2) },
        new Position[] { new(0, 1), new(0, 2), new(1, 1), new(2, 1) },
        new Position[] { new(1, 0), new(1, 1), new(1, 2), new(2, 2) },
        new Position[] { new(0, 1), new(1, 1), new(2, 1), new(2, 0) }
    })
    { }
}

public class LBlock : Block
{
    public LBlock() : base(3, 0, 3, new Position[][]
    {
        new Position[] { new(0,2), new(1,0), new(1,1), new(1,2) },
        new Position[] { new(0,1), new(1,1), new(2,1), new(2,2) },
        new Position[] { new(1,0), new(1,1), new(1,2), new(2,0) },
        new Position[] { new(0,0), new(0,1), new(1,1), new(2,1) }
    })
    { }
}
```

IBlock, **JBlock** and **LBlock** represent some of the **Blocks** in the game which are shaped like an *I*, *J* and *L*

Step 11

Still in the **namespace** of **Tetrominos** in *Library.cs* after the **Comment** of **// OBlock, SBlock, TBlock & ZBlock Class** type the following:

```
public class OBlock : Block
{
    public OBlock() : base(4, 0, 4, new Position[][]
    {
        new Position[] { new(0,0), new(0,1), new(1,0), new(1,1) }
    })
    { }
}

public class SBlock : Block
{
    public SBlock() : base(5, 0, 3, new Position[][]
    {
        new Position[] { new(0,1), new(0,2), new(1,0), new(1,1) },
        new Position[] { new(0,1), new(1,1), new(1,2), new(2,2) },
        new Position[] { new(1,1), new(1,2), new(2,0), new(2,1) },
        new Position[] { new(0,0), new(1,0), new(1,1), new(2,1) }
    })
    { }
}

public class TBlock : Block
{
    public TBlock() : base(6, 0, 3, new Position[][]
    {
        new Position[] { new(0,1), new(1,0), new(1,1), new(1,2) },
        new Position[] { new(0,1), new(1,1), new(1,2), new(2,1) },
        new Position[] { new(1,0), new(1,1), new(1,2), new(2,1) },
        new Position[] { new(0,1), new(1,0), new(1,1), new(2,1) }
    })
    { }
}

public class ZBlock : Block
{
    public ZBlock() : base(7, 0, 3, new Position[][]
    {
        new Position[] { new(0,0), new(0,1), new(1,1), new(1,2) },
        new Position[] { new(0,2), new(1,1), new(1,2), new(2,1) },
        new Position[] { new(1,0), new(1,1), new(2,1), new(2,2) },
        new Position[] { new(0,1), new(1,0), new(1,1), new(2,0) }
    })
    { }
}
```

OBlock, **SBlock**, **TBlock** and **ZBlock** represent some of the **Blocks** in the game which are shaped like an *O*, *S*,*T* and *Z*.

Step 12

While still in the **`namespace`** of **`Tetrominos`** in *Library.cs* after the **Comment** of **`// Queue Class`** type the following:

```
public class Queue
{
    private readonly Block[] _blocks = new Block[]
    {
        new IBlock(),
        new JBlock(),
        new LBlock(),
        new OBlock(),
        new SBlock(),
        new TBlock(),
        new ZBlock()
    };

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

    private Block Choose() =>
        _blocks[_random.Next(0, _blocks.Length)];

    public Block Next { get; private set; }

    public Queue() =>
        Next = Choose();

    public Block Get()
    {
        var block = Next;
        do
        {
            Next = Choose();
        } while(block.Id == Next.Id);
        return block;
    }
}
```

`Queue` represents the possible **Blocks** that will be returned from the set of **Blocks** that have been defined.

Step 13

While still in the **namespace** of **Tetrominos** in *Library.cs* in the **Class** of **Board** after the **Comment** of **// Board Member, Properties & Constructor** type the following **Member, Properties** and **Constructor**:

