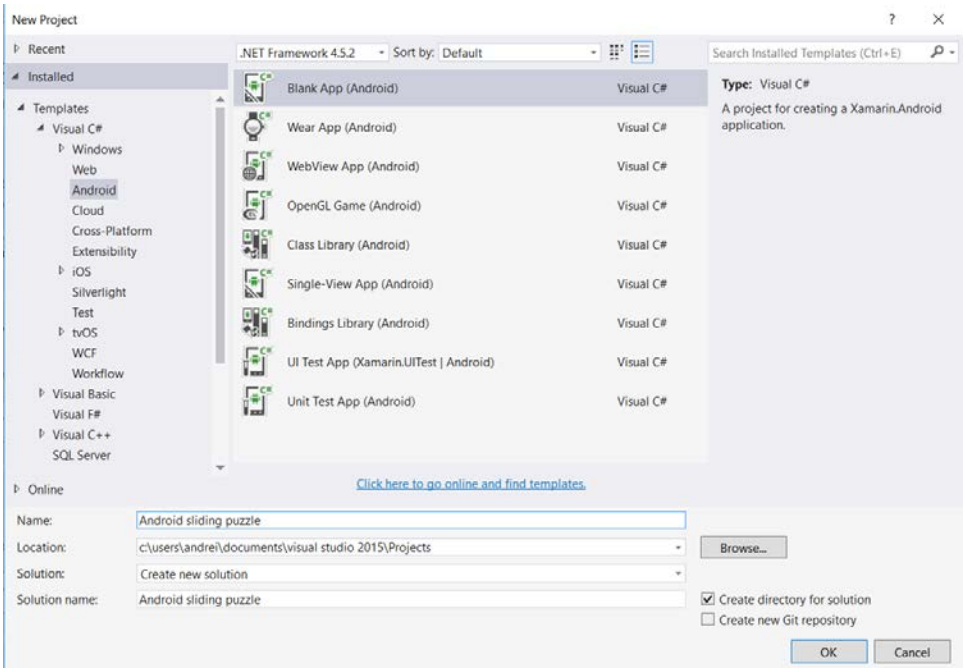
Android sliding puzzle in Xamarin

Following this tutorial you will create your own very first Android application. There are no prerequisited to build a great app and you will create a nice looking sliding puzzle following an easy-to-follow approach. I will explain at every stage what we are trying to achive and how to code each element used.

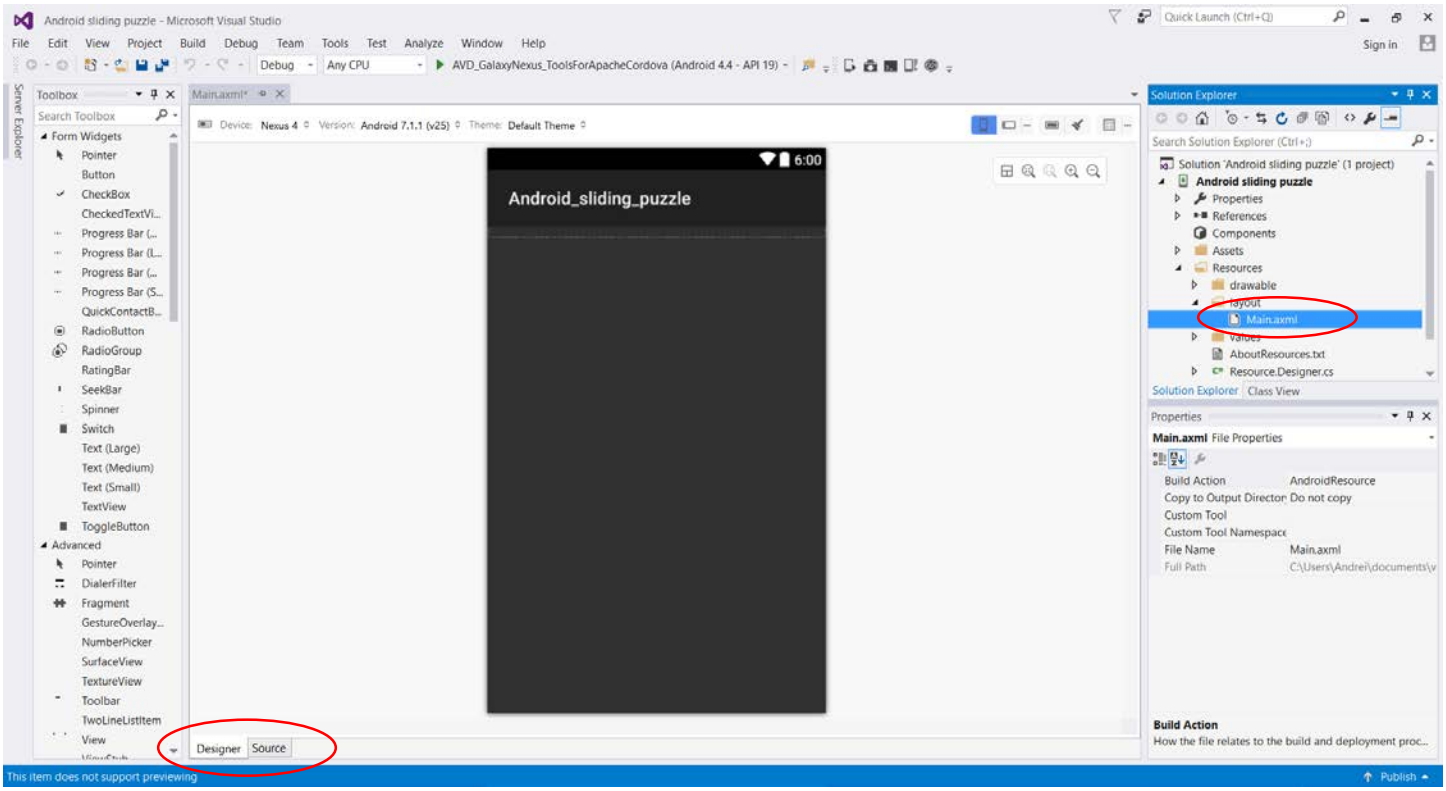
Take a look, this is what you'll create!

