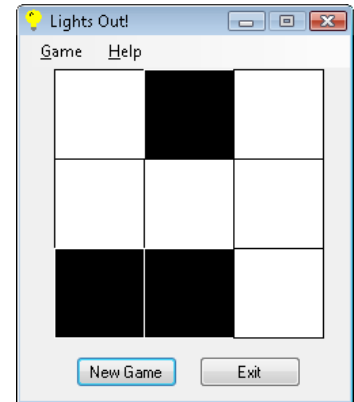


Assignment 1 – **Lights Out!** in C#
GUI Programming
10 points

Introduction

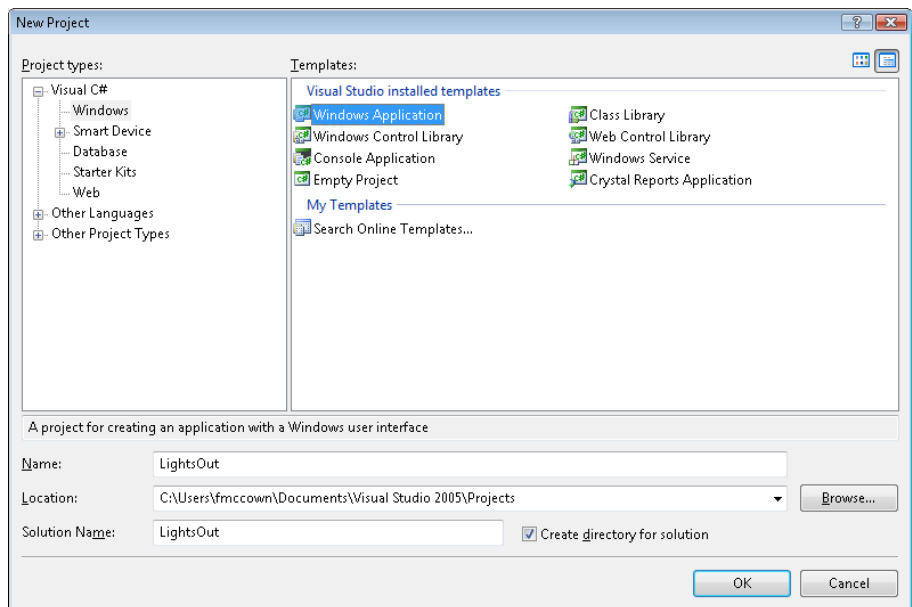
In this lab you will create a simple C# application with a menu, some buttons, and an About dialog box. You will learn how to create a menu, handle button presses, handle menu selections, display child forms, use GDI+ functions, and display a message dialog box.

Lights Out! is a simple game in which the user is presented a 3 by 3 grid of randomly on (white) and off (black) squares. The user tries to turn the entire grid off by clicking the appropriate squares. When a square is pressed, it inverts itself and any squares immediately surrounding it (including diagonal squares). Take some time to play with my version of the game which is located at <http://www.harding.edu/fmccown/gui/lightsout.exe>



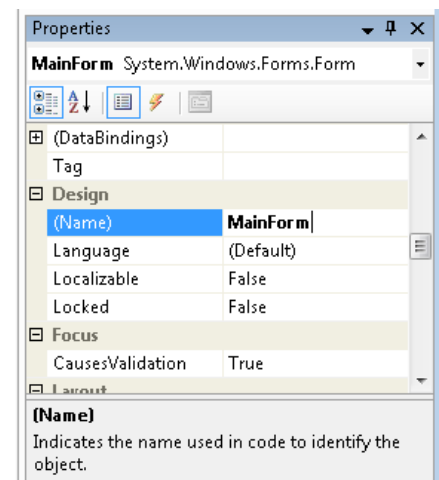
Create a New Project

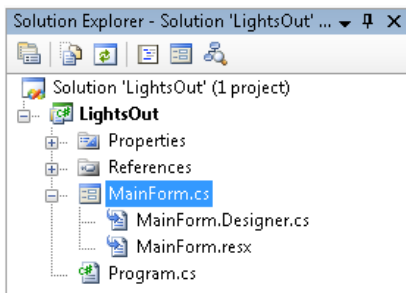
Start a new Visual C# **Windows Application** project by selecting File → New → Project... from the menu. From the dialog box that appears (pictured below), create a Project named LightsOut; the Solution will by default use the same name. Note that the default location will be in your Projects folder.