```
private readonly int[,] _board;

public int Rows { get; }

public int Columns { get; }

public int this[int row, int column]
{
    get => _board[row, column];
    set => _board[row, column] = value;
}

public Board(int rows, int columns) =>
    (Rows, Columns, _board) = (rows, columns, new int[rows, columns]);
```

The **Member** of **_board** will represent the elements for the layout of the game and the **Properties** will define the **Rows** and **Columns** for this with the ability to assign these using the **this** along with the **Constructor** and the **_board** will be updated with the **Methods** that will be defined in the next **Steps**.

Step 14

While still in the **namespace** of **Tetrominos** in *Library.cs* in the **Class** of **Board** after the **Comment** of **// Board ClearRow, MoveRowDown, IsInside & IsOutside Methods** type the following **Methods**:

```
private void ClearRow(int row)
{
    for (int column = 0; column < Columns; column++)
    {
        _board[row, column] = 0;
    }
}

private void MoveRowDown(int row, int rows)
{
    for (int column = 0; column < Columns; column++)
    {
        _board[row + rows, column] = _board[row, column];
        _board[row, column] = 0;
    }
}

public bool IsInside(int row, int column) =>
    row >= 0 && row < Rows && column >= 0 && column < Columns;

public bool IsEmpty(int row, int column) =>
    IsInside(row, column) && _board[row, column] == 0;
```

Step 15

While still in the **namespace** of **Tetrominos** in *Library.cs* in the **Class** of **Board** after the **Comment** of **// Board IsRowFull, IsRowEmpty & ClearFullRow Methods** type the following **Methods**:

```
public bool IsRowFull(int row)
{
    for(int column = 0; column < Columns; column++)
    {
        if(_board[row, column] == 0)
            return false;
    }
    return true;
}

public bool IsRowEmpty(int row)
{
    for (int column = 0; column < Columns; column++)
    {
        if (_board[row, column] != 0)
            return false;
    }
    return true;
}

public int ClearFullRows()
{
    var cleared = 0;
    for(int row = Rows - 1; row >= 0; row--)
    {
        if(IsRowFull(row))
        {
            ClearRow(row);
            cleared++;
        }
        else if(cleared > 0)
            MoveRowDown(row, cleared);
    }
    return cleared;
}
```

Step 16

While still in the **`namespace`** of **`Tetrominos`** in *Library.cs* in the **`Class`** of **`State`** after the **Comment** of **`// State Members & Properties`** type the following **Members** and **Properties**:

```
private readonly Board _board;
private Block _current;

public Board Board => _board;

public Queue Queue { get; }

public bool Over { get; private set; }

public int Score { get; private set; }

public Block Held { get; private set; }

public bool CanHold { get; private set; }

public Block Current
{
    get => _current;
    private set => Update(value);
}
```

Step 17

While still in the **`namespace`** of **`Tetrominos`** in *Library.cs* in the **`Class`** of **`State`** after the **Comment** of **`// Private Methods`** type the following **Methods**:

```
private bool Fits()
{
    foreach(var position in _current.Positions)
        if(!_board.IsEmpty(position.Row, position.Column))
            return false;
    return true;
}

private void Update(Block value)
{
    _current = value;
    _current.Reset();
    for(int i = 0; i < 2; i++)
    {
        _current.Move(1, 0);
        if(!Fits())
            _current.Move(-1, 0);
    }
}

private bool IsOver() =>
    !(_board.IsRowEmpty(0) && _board.IsRowEmpty(1));

private void Place()
{
    foreach (var position in Current.Positions)
        _board[position.Row, position.Column] = Current.Id;
    Score += _board.ClearFullRows();
    if (IsOver())
        Over = true;
    else
    {
        Current = Queue.Get();
        CanHold = true;
    }
}

private int GetDistance(Position position)
{
    var drop = 0;
    while(_board.IsEmpty(position.Row + drop + 1, position.Column))
        drop++;
    return drop;
}
```

`State` represents the conditions of the game including being able to place **Blocks** and determine if the game is over.

Step 18

While still in the **namespace** of **Tetrominos** in *Library.cs* in the **Class** of **State** after the **Comment** of **// State Constructor and Hold, RotateClockwise & RotateAntiClockwise Methods** type the following **Constructor** and **Methods**:

```
public State()
{
    _board = new Board(22, 10);
    Queue = new Queue();
    Current = Queue.Get();
    CanHold = true;
}

public void Hold()
{
    if(!CanHold)
        return;
    if(Held == null)
    {
        Held = Current;
        Current = Queue.Get();
    }
    else
        (Held, Current) = (Current, Held);
    CanHold = false;
}

public void RotateClockwise()
{
    Current.RotateClockwise();
    if(!Fits())
        Current.RotateAntiClockwise();
}

public void RotateAntiClockwise()
{
    Current.RotateAntiClockwise();
    if (!Fits())
        Current.RotateClockwise();
}
```

Hold will hold a **Block** for the game and **RotateClockwise** and **RotateAntiClockwise** will rotate a **Block**.

Step 19

While still in the **namespace** of **Tetrominos** in *Library.cs* in the **Class** of **State** after the **Comment** of **// State Left, Right, Down, Distance & Drop Methods** type the following **Methods**:

```
public void Left()
{
    Current.Move(0, -1);
    if(!Fits())
        Current.Move(0, 1);
}

public void Right()
{
    Current.Move(0, 1);
    if (!Fits())
        Current.Move(0, -1);
}

public void Down()
{
    Current.Move(1, 0);
    if(!Fits())
    {
        Current.Move(-1, 0);
        Place();
    }
}

public int Distance()
{
    var drop = _board.Rows;
    foreach(var position in Current.Positions)
    {
        drop = Math.Min(drop, GetDistance(position));
    }
    return drop;
}

public void Drop()
{
    Current.Move(Distance(), 0);
    Place();
}
```

Left, **Right** and **Down** will move a **Block** accordingly and **Distance** will determine the minimum position on the **Board** and **Drop** will place a **Block** there.

Step 20

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

```
private const int max_delay = 1000;
private const int min_delay = 75;
private const int increase = 25;
private const int preview = 4;
private const int size = 36;
private readonly Brush[] _brushes = new Brush[]
{
    new SolidColorBrush(Colors.White),
    new SolidColorBrush(Colors.Cyan),
    new SolidColorBrush(Colors.Blue),
    new SolidColorBrush(Colors.Orange),
    new SolidColorBrush(Colors.Gold),
    new SolidColorBrush(Colors.Green),
    new SolidColorBrush(Colors.Purple),
    new SolidColorBrush(Colors.Red)
};
private Grid _grid;
private Grid _next;
private Grid _hold;
private TextBlock _text;
private Piece[,] _pieces;
private State _state;

public enum Moves
{
    Left,
    Right,
    Down,
    RotateClockwise,
    RotateAntiClockwise,
    Hold,
    Drop
}
```

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 an **`enum`** which is used to define moves possible in the game.

Step 21

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

```
private static Piece[,] SetPieces(Grid grid, int rows, int columns)
{
    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 piece = new Piece
            {
                Width = size,
                Height = size,
                IsSquare = true,
                Stroke = new SolidColorBrush(Colors.Black)
            };
            Grid.SetColumn(piece, column);
            Grid.SetRow(piece, row);
            grid.Children.Add(piece);
            pieces[row, column] = piece;
        }
    }
    return pieces;
}

private void Board(Board board)
{
    for (int row = 0; row < board.Rows; row++)
    {
        for (int column = 0; column < board.Columns; column++)
        {
            var id = board[row, column];
            var counter = _pieces[row, column];
            counter.Opacity = 1;
            counter.Fill = _brushes[id];
        }
    }
}
```

`SetPieces` is used to define where the elements that make up the visuals for a **Block** using **`Piece`** is performed and **`Board`** is used to set up the **Board**.

Step 22

While still in the **`namespace`** of **`Tetrominos`** in *Library.cs* and in the **`class`** of **`Library`** after the **Comment** of **`// Block, Preview, Next & Held Methods`** type in the following **Methods**:

```
private void Block(Block block)
{
    foreach(var position in block.Positions)
    {
        var counter = _pieces[position.Row, position.Column];
        counter.Opacity = 1;
        counter.Fill = _brushes[block.Id];
    }
}

private void Preview(Grid grid, Block block = null)
{
    foreach (var piece in grid.Children.Cast<Piece>())
    {
        piece.Fill = _brushes[0];
    }
    if (block != null)
    {
        foreach (var position in block.Preview)
        {
            var piece = grid.Children.Cast<Piece>()
                .First(f => Grid.GetRow(f) == position.Row
                    && Grid.GetColumn(f) == position.Column);
            piece.Fill = _brushes[block.Id];
        }
    }
}

private void Next(Queue queue) =>
    Preview(_next, queue.Next);

private void Held(Block block)
{
    if (block == null)
        Preview(_hold);
    else
        Preview(_hold, block);
}
```

`Block` is used to set elements of the **Board** for a **Block** and **`Preview`** is used to get the elements to display what a **Block** looks like. **`Next`** will use **`Preview`** to display the upcoming **Block** and **`Held`** will use **`Preview`** to display the **Block** being **Held**.

Step 23

While still in the **`namespace`** of **`Tetrominos`** in *Library.cs* and in the **`class`** of **`Library`** after the **Comment** of **`// Ghost, Score, Over & Draw Methods`** type in the following **Methods**:

```
private void Ghost(Block block)
{
    int distance = _state.Distance();
    foreach (var position in block.Positions)
    {
        var counter = _pieces[position.Row + distance, position.Column];
        counter.Opacity = 0.25;
        counter.Fill = _brushes[block.Id];
    }
}

private void Score(int score) =>
    _text.Text = $"Score {score}";

private void Over(int score) =>
    _text.Text = $"Game Over! Final Score {score}";

private void Draw(State state)
{
    Board(state.Board);
    Ghost(state.Current);
    Block(state.Current);
    Next(state.Queue);
    Held(state.Held);
    Score(state.Score);
}
```

`Ghost` is used to show a semi-transparent version of the **Block** where it would end up if placed, **`Score`** and **`Over`** will be used to set the output to be displayed accordingly. **`Draw`** will be used to call the **Methods** which will be used as part of the game-loop.

Step 24

While still in the **namespace** of **Tetrominos** in *Library.cs* and in the **class** of **Library** after the **Comment** of **// Loop & Move Methods** type in the following **Methods**:

```
private async Task Loop()
{
    Draw(_state);
    while(!_state.Over)
    {
        var delay = Math.Max(min_delay, max_delay - (_state.Score * increase));
        await Task.Delay(delay);
        _state.Down();
        Draw(_state);
    }
    Over(_state.Score);
}