This tutorial is a more detailed reproduction of the course Xamarin Android Sliding Puzzle C# (https://www.udemy.com/xamarin-android-sliding-puzzle-csharp) by Amir J. The code is fully rewritten and commented by myself after his guidance. Still, the idea of the application belongs to him.

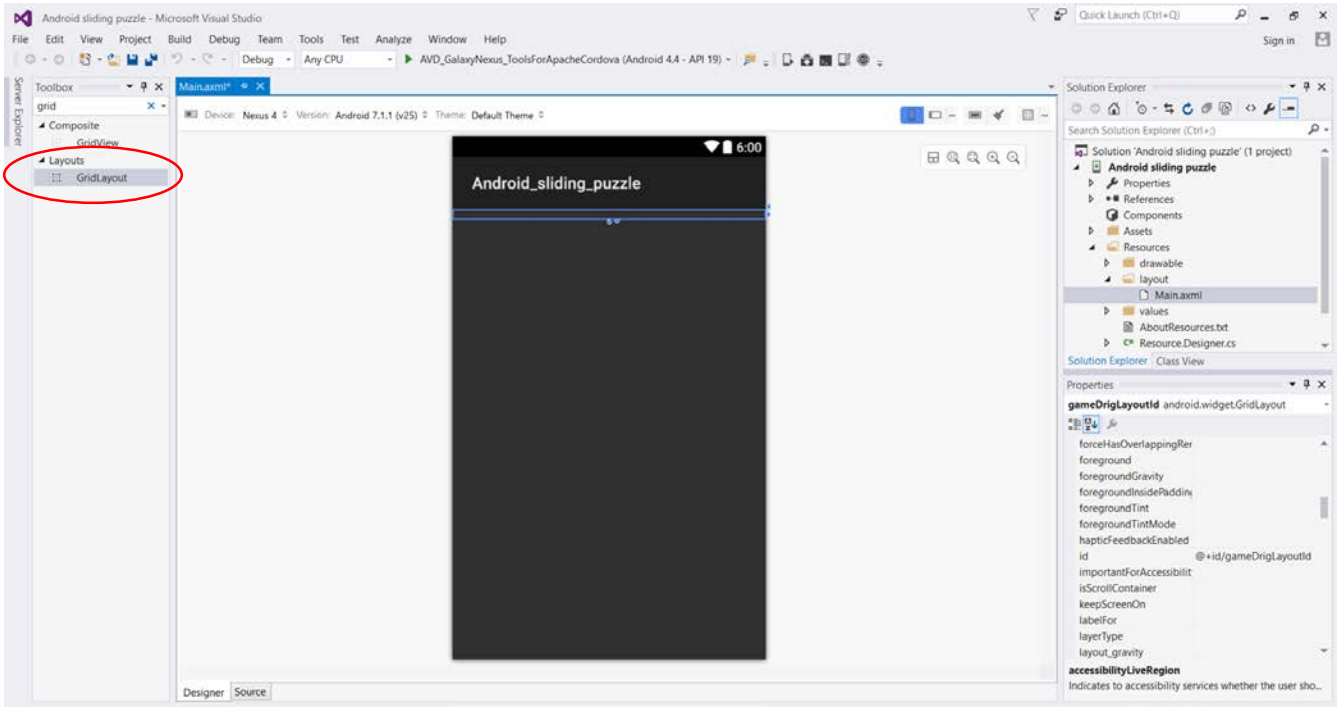
1. Let's create a new Androd project. Click on **File -> New -> Project**. Them in the left menu go to **Templates -> Visual C# -> Android** and select Black App (Android). Give it the name "Android sliding puzzle" and we are good to go.



2. Now we want to create our first button, one that will shuffle the puzzle. In the Solution explorer (right) go to **Android sliding puzzle -> Resources -> layout -> Main.axml**. This file is resposable for the appearance of our application, so here we need to add all the buttons. If you don't see the Android screen, **be sure that you have made all the updates to Xamarin and you are in the Designer tab** (bottom-left corner).

