**Cards Game** shows how to create the look-and-feel of some **Playing Cards** and to see if it’s possible to match a pair based on their face value in the game also known as **snap**

## Step 1

|  |  |
| --- | --- |
|  | Follow **Setup and Start** on how to Install and/or Get Started with **Visual Studio 2019** if not already or in **Windows 10** choose **Start**, find and select **Visual Studio 2019** then from the **Get started** screen select **Create a new project** |
| A screenshot of a cell phone  Description automatically generated | Then choose **Blank App (Universal Windows)** and select **Next** and then in **Configure your new project** enter the **Project name** as **CardsGame** and select **Create** |
| A screenshot of a social media post  Description automatically generated | Finally, in **New Universal Windows Platform Project** pick the **Target version** and **Minimum version** to be at least **Windows 10, version 1903 (10.0; Build 18362)** and then select **OK** |

**Target Version** will control the most recent features of **Windows 10** your application can use. To make sure you always have the most recent version, check for any **Notifications** or **Updates** in **Visual Studio 2019**

## Step 2

|  |  |
| --- | --- |
| A screenshot of a cell phone  Description automatically generated | Choose **Project** then **Add New Item...** from the **Menu** in **Visual Studio 2019** |

## Step 3

|  |  |
| --- | --- |
| A close up of a logo  Description automatically generated | Then choose **Code File** from **Add New Item** in **Visual Studio 2019**, enter the **Name** as **Library.cs** and select **Add** |

## Step 4

In the **Code** View of **Library.cs** will be displayed and in this the following should be entered:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using Windows.UI;  using Windows.UI.Popups;  using Windows.UI.Xaml;  using Windows.UI.Xaml.Controls;  using Windows.UI.Xaml.Media;  public class Library  {  private const string title = "Cards Game";  private const string one = "one";  private const string two = "two";  private const int clubs = 127184;  private const int diamonds = 127168;  private const int hearts = 127152;  private const int spades = 127136;  private const int maximum = 52;  private const int amount = 13;  private readonly Dictionary<int, int> \_suits =  new Dictionary<int, int>()  {  { 1, clubs },  { 14, diamonds },  { 28, hearts },  { 40, spades },  };  private readonly List<int> \_faces = new List<int>()  {  14, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14  };  private int \_first, \_second;  private int \_score, \_counter;  private int \_cardOne, \_cardTwo;  private List<int> \_deckOne = new List<int>();  private List<int> \_deckTwo = new List<int>();  private Random \_random = new Random((int)DateTime.Now.Ticks);  } |

There are using statements to include necessary functionality. \_suits is a Dictionary<int, int> is a **two-dimensional array** of values that will represent the **suit** of a **card** and \_faces is a List<int> which will represent the **face** of a **card** and Random is used to create the numbers for the **cards**

Then below the **private Random \_random = new Random((int)DateTime.UtcNow.Ticks);** line the following **methods** should be entered:

|  |
| --- |
| public void Show(string content, string title)  {  \_ = new MessageDialog(content, title).ShowAsync();  }  private List<int> Choose(int total)  {  return Enumerable.Range(1, total)  .OrderBy(r => \_random.Next(0, total)).ToList();  } |

Show **method** is used to display a basic MessageDialog and Choose **method** pick a set of random numbers

Below **private void Choose(...) { ... } method** the following **method** should be entered:

|  |
| --- |
| private Viewbox Face(int face, Color fill)  {  TextBlock textblock = new TextBlock()  {  IsColorFontEnabled = true,  Text = char.ConvertFromUtf32(face),  TextLineBounds = TextLineBounds.Tight,  Foreground = new SolidColorBrush(fill),  FontFamily = new FontFamily("Segoe UI Emoji"),  HorizontalTextAlignment = TextAlignment.Center  };  return new Viewbox()  {  Margin = new Thickness(0, 0, 0, 2),  Child = textblock  };  } |

Face **method** is used to create a TextBlock which be used for the front and back of a **card**

After the **private void Face(...) { ... }** **method** the following **method** should be entered:

|  |
| --- |
| private Viewbox Add(int value, Color? fill = null)  {  int index = value % amount;  int suit = \_suits.Where(w => value >= w.Key)  .Select(s => s.Value).LastOrDefault();  int face = spades;  if (suit > 0)  {  fill = (suit == hearts || suit == diamonds) ?  Colors.Red : Colors.Black;  face = suit + \_faces[index];  }  return Face(face, fill.Value);  } |