public void Move(string value)
{
    var move = Enum.Parse(typeof(Moves), value);
    if (_state.Over)
        return;
    switch (move)
    {
        case Moves.Left:
            _state.Left();
            break;
        case Moves.Right:
            _state.Right();
            break;
        case Moves.Down:
            _state.Down();
            break;
        case Moves.RotateClockwise:
            _state.RotateClockwise();
            break;
        case Moves.RotateAntiClockwise:
            _state.RotateAntiClockwise();
            break;
        case Moves.Hold:
            _state.Hold();
            break;
        case Moves.Drop:
            _state.Drop();
            break;
        default:
            return;
    }
    Draw(_state);
}
```

Loop is used to create the game-loop and will also increase the speed of the game as it progresses to make the game harder, and **Move** will perform the actions based on the value that match the **enum**.

Step 25

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

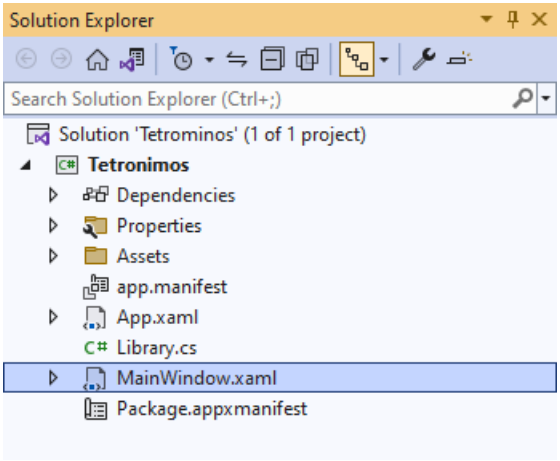
```
private void Layout(Grid grid)
{
    grid.Children.Clear();
    grid.RowDefinitions.Clear();
    grid.ColumnDefinitions.Clear();
    grid.ColumnDefinitions.Add(new ColumnDefinition()
    {
        Width = GridLength.Auto
    });
    grid.ColumnDefinitions.Add(new ColumnDefinition()
    {
        Width = new GridLength(1, GridUnitType.Star)
    });
    StackPanel panel = new()
    {
        Orientation = Orientation.Vertical,
    };
    _text = new TextBlock();
    panel.Children.Add(_text);
    panel.Children.Add(new TextBlock() { Text = "Next" });
    _next = new Grid();
    SetPieces(_next, preview, preview);
    panel.Children.Add(_next);
    panel.Children.Add(new TextBlock() { Text = "Hold" });
    _hold = new Grid();
    SetPieces(_hold, preview, preview);
    panel.Children.Add(_hold);
    grid.Children.Add(panel);
    _grid = new Grid();
    grid.Children.Add(_grid);
    _pieces = SetPieces(_grid, _state.Board.Rows, _state.Board.Columns);
    Grid.SetColumn(_grid, 1);
}

public async void New(Grid grid)
{
    _grid = grid;
    _state = new State();
    Layout(grid);
    await Loop();
}
```

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

Step 26

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



Step 27

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 28

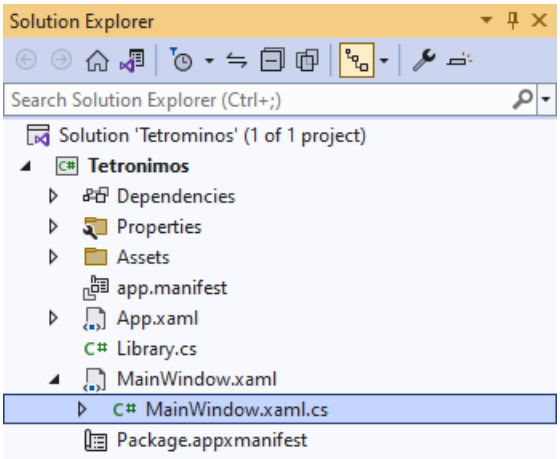
While still in the **XAML** for **MainWindow.xaml** above **`</Window>`**, type in the following **XAML**:

```
<Grid>
    <Viewbox>
        <Grid Margin="50" Name="Display"
        HorizontalAlignment="Center"
        VerticalAlignment="Center" Loaded="New"/>
    </Viewbox>
    <CommandBar VerticalAlignment="Bottom">
        <AppBarButton Icon="Back" Label="Left"
        Tag="Left" Click="Move"/>
        <AppBarButton Icon="Forward" Label="Right"
        Tag="Right" Click="Move"/>
        <AppBarButton Icon="Download" Label="Down"
        Tag="Down" Click="Move"/>
        <AppBarButton Icon="Priority" Label="Drop"
        Tag="Drop" Click="Move"/>
        <AppBarButton Icon="Redo" Label="Rotate Clockwise"
        Tag="RotateClockwise" Click="Move"/>
        <AppBarButton Icon="Undo" Label="Rotate Clockwise"
        Tag="RotateAntiClockwise" Click="Move"/>
        <AppBarButton Icon="Stop" Label="Hold"
        Tag="Hold" Click="Move"/>
        <AppBarButton Icon="Page2" Label="New" Click="New"/>
    </CommandBar>
</Grid>
```

This **XAML** contains a **`Grid`** to contain the game itself along with a **`CommandBar`** which will be used to perform the moves in the game.

Step 29

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 30

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 31

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 Move(object sender, RoutedEventArgs e) =>
    _library.Move(((AppBarButton)sender).Tag as string);
```