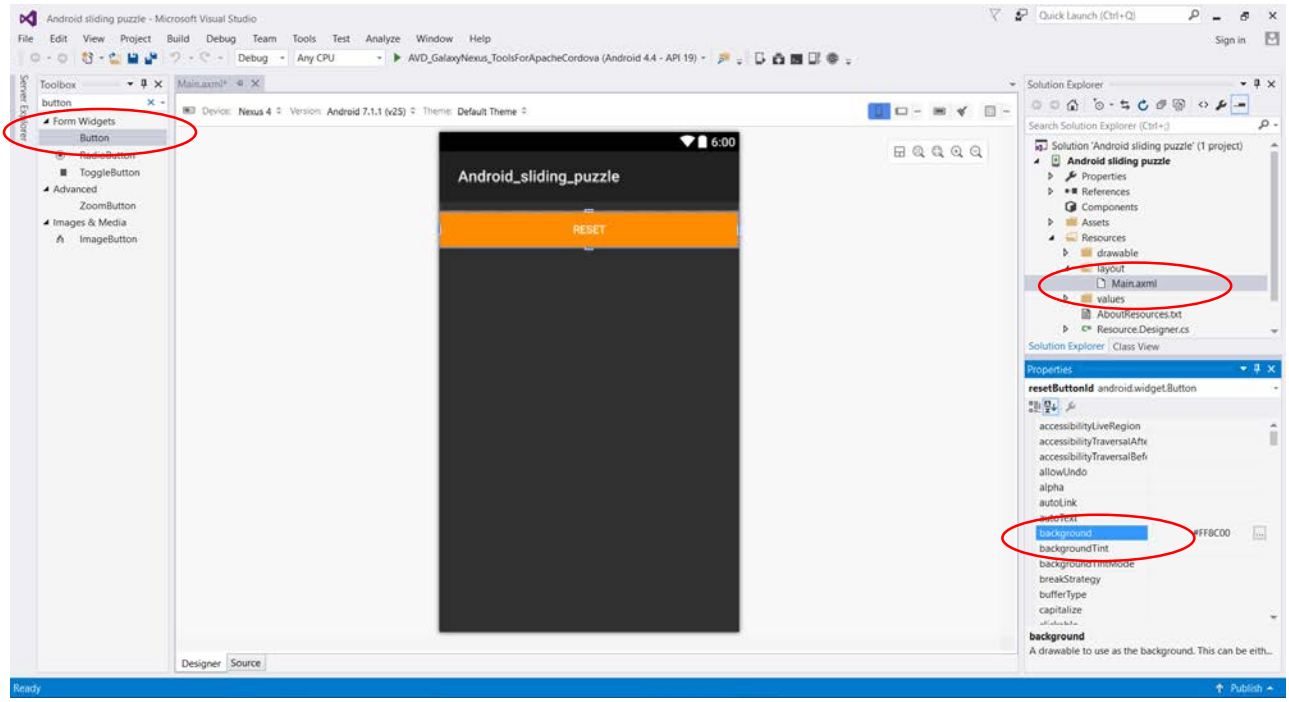


3. In order to add layout buttons, be sure that you have the toolbox activated. **Go to View -> Toolbox and check it**. Now, a menu will all kind of buttons should have appeared. Now look for a 'Grid layout' object and drag and drop one on the screen.

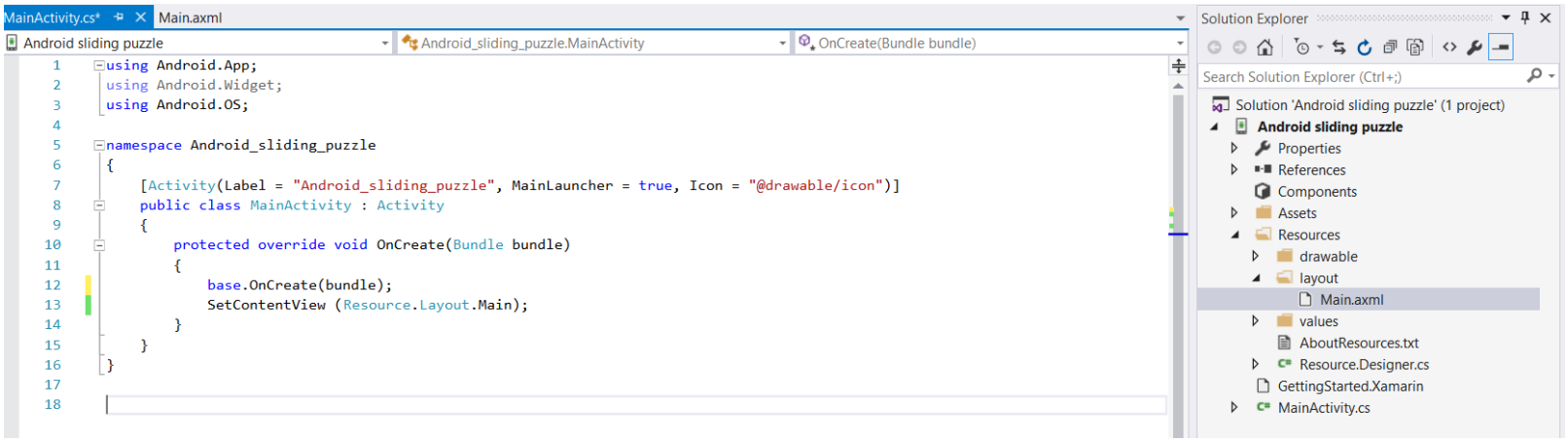


4. Now look for a 'Button' and also drag and drop it on the screen. We need to change the ID's of the two elements we have just created, in order to easily access them in the future. Double click on the grid and in the Properties menu (right side) change the property 'id' to be **'@+id/gameGridLayoutId'**.

   The same thing, click on the button and change it's 'id' property to be **'@+id/resetButtonId'**. Now change the text property to be 'Reset'. Change the background property to be #FF8C00. Save the file and now your app should look like:



5. **Let's start coding!** Go in the MainActivity.cs file and be sure that you have the following code written.




```
using Android.App;
using Android.Widget;
using Android.OS;

namespace Android_sliding_puzzle
{
    [Activity(Label = "Android_sliding_puzzle", MainLauncher = true, Icon = "@drawable/icon")]
    public class MainActivity : Activity
    {
        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);
            SetContentView (Resource.Layout.Main);
        }
    }
}
```

