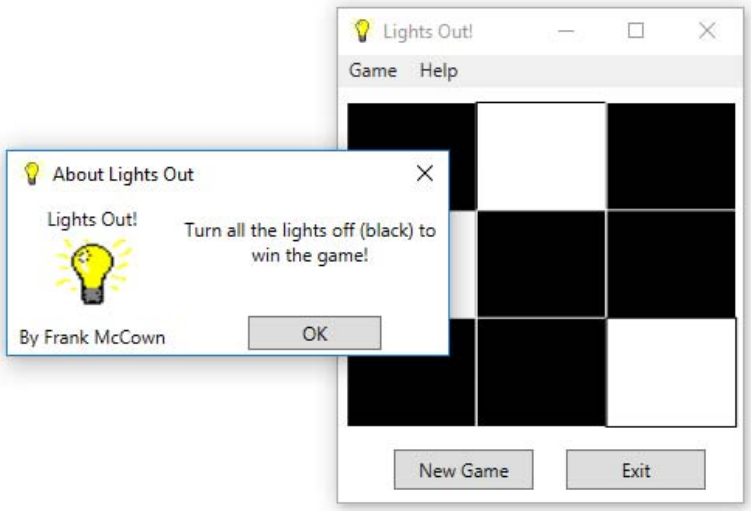
WPF Lights Out!
GUI Programming
10 Points

The purpose of this assignment is for you to learn more about WPF by building a simple WPF application. Your Lights Out! application will work like the Windows Forms program from earlier in the semester. You will need a window icon, menu, buttons, about dialog box, etc. You will see that the WPF version of Lights Out is somewhat different than the Windows Forms version, but they also share many similarities.



Create a WPF project called **LightsOut** in Visual Studio.

Using the LightsOutGame Class

The end of this assignment lists a LightsOutGame class that you must use in your project. The LightsOutGame class encapsulates the game logic but is not tied to the UI. It uses a 2D boolean array to store the on/off values of the grid and exposes a public property and several methods you will use to implement your game.

Add a new class to your project called LightsOutGame, and copy and paste the LightsOutGame code into your new class. Create a class-level variable for LightsOutGame in MainWindow.xaml.cs, and instantiate it in the MainWindow class.

```
private LightsOutGame game;

public MainWindow()
{
   InitializeComponent();

   game = new LightsOutGame();
}
```

Creating the Grid

The Grid layout is the default layout control (or layout manager) used in the and is ideal for laying out the black and white game board and buttons in the window. Within the Grid, you will need a Canvas for representing the game grid.

Your game grid will hold nine Rectangles which represent each of the white/black squares. You can place `Rectangle` controls directly in the Canvas XAML, or you can add the Rectangles at runtime programmatically (which is helpful if you are doing the extra credit).

The `CreateGrid()` function below creates 9 `Rectangle` objects with the proper `Width` and `Height` attributes and adds them to the `Canvas` (named `boardCanvas`) at the appropriate location. Each Rectangle's Tag property stores a Point struct, which indicates the row and column of the Rectangle. It also registers a `MouseLeftButtonDown` event handler, which will later be used to determine which Rectangle was clicked on.

```
private void CreateGrid()
{
   int rectSize = (int)boardCanvas.Width / gameLogic.NumCells;

   // Create rectangles for grid
   for (int r = 0; r < game.NumCells; r++)
   {
```

```
      for (int c = 0; c < game.NumCells; c++)
      {
         Rectangle rect = new Rectangle();
         rect.Fill = Brushes.White;
         rect.Width = rectSize + 1;
         rect.Height = rect.Width + 1;
         rect.Stroke = Brushes.Black;

         // Store each row and col as a Point
         rect.Tag = new Point(r, c);

         // Register event handler
         rect.MouseLeftButtonDown += Rect_MouseLeftButtonDown;

         // Put the rectangle at the proper location within the canvas
         Canvas.SetTop(rect, r * rectSize);
         Canvas.SetLeft(rect, c * rectSize);

         // Add the new rectangle to the canvas' children
         boardCanvas.Children.Add(rect);
      }
   }
}
```

Call the `CreateGrid()` function from the MainWindow's constructor.

Handling Grid Clicks

When the user clicks on a Rectangle, the `Rect_MouseLeftButtonDown()` event handler is called. The method determines which of the 9 buttons the user clicked on by examining the Rectangle's `Tag` property, which stores the row and column of the Rectangle.

```
private void Rect_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)
{
    Rectangle rect = sender as Rectangle;
    var rowCol = (Point)rect.Tag;
    int row = (int)rowCol.X;
    int col = (int)rowCol.Y;

    game.MakeMove(row, col);
}
```

Calling the `MakeMove()` method changes the underlying 2D Boolean array used by the LightsOutGame class, but it does not change what is displayed in the game grid.