Create the Main Form

Using the **Properties** window (pictured right), change the **Name** of the form to "MainForm" and the **Text** (title bar caption) to "Lights Out!". If you do not see a Properties window, select View → Properties Window from the menu. Save the form by selecting File → Save Form1 As... from the menu, and save it as MainForm.cs. Although it's not necessary that the .cs file be named the same as the class defined inside it like it is in Java, this is still a good convention to follow.



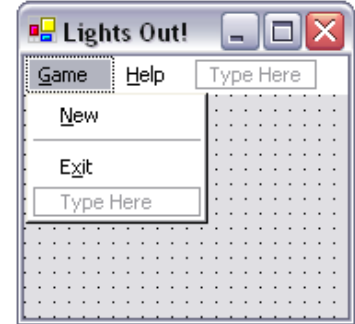


Once you have renamed the form, you will see three .cs files in your project from the **Solution Explorer** (pictured left):

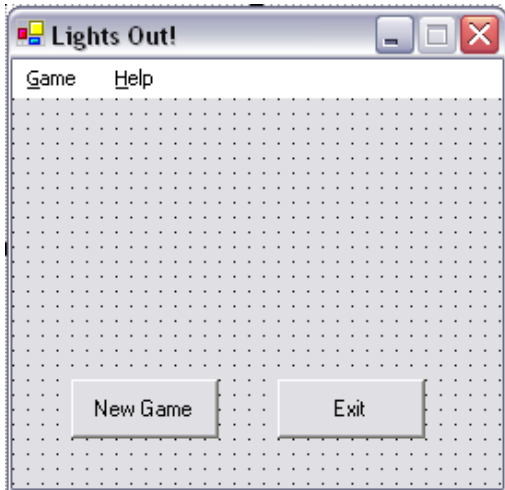
1. MainForm.cs – will contain all the code that you write.
2. MainForm.Designer.cs – contains all the code pertaining to the form that VS will produce automatically for you (so editing this file is not advised).
3. Program.cs – contains Main(), the entry point for the application, and is automatically produced by VS.

Add a menu to the MainForm by placing a **MenuStrip** from the Toolbox onto the form. If

you don't see the Toolbox, make it visible by selecting View → Toolbox from the menu. Create “New” and “Exit” options under “Game” and “About” under “Help” as in the illustration to the right. Place an “&” character before each letter to make it underlined. Give each menu item an appropriate name like “NewMenuItem”. A “-” is used to create a separator (to separate New and Exit).



Add two **Button** objects to the form named “NewGameButton” and “ExitButton”, and place them at the bottom of the form as shown in the illustration below. Change the Text properties of the buttons to “New Game” and “Exit”.



Run the program (Ctrl-F5) to see how the application behaves at this point. When you click the buttons or select items from the menu, nothing happens. The window is re-sizeable, but the buttons are fixed in their current position. Terminate the application by clicking the close button on the window.

Because our game window should always remain the same size, we should disable the ability to re-size it. Do this by changing the **FormBorderStyle** property of the main form to FixedSingle. Remove the maximize button by changing the MainForm's **MaximizeBox** property to false. In order to make the game window pop-up in the center of the screen, change the main form's **StartPosition** property to CenterScreen.

The application should also use a custom icon. From the menu, select Project → Add New Item... Select Icon File from the dialog box and name the icon file anything you want. The icon will display in edit mode, allowing you to modify it. There are two versions of the icon stored in the same file- the 32x32 pixel icon, and the 16x16 pixel icon. Draw a picture of a light bulb for **both** the 32x32 and 16x16 pixel icons. You can switch between the two icons by selecting *Image* → *Current Icon Image Types* → *16x16, 16 colors* and *32x32, 16 colors* from the menu.

Once you have finished making a light bulb icon, add the icon to the MainForm by selecting the MainForm in design view and setting its **Icon** property to the .ico file you just created. You should now see the icon in the title bar of the MainForm. Also select Project → LightsOut Properties... from the menu and under the Application tab, set the icon to the .ico file you created. This will make the .exe file use your icon instead of the default one.

Adding Code