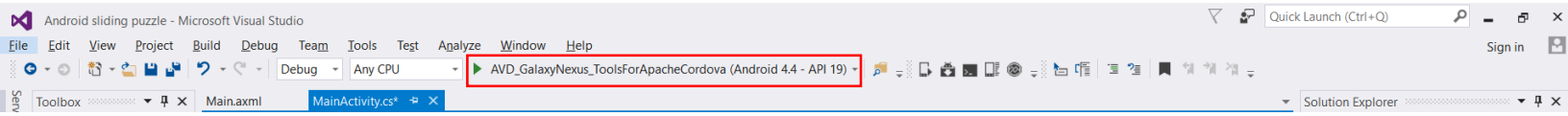
6. We need to get the Grid and button back in the main program, in order to make changes to them. So let's do that by making the following changes in the MainActivity.cs

```
using Android.App;
using Android.Widget;
using Android.OS;

namespace Android_sliding_puzzle
{
    [Activity(Label = "Android_sliding_puzzle", MainLauncher = true, Icon = "@drawable/icon")]

    public class MainActivity : Activity
    {
        #region
        Button resetButton;
        GridLayout mainLayout;
        #endregion

        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);
            SetContentView (Resource.Layout.Main);

            resetButton = FindViewById<Button>(Resource.Id.resetButtonId);
            mainLayout = FindViewById<GridLayout>(Resource.Id.gameGridLayoutId);
        }
    }
}
```

7. Our app should work on multiple Android devices, independent on their screen size. So our grid should change the size according to the screen size. Make the following changes in your code.

```
using Android.App;
using Android.Widget;
using Android.OS;
using Android.Graphics;

namespace Android_sliding_puzzle
{
    [Activity(Label = "Android_sliding_puzzle", MainLauncher = true, Icon = "@drawable/icon")]

    public class MainActivity : Activity
    {
        #region
        Button resetButton;
        GridLayout mainLayout;

        int gameViewWidth;
        #endregion

        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);
            SetContentView (Resource.Layout.Main);

            setGameView();
        }

        private void setGameView ()
        {
            // get the layout elements from the view
            resetButton = FindViewById<Button>(Resource.Id.resetButtonId);
            mainLayout = FindViewById<GridLayout>(Resource.Id.gameGridLayoutId);

            // get the windth of the Android screen
            gameViewWidth = Resources.DisplayMetrics.WidthPixels;

            // set the numbers of rows and colums of the grid
            mainLayout.ColumnCount = 4;
            mainLayout.RowCount = 4;

            // the grid should be of square size, so the height and width are equal to the windth of the phone
            mainLayout.LayoutParameters = new LinearLayout.LayoutParams(gameViewWidth, gameViewWidth);
            // let the colour of the layout to be gray
            mainLayout.SetBackgroundColor(Color.Gray);
        }
    }
}
```

TIP: If you want to check in real time how your app look like, run it on the virtual device.



8. Let's move on to creating the tiles of the grid! As you saw on the first page, the grid is 4 x 4, meaning that it has 16 tiles in total. Let's firstly try to make the top-left tile and then we'll do the others!

```
public class MainActivity : Activity
{
    #region
    Button resetButton;
    GridLayout mainLayout;

    int gameViewWidth;
    int tileWidth;
    #endregion

    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);
        SetContentView (Resource.Layout.Main);

        setGameView();

        makeTilesMethod();
    }

    // method to create the grid layout
    private void makeTilesMethod()
    {
        // the grid is 4 x 4 -> the width of a tile is a quarter of the width of the grid
        tileWidth = gameViewWidth / 4;

        // we want to make a tile and add it in our main layout
        TextView textTile = new TextView(this);

        // we want to put the first tile in the top-left corner
        GridLayout.Spec rowSpec = GridLayout.InvokeSpec(0);
        GridLayout.Spec colSpec = GridLayout.InvokeSpec(0);

        GridLayout.LayoutParams tileLayoutParams = new GridLayout.LayoutParams(rowSpec, colSpec);

        // set the width and height of the tile
        tileLayoutParams.Width = tileWidth;
        tileLayoutParams.Height = tileWidth;

        // make the changes in the actual textTile
        textTile.LayoutParameters = tileLayoutParams;
        textTile.SetBackgroundColor(Color.Green);

        // add the tile in the main layout
        mainLayout.AddView(textTile);
    }

    private void setGameView ()
    {
        // get the layout elements from the view
        resetButton = FindViewById<Button>(Resource.Id.resetButtonId);
        mainLayout = FindViewById<GridLayout>(Resource.Id.gameGridLayoutId);

        // get the windth of the Android screen
        gameViewWidth = Resources.DisplayMetrics.WidthPixels;

        // set the numbers of rows and colums of the grid
        mainLayout.ColumnCount = 4;
        mainLayout.RowCount = 4;

        // the grid should be of square size, so the height and width are equal to the windth of the phone
        mainLayout.LayoutParameters = new LinearLayout.LayoutParams(gameViewWidth, gameViewWidth);
        // let the colour of the layout to be gray
        mainLayout.SetBackgroundColor(Color.Gray);
    }
}
}
```

Run your app the virtual device to be sure it's working.

9. We now need to extend the grid to be 4 x 4, not just one tile. So we need to iterate on each row and column and create a tile and put it there. So we'll use two 'for loops' which will do exactly that (will go on every row and every column) and put a tile there.