Displaying the Grid

The `DrawGrid()` function below it iterates through all the `Rectangle` objects that were previously added to the `boardCanvas` and sets their color appropriately.

```
private void DrawGrid()
{
   int index = 0;

   // Set the colors of the rectangles
   for (int r = 0; r < game.NumCells; r++)
   {
       for (int c = 0; c < game.NumCells; c++)
       {
           Rectangle rect = boardCanvas.Children[index] as Rectangle;
           index++;
```

```
            if (game.GetGridValue(r, c))
            {
                // On
                rect.Fill = Brushes.White;
                rect.Stroke = Brushes.Black;
             }
             else
             {
                // Off
                rect.Fill = Brushes.Black;
                rect.Stroke = Brushes.White;
             }
        }
    }
}
```

Add a call to `DrawGrid()` in the `Rect_MouseLeftButtonDown()` event handler after the move has been made on the `game` object. You must also call `DrawGrid()` after creating the grid in the MainWindow constructor.

Run your program and verify that you can click on the game grid and see the appropriate squares turn on and off (white and black).

Winning the Game

The LightsOutGame class has a method to determine if the player has been won the game. Call this method when a move is made. Use the MessageBox.Show() method to display a "You've won!" message.

Menu

Use the Menu control to create a menu. It should contain MenuItems "Game" and "Help". The "Game" MenuItem should contain MenuItems for "New" and "Exit" that start a new game and exits the application, respectively. A separator (`<Separator />`) should appear before the Exit menu item. The Help menu should contain an "About" menu item that launches the About dialog box.

Here is an example menu item that executes the `MenuExit_Click()` method when selected:

```
<MenuItem Header="E_xit" Click="MenuExit_Click" ToolTip="Exits the application" />
```

Game → New: This option should start a new game by calling the `LightsOutGame.NewGame()` method and `DrawGrid()`.

Game → Exit: The `MenuExit_Click()` method should terminate the app using `Application.Current.Shutdown()`.

Help → About: Display the About dialog box modally.

Make sure the New Game and Exit buttons on your MainWindow call the same event handlers for the New and Exit game menu options.

About Dialog Box

To create an About dialog box, add a **Window (WPF)** to your project. Use some other layout manager besides Grid in order to get some experience with another layout. Display your name, an image, and brief instructions on how to play the

game. To display an image, add the image file to your project and set the Build Action to Resource. Add an Image control and set the Image's Source attribute to the image name.

```
<Image Name="image1" Stretch="Fill" Source="lightbulb.ico" />
```

To close the About dialog, add a Click event handler for your OK button that sets the About Window's `DialogResult` to true.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    this.DialogResult = true;
}
```

Instantiate and show the dialog box modally like you would in a Windows Forms application.

Finishing Up

Make sure you are able to play a game of Lights Out, that the About dialog box displays, etc. You can make some improvements for extra points:

- **1 bonus point** for adding the ability to resize the window and have the grid increase/decrease in size dynamically to fill the window. Hint: A Grid control will do this automatically, or if you use a Canvas then use a ViewBox to handle the resizing. There is no code necessary to write to get the resize to work.

- **1 bonus point** for adding the ability to change the game grid size to 3, 5, or 7. Use a menu with checked menu items. Make sure only one menu item is checked at a time. When another is checked, the game board should use the selected size, and a new game should be started.

Zip up your entire project and submit it to Easel before class on the due date. Make sure your about dialog identifies **you** as the author.

LightsOutGame Class

```
using System;

namespace LightsOut
{
    class LightsOutGame
    {
        private int numCells = 3;               // Number of cells in grid

        private bool[,] grid;                   // Stores on/off state of cells in grid
        private Random rand;                    // Used to generate random numbers

        // Returns the number of horizontal/vertical cells in the grid
        public int NumCells
        {
            get
            {
                return numCells;
            }
            set
            {
                if (value >= 3 && value <= 5)
                {
                    numCells = value;
                }
                else
                {
                    throw new ArgumentOutOfRangeException("NumCells may only be 3, 4, or 5.");
                }
```

```
            }
        }

        public LightsOutGame()
        {
            rand = new Random();    // Initialize random number generator

            grid = new bool[numCells, numCells];

            // Turn entire grid on
            for (int r = 0; r < numCells; r++)
            {
                for (int c = 0; c < numCells; c++)
                {
                    grid[r, c] = true;
                }
            }
        }

        // Returns the grid value at the given row and column
        public bool GetGridValue(int row, int col)
        {
            return grid[row, col];
        }

        // Randomizes the grid
        public void NewGame()
        {
            grid = new bool[numCells, numCells];

            // Fill grid with either white or black
            for (int r = 0; r < numCells; r++)
            {
                for (int c = 0; c < numCells; c++)
                {
                    grid[r, c] = rand.Next(2) == 1;
                }
            }
        }

        // Inverts the selected box and all surrounding boxes
        public void MakeMove(int row, int col)
        {
            for (int i = row - 1; i <= row + 1; i++)
            {
                for (int j = col - 1; j <= col + 1; j++)
                {
                    if (i >= 0 && i < numCells && j >= 0 && j < numCells)
                    {
                        grid[i, j] = !grid[i, j];
                    }
                }
            }
        }

        // Returns true if all cells are off
        public bool PlayerWon()
        {
            for (int r = 0; r < numCells; r++)
            {
                for (int c = 0; c < numCells; c++)
                {
                    if (grid[r, c])
                    {
                        return false;
                    }
                }
            }

            return true;
        }
    }
}
```