Add **method** is used create the Face of a **card** and will work out which suit and face it should be

Then after the **private Viewbox Add()...) { ... }** **method** the following **method** should be entered:

|  |
| --- |
| private void Card(Grid grid, string name, int value, Color fill)  {  Grid card = new Grid()  {  Name = name,  CornerRadius = new CornerRadius(5),  Background = new SolidColorBrush(Colors.WhiteSmoke)  };  card.Children.Add(Add(value, fill));  grid.Children.Clear();  grid.Children.Add(card);  } |

Card **method** is used to create the layout of a **card** using a Grid and uses the Add **method**

Next after the **private void Card(...) { ... }** **method** the following **method** should be entered:

|  |
| --- |
| private void Set(Grid grid, string name, int value)  {  Grid card = (Grid)grid.FindName(name);  card.Children.Clear();  card.Children.Add(Add(value));  } |

Set **method** uses Add to create each face of a **card** based on the given value

Then after **private void Set(...) { ... }** **method** the following public **method** should be entered:

|  |
| --- |
| public void New(Grid deckOne, Grid deckTwo)  {  \_score = 0;  \_counter = 0;  \_cardOne = 0;  \_cardTwo = 0;  \_deckOne = Choose(maximum);  \_deckTwo = Choose(maximum);  Card(deckOne, one, 0, Colors.DarkRed);  Card(deckTwo, two, 0, Colors.DarkBlue);  } |

New **method** will select values for each **deck** using Choose and setup layout of each Grid using Card **method**

Finally after **private void New(...) { ...}** **method** the following public **method** should be entered:

|  |
| --- |
| public void Play(Grid deckOne, Grid deckTwo)  {  if (\_deckOne != null && \_deckTwo != null)  {  if (\_cardOne < maximum && \_cardTwo < maximum)  {  \_first = \_deckOne[\_cardOne];  Set(deckOne, one, \_first);  \_cardOne++;  \_second = \_deckTwo[\_cardTwo];  Set(deckTwo, two, \_second);  \_cardTwo++;  // Ignore Suit or Face for Match  if (\_first % amount == \_second % amount)  {  \_score++;  Show("Match!", title);  }  \_counter++;  }  else  {  Show($"Game Over! Matched {\_score} of {\_counter}!", title);  }  }  } |

Play **method** will play the game and will use Set to update a **card** and will then compare the selected ones to see if this is a match or not and if the game is over

## Step 5

|  |  |
| --- | --- |
|  | In the **Solution Explorer** of **Visual Studio 2019** select **MainPage.xaml** |

## Step 6

|  |  |
| --- | --- |
| A screenshot of a cell phone  Description automatically generated | Choose **View** then **Designer** from the **Menu** in **Visual Studio 2019** |

## Step 7

In the **Design** View and **XAML** View of **Visual Studio 2019** will be displayed, and in this between the **Grid** and **/Grid** elements enter the following **XAML**:

|  |
| --- |
| <Viewbox>  <Grid>  <Grid.RowDefinitions>  <RowDefinition Height="\*"/>  <RowDefinition Height="Auto"/>  <RowDefinition Height="\*"/>  </Grid.RowDefinitions>  <Grid.ColumnDefinitions>  <ColumnDefinition Width="\*"/>  <ColumnDefinition Width="Auto"/>  <ColumnDefinition Width="\*"/>  <ColumnDefinition Width="Auto"/>  <ColumnDefinition Width="\*"/>  </Grid.ColumnDefinitions>  <Grid Name="DeckOne" Margin="50" Grid.Column="1" Grid.Row="1"  Height="140" Width="100" Tapped="DeckOne\_Tapped"/>  <Grid Name="DeckTwo" Margin="50" Grid.Column="3" Grid.Row="1"  Height="140" Width="100" Tapped="DeckTwo\_Tapped"/>  </Grid>  </Viewbox>  <CommandBar VerticalAlignment="Bottom">  <AppBarButton Icon="Page2" Label="New" Click="New\_Click"/>  </CommandBar> |

The first block of **XAML** the main user interface features a **Grid** with two **Grid** Controls within to represent the **cards**. The second block of **XAML** is the **CommandBar** which contains **New** to reset the game

## Step 8

|  |  |
| --- | --- |
|  | Choose **View** then **Code** from the **Menu** in **Visual Studio 2019** |