   Because we want our app to look great and be downloaded by many people, we'll also leave some space between the tiles so they now intercalate. Now make the following changes in the code.

```
11  public class MainActivity : Activity
12  {
13      #region
14      Button resetButton;
15      GridLayout mainLayout;
16
17      int gameViewWidth;
18      int tileWidth;
19      #endregion
20
21      protected override void OnCreate(Bundle bundle)
22      {
23          base.OnCreate(bundle);
24          SetContentView (Resource.Layout.Main);
25
26          setGameView();
27
28          makeTilesMethod();
29      }
30
31      // method to create the grid layout
32      private void makeTilesMethod()
33      {
34          // the grid is 4 x 4 -> the width of a tile is a quarter of the width of the grid
35          tileWidth = gameViewWidth / 4;
36
37          // we need to create all 16 tiles, they being placed on 4 rows and 4 columns
38          for (int row = 0; row < 4; ++row)
39          {
40              for (int column = 0; column < 4; ++column)
41              {
42                  // we want to make a tile and add it in our main layout
43                  TextView textTile = new TextView(this);
44
45                  // we want to put a tile on the row 'row' and column 'column'
46                  GridLayout.Spec rowSpec = GridLayout.InvokeSpec(row);
47                  GridLayout.Spec colSpec = GridLayout.InvokeSpec(column);
48
49                  GridLayout.LayoutParams tileLayoutParams = new GridLayout.LayoutParams(rowSpec, colSpec);
50
51                  // set the width and height of the tile
52                  tileLayoutParams.Width = tileWidth - 10;  // (-10) for visibility
53                  tileLayoutParams.Height = tileWidth - 10; // (-10) for visibility
54                  tileLayoutParams.SetMargins(5, 5, 5, 5);  // for making a nice looking contour
55
56
57                  // make the changes in the actual textTile
58                  textTile.LayoutParameters = tileLayoutParams;
59                  textTile.SetBackgroundColor(Color.Green);
60
61                  // add the tile in the main layout
62                  mainLayout.AddView(textTile);
63              }
64          }
65      }
66
67      private void setGameView ()
68      {
69          // get the layout elements from the view
70          resetButton = FindViewById<Button>(Resource.Id.resetButtonId);
71          mainLayout = FindViewById<GridLayout>(Resource.Id.gameGridLayoutId);
72
73          // get the windth of the Android screen
74          gameViewWidth = Resources.DisplayMetrics.WidthPixels;
75
76          // set the numbers of rows and colums of the grid
77          mainLayout.ColumnCount = 4;
78          mainLayout.RowCount = 4;
79
80          // the grid should be of square size, so the height and width are equal to the windth of the phone
81          mainLayout.LayoutParameters = new LinearLayout.LayoutParams(gameViewWidth, gameViewWidth);
82          // let the colour of the layout to be gray
83          mainLayout.SetBackgroundColor(Color.Gray);
84      }
85  }
86  }
```

10. Now let's put some text on each tile, such as the number of tile which is from 1 to 16. To do that, we need to take a counter and increase it after every tile is made, so the on the new tile it will be increased to the next value.

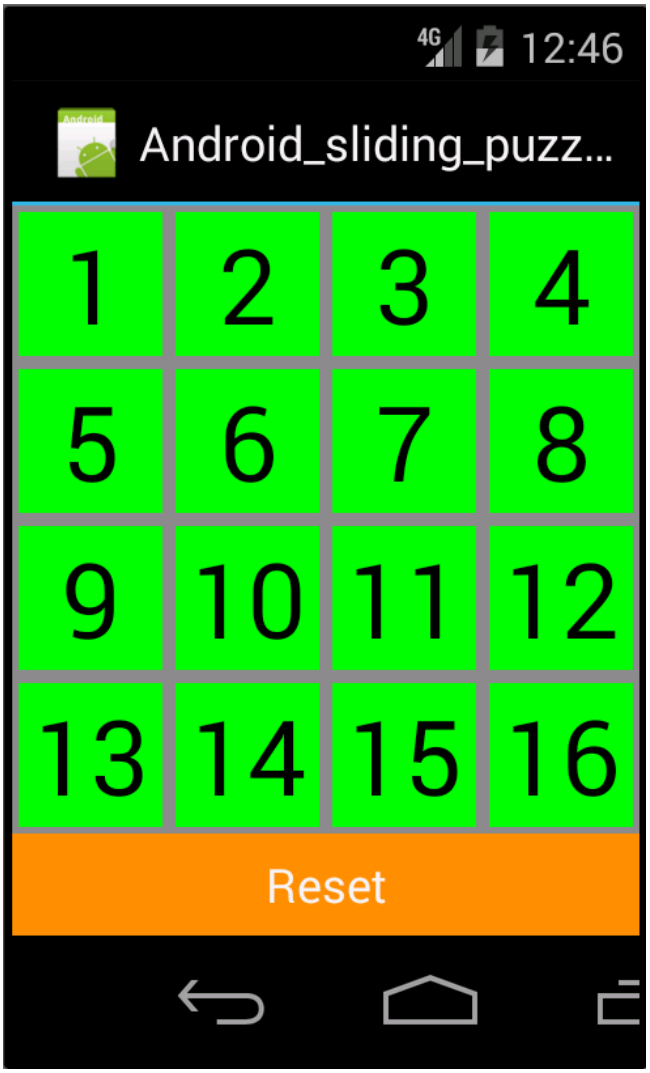
Firstly, add 'using Android.Views;' on top of your file.

```
1  using Android.App;
2  using Android.Widget;
3  using Android.OS;
4  using Android.Graphics;
5  using System;
6  using Android.Views;
```

Secondly, let's write the counter number of every tile. Whenever you get lost in the code, take a look in the left side on the rows number and be sure that you are looking at the same rows ☺

```
38          // create a counter to keep in mind the number of the tile
39          int counter = 1;
40
41          // we need to create all 16 tiles, they being placed on 4 rows and 4 columns
42          for (int row = 0; row < 4; ++row)
43          {
44              for (int column = 0; column < 4; ++column)
45              {
46                  // we want to make a tile and add it in our main layout
47                  TextView textTile = new TextView(this);
48
49                  // we want to put a tile on the row 'row' and column 'column'
50                  GridLayout.Spec rowSpec = GridLayout.InvokeSpec(row);
51                  GridLayout.Spec colSpec = GridLayout.InvokeSpec(column);
52
53                  GridLayout.LayoutParams tileLayoutParams = new GridLayout.LayoutParams(rowSpec, colSpec);
54
55                  // print the number of the tile on it
56                  textTile.Text = counter.ToString();
57                  textTile.SetTextColor(Color.Black);
58                  textTile.TextSize = 40;
59                  textTile.Gravity = GravityFlags.Center;  // put the text in the center of the tile
60
61                  // set the width and height of the tile
62                  tileLayoutParams.Width = tileWidth - 10;  // (-10) for visibility
63                  tileLayoutParams.Height = tileWidth - 10; // (-10) for visibility
64                  tileLayoutParams.SetMargins(5, 5, 5, 5);  // for making a nice looking contour
65
66
67                  // make the changes in the actual textTile
68                  textTile.LayoutParameters = tileLayoutParams;
69                  textTile.SetBackgroundColor(Color.Green);
70
71                  // add the tile in the main layout
72                  mainLayout.AddView(textTile);
73
74                  // increase the counter to the next number
75                  counter = counter + 1;
76              }
77          }
```