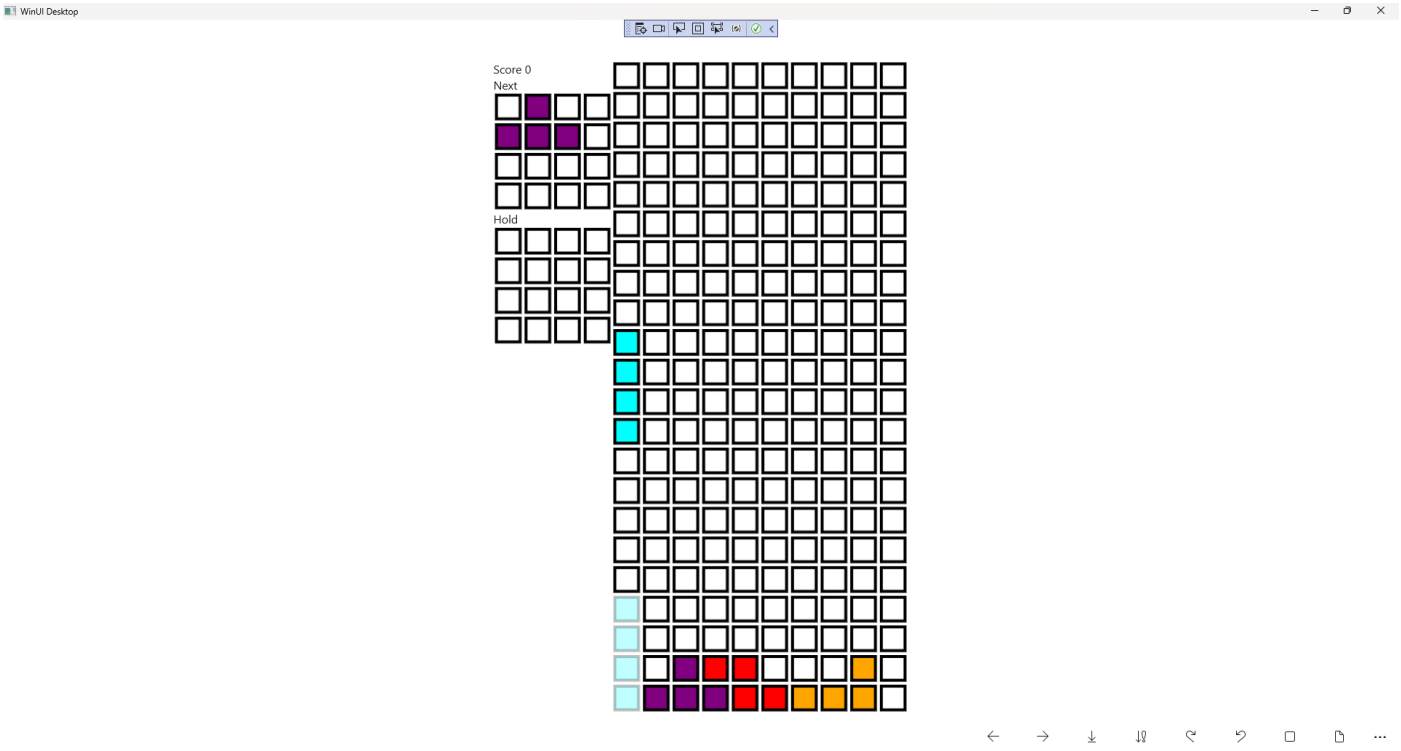
Here an **Instance** of the **Class** of **`Library`** is created then below this is the **Method** of **`New`** and **`Move`** that will be used with **Event Handler** from the **XAML**, this **Method** uses Arrow Syntax with the **`=>`** for an Expression Body which is useful when a **Method** only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

Step 32

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

Step 33

Once running you can tap on the appropriate button on the **`CommandBar`** to play the game where the goal is to get as many lines as possible, when you do that line will disappear but if you reach the top then the game is over, you can also **Hold** a **Block** or see what one is **Next** or select *New* to start a new game.



Step 34

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!