## Step 9

Once in the **Code** View, below the end of **public MainPage() { ... }** the following Code should be entered:

|  |
| --- |
| private Library library = new Library();  private void New\_Click(object sender, RoutedEventArgs e)  {  library.New(DeckOne, DeckTwo);  }  private void DeckOne\_Tapped(object sender, RoutedEventArgs e)  {  library.Play(DeckOne, DeckTwo);  }  private void DeckTwo\_Tapped(object sender, RoutedEventArgs e)  {  library.Play(DeckOne, DeckTwo);  } |

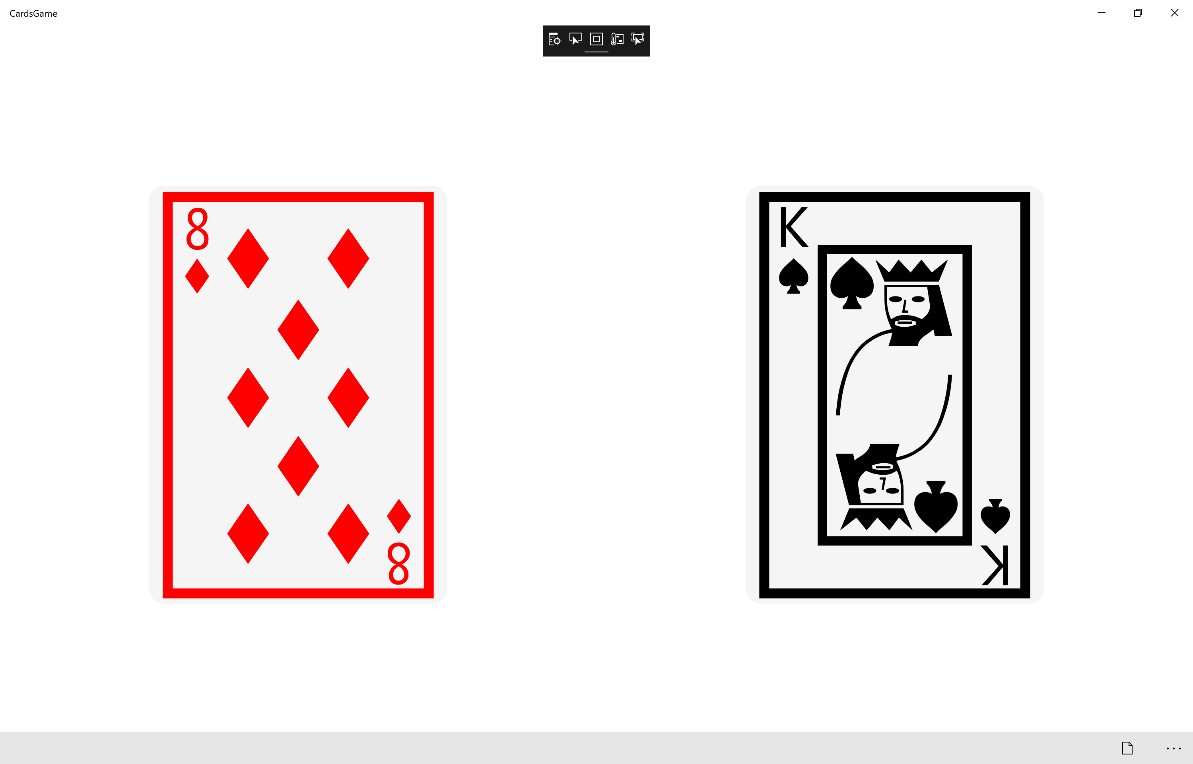
Below the **MainPage** method an instance of the Library **class** is created. In the New\_Click(...) **Event** handler will setup the game with the New **method**, DeckOne\_Tapped(...) and DeckTwo\_Tapped will call the Play **method** in the Library **class**

## Step 10

|  |  |
| --- | --- |
|  | That completes the **Universal Windows Platform** Application, in **Visual Studio 2019** select **Local Machine** to run the Application |

## Step 11

Once the Application is running you can then click the **New**, then Tap on either of the **decks** to show a **card**, match value in both **decks** to score a point, do this until all the **cards** have been displayed



## Step 12

|  |  |
| --- | --- |
| A picture containing object  Description automatically generated | To Exit the Application, select the **Close** button in the top right of the Application |