This how out app should look like until now



11. We want to randomize the tiles now, so that each new game is unique. For each tile, we need to randomly choose a position where to put that tile. We'll store the positions of the tiles in a list (also called ArrayList), think of it like a list where for each tile number you also write the final position. Then we'll remove the tile 16th, because we need one free space for tiles to move ☺

Firstly, add this line on top of the code:

```
7   using System.Collections;
```

Create the lists for storing information about the tile and their coordiantes. (pay attention to the lines of the code)

```
15      #region
16      Button resetButton;
17      GridLayout mainLayout;
18
19      // lists for storing the tiles and their coordinates
20      ArrayList tilesList;
21      ArrayList coordinatesList;
22
23      int gameViewWidth;
24      int tileWidth;
25      #endregion
```

```
37      // method to create the grid layout
38      private void makeTilesMethod()
39      {
40          // the grid is 4 x 4 -> the width of a tile is a quarter of the width of the grid
41          tileWidth = gameViewWidth / 4;
42
43          // create a counter to keep in mind the number of the tile
44          int counter = 1;
45
46          // initialize the lists for storing the coordinates of the tiles
47          tilesList = new ArrayList();
48          coordinatesList = new ArrayList();
49
50          // we need to create all 16 tiles, they being placed on 4 rows and 4 columns
51          for (int row = 0; row < 4; ++row)
52          {
53              for (int column = 0; column < 4; ++column)
54              {
55                  // we want to make a tile and add it in our main layout
56                  TextView textTile = new TextView(this);
57
58                  // we want to put a tile on the row 'row' and column 'column'
59                  GridLayout.Spec rowSpec = GridLayout.InvokeSpec(row);
60                  GridLayout.Spec colSpec = GridLayout.InvokeSpec(column);
61
62                  GridLayout.LayoutParams tileLayoutParams = new GridLayout.LayoutParams(rowSpec, colSpec);
63
64                  // print the number of the tile on it
65                  textTile.Text = counter.ToString();
66                  textTile.SetTextColor(Color.Black);
67                  textTile.TextSize = 40;
68                  textTile.Gravity = GravityFlags.Center;  // put the text in the center of the tile
69
70                  // set the width and height of the tile
71                  tileLayoutParams.Width = tileWidth - 10;  // (-10) for visibility
72                  tileLayoutParams.Height = tileWidth - 10; // (-10) for visibility
73                  tileLayoutParams.SetMargins(5, 5, 5, 5);  // for making a nice looking contour
74
75
76                  // make the changes in the actual textTile
77                  textTile.LayoutParameters = tileLayoutParams;
78                  textTile.SetBackgroundColor(Color.Green);
79
80                  // keep the coordinates of a tile (as a point in space, where the X coordinate is the column and Y coordiante is the row)
81                  Point thisLocation = new Point(column, row);
82                  // add the coordiante of this point in the coordinatesList
83                  coordinatesList.Add(thisLocation);
84                  // add the tile in the list, to use it later
85                  tilesList.Add(textTile);
86
87                  // add the tile in the main layout
88                  mainLayout.AddView(textTile);
89
90                  // increase the counter to the next number
91                  counter = counter + 1;
92              }
93          }
94
95          // remove the 16th tile -> tilesList start from the position 0 (not 1 as expected), so the 16th element is on the position 15
96          mainLayout.RemoveView((TextView)tilesList[15]);
97          // remove the 16th tile also from our list
98          tilesList.RemoveAt(15);
99      }
```

12. We now want to shuffle the tiles when the games start, so that each new game will be different and challanging! We'll create a method randomizeMethod() for doing that. All the necessary comments are inside the code.

Add this line at the beginning of the code:

```
7   using System.Collections;
```

Next write the code for randomising the puzzle.

```
27          protected override void OnCreate(Bundle bundle)
28          {
29              base.OnCreate(bundle);
30              SetContentView (Resource.Layout.Main);
31
32              setGameView();
33              makeTilesMethod();
34              randomizeMethod();
35          }
```

```
101         private void randomizeMethod()
102         {
103             // take a helper to created random number
104             Random myRand = new Random();
105
106             // store a copy of the coordinates list
107             ArrayList copyCoordsList = new ArrayList(coordinatesList);
108
109             // take each tile (in the variable any) and shuffle it to a new position
110             foreach (TextView any in tilesList)
111             {
112                 // take random coordinates where to put this tile (tile any)
113                 int randIndex = myRand.Next(0, copyCoordsList.Count);
114                 // and store the coordinates in a variable (which is a point in space)
115                 Point thisRandLoc = (Point)copyCoordsList[randIndex];
116
117                 // create a new tile (which is a 1 x 1 part of the whole grid)
118                 GridLayout.Spec rowSpec = GridLayout.InvokeSpec(thisRandLoc.Y);
119                 GridLayout.Spec colSpec = GridLayout.InvokeSpec(thisRandLoc.X);
120                 GridLayout.LayoutParams randLayoutParam = new GridLayout.LayoutParams(rowSpec, colSpec);
121
122                 // set the appearance of the tile to look similar
123                 randLayoutParam.Width = tileWidth - 10;
124                 randLayoutParam.Height = tileWidth - 10;
125                 randLayoutParam.SetMargins(5, 5, 5, 5);
126
127                 // set the tile position to be the new shuffled position
128                 any.LayoutParameters = randLayoutParam;
129
130                 // remove the coordinate, so we can't use it anymore for another tile
131                 copyCoordsList.RemoveAt(randIndex);
132             }
133         }
```

13. Now let's suffle the game everytime the Reset button is pressed. We need to create a method before the setGameView methon and set the button to call it at every press.

```
135         // function to reset the puzzle
136         void resetMethod(object sender, System.EventArgs e)
137         {
138             randomizeMethod();
139         }
140
141         private void setGameView ()
142         {
143             // get the layout elements from the view
144             resetButton = FindViewById<Button>(Resource.Id.resetButtonId);
145             resetButton.Click += resetMethod;
146             mainLayout = FindViewById<GridLayout>(Resource.Id.gameGridLayoutId);
147
148             // get the windth of the Android screen
149             gameViewWidth = Resources.DisplayMetrics.WidthPixels;
150
151             // set the numbers of rows and colums of the grid
152             mainLayout.ColumnCount = 4;
153             mainLayout.RowCount = 4;
154
155             // the grid should be of square size, so the height and width are equal to the windth of the phone
156             mainLayout.LayoutParameters = new LinearLayout.LayoutParams(gameViewWidth, gameViewWidth);
157             // let the colour of the layout to be gray
158             mainLayout.SetBackgroundColor(Color.Gray);
159         }
```

And now everytime we press the Reset button, the puzzle shuffles. Yuhuuu!

14. Our goal now remains to make the tiles move when we touch them. More specifically, if a tile is situated near the empty tile, it should swap its position with the empty tile.

    In order to do that, we need to know where each tile is situated, right? So that when we click on it to change its position to the empty tiles's one. Add the new code under the setGameView() method.

```
141         private void setGameView ()
142         {
143             // get the layout elements from the view
144             resetButton = FindViewById<Button>(Resource.Id.resetButtonId);
145             resetButton.Click += resetMethod;
146             mainLayout = FindViewById<GridLayout>(Resource.Id.gameGridLayoutId);
147 
148             // get the windth of the Android screen
149             gameViewWidth = Resources.DisplayMetrics.WidthPixels;
150 
151             // set the numbers of rows and colums of the grid
152             mainLayout.ColumnCount = 4;
153             mainLayout.RowCount = 4;
154 
155             // the grid should be of square size, so the height and width are equal to the windth of the phone
156             mainLayout.LayoutParameters = new LinearLayout.LayoutParams(gameViewWidth, gameViewWidth);
157             // let the colour of the layout to be gray
158             mainLayout.SetBackgroundColor(Color.Gray);
159         }
160 
161         // add new feature to the TextView, to know it's row and column
162         class MyTextView : TextView
163         {
164             Activity myContext;
165 
166             public MyTextView(Activity context) : base(context)
167             {
168                 myContext = context;
169             }
170 
171             public int xPos { set; get; }
172             public int yPos { set; get; }
173         }
```

15. We now transformed the tile TextView into MyTextView, so we need to make additional changes in the code: where we wrote TextView, we need to change to MyTextView.

1st change in makeTilesMethod()

```
51              for (int row = 0; row < 4; ++row)
52              {
53                  for (int column = 0; column < 4; ++column)
54                  {
55                      // we want to make a tile and add it in our main layout
56                      MyTextView textTile = new MyTextView(this);
```

2nd change also in makeTilesMethod()

```
95              // remove the 16th tile -> tilesList start from the position 0 (not 1 as expected), so the 16th element is on
96              mainLayout.RemoveView((MyTextView)tilesList[15]);
```

3rd change in the randomizeMethod()

```
109             // take each tile (in the variable any) and shuffle it to a new position
110             foreach (MyTextView any in tilesList)
111             {
```

16. Now, we need to make a small change. We told that when we touch on a tile, it should move, right? Yes, but we didn't implement it yet, so let's do that! Add the following code in the two for-loops from the `makeTilesMethod()`.

```
                // keep the coordinates of a tile (as a point in space, where the X coordinate is the column and Y coordiante is the row)
                Point thisLocation = new Point(column, row);
                // add the coordiante of this point in the coordinatesList
                coordinatesList.Add(thisLocation);
                // add the tile in the list, to use it later
                tilesList.Add(textTile);

                // remember the position if the tile
                textTile.xPos = thisLocation.X;
                textTile.yPos = thisLocation.Y;

                // assign a method to execute when we toch the button
                textTile.Touch += TextTile_Touch;

                // add the tile in the main layout
                mainLayout.AddView(textTile);

                // increase the counter to the next number
                counter = counter + 1;
            }
        }
```

And also in the foreach from the randomizeMethod(), let's remember the location of the tiles.

```
            // take each tile (in the variable any) and shuffle it to a new position
            foreach (MyTextView any in tilesList)
            {
                // take random coordinates where to put this tile (tile any)
                int randIndex = myRand.Next(0, copyCoordsList.Count);
                // and store the coordinates in a variable (which is a point in space)
                Point thisRandLoc = (Point)copyCoordsList[randIndex];

                // create a new tile (which is a 1 x 1 part of the whole grid)
                GridLayout.Spec rowSpec = GridLayout.InvokeSpec(thisRandLoc.Y);
                GridLayout.Spec colSpec = GridLayout.InvokeSpec(thisRandLoc.X);
                GridLayout.LayoutParams randLayoutParam = new GridLayout.LayoutParams(rowSpec, colSpec);

                // also keep the location of the tile any
                any.xPos = thisRandLoc.X;
                any.yPos = thisRandLoc.Y;

                // set the appearance of the tile to look similar
                randLayoutParam.Width = tileWidth - 10;
                randLayoutParam.Height = tileWidth - 10;
                randLayoutParam.SetMargins(5, 5, 5, 5);

                // set the tile position to be the new shuffled position
                any.LayoutParameters = randLayoutParam;

                // remove the coordinate, so we can't use it anymore for another tile
                copyCoordsList.RemoveAt(randIndex);
            }
```

17. Now let's create the function that moves the tile when it's touched. We'll create it before the class `MyTextView`.

```
175     // function that executes when the tile is touched
176     void TextTile_Touch(object sender, View.TouchEventArgs e)
177     {
178         if (e.Event.Action == MotionEventActions.Up)
179         {
180             MyTextView thisTile = (MyTextView)sender;
181
182             // just write the position of the tile in the Xamarin console, to see each tile when moves
183             Console.WriteLine("\r tile is at: \r x={0} \r y={1}", thisTile.xPos, thisTile.yPos);
184         }
185     }
186
187     // add new feature to the TextView, to know it's row and column
188     class MyTextView : TextView
189     {
190         Activity myContext;
191
192         public MyTextView(Activity context) : base(context)
193         {
194             myContext = context;
195         }
196
197         public int xPos { set; get; }
198         public int yPos { set; get; }
199     }
```

18. We need to remember where is the empty tile, right? So let's firstly declare a varible to keep this information.

```
15          #region
16          Button resetButton;
17          GridLayout mainLayout;
18
19          // lists for storing the tiles and their coordinates
20          ArrayList tilesList;
21          ArrayList coordinatesList;
22
23          int gameViewWidth;
24          int tileWidth;
25
26          // varible to store where is the empty slot at every moment
27          Point emptySpot;
28          #endregion
```

Then, after the foreach from the randomizeMethod() finishes, we'll memorize where the empty tile is.

```
119             // take each tile (in the variable any) and shuffle it to a new position
120             foreach (MyTextView any in tilesList)
121             {
122                 // take random coordinates where to put this tile (tile any)
123                 int randIndex = myRand.Next(0, copyCoordsList.Count);
124                 // and store the coordinates in a variable (which is a point in space)
125                 Point thisRandLoc = (Point)copyCoordsList[randIndex];
126
127                 // create a new tile (which is a 1 x 1 part of the whole grid)
128                 GridLayout.Spec rowSpec = GridLayout.InvokeSpec(thisRandLoc.Y);
129                 GridLayout.Spec colSpec = GridLayout.InvokeSpec(thisRandLoc.X);
130                 GridLayout.LayoutParams randLayoutParam = new GridLayout.LayoutParams(rowSpec, colSpec);
131
132                 // also keep the location of the tile any
133                 any.xPos = thisRandLoc.X;
134                 any.yPos = thisRandLoc.Y;
135
136                 // set the appearance of the tile to look similar
137                 randLayoutParam.Width = tileWidth - 10;
138                 randLayoutParam.Height = tileWidth - 10;
139                 randLayoutParam.SetMargins(5, 5, 5, 5);
140
141                 // set the tile position to be the new shuffled position
142                 any.LayoutParameters = randLayoutParam;
143
144                 // remove the coordinate, so we can't use it anymore for another tile
145                 copyCoordsList.RemoveAt(randIndex);
146             }
147
148             // we have deleted 15 out of 16 tiles, so the remaining one is the one that is empty
149             emptySpot = (Point)copyCoordsList[0];
150         }
```

19. This is the last step until having a fully functional application! We just need to move the tiles when they are pressed. So, if a tile is touched and it is near the empty space, we'll move it there.

    How do we check this condition? We just have to compute the distance between the touched tile and the empty place, similar with what you've learned at school. The code speaks from itself ☺

```
178         // function that executes when the tile is touched
179         void TextTile_Touch(object sender, View.TouchEventArgs e)
180         {
181             if (e.Event.Action == MotionEventActions.Up)
182             {
183                 MyTextView thisTile = (MyTextView)sender;
184
185                 // just write the position of the tile in the Xamarin console, to see each tile when moves
186                 Console.WriteLine("\r tile is at: \r x={0} \r y={1}", thisTile.xPos, thisTile.yPos);
187
188                 // compute the distace between the tile which was pressed and the empty space
189                 float xDif = (float)Math.Pow(thisTile.xPos - emptySpot.X, 2);
190                 float yDif = (float)Math.Pow(thisTile.yPos - emptySpot.Y, 2);
191                 float dist = (float)Math.Sqrt(xDif + yDif);
192
193                 // if the tile was near the empty space
194                 if (dist == 1)
195                 {
196                     // memorize the current position of the tile
197                     Point curPoint = new Point(thisTile.xPos, thisTile.yPos);
198
199                     // we want to put the tile on the place of the empty space
200                     GridLayout.Spec rowSpec = GridLayout.InvokeSpec(emptySpot.Y);
201                     GridLayout.Spec colSpec = GridLayout.InvokeSpec(emptySpot.X);
202
203                     GridLayout.LayoutParams newLocParams = new GridLayout.LayoutParams(rowSpec, colSpec);
204
205                     // the tile moves in the empty space
206                     thisTile.xPos = emptySpot.X;
207                     thisTile.yPos = emptySpot.Y;
208
209                     // we set the appearance of the tile
210                     newLocParams.Width = tileWidth - 10;
211                     newLocParams.Height = tileWidth - 10;
212                     newLocParams.SetMargins(5, 5, 5, 5);
213
214                     thisTile.LayoutParameters = newLocParams;
215
216                     // the empty place goes where was the pressed tile before
217                     emptySpot = curPoint;
218                 }
219             }
220         }
```

Congratulations! You have built your first Android application!

Now go and show it to your parents and friends, they would not believe you made it. But you know it wasn't that hard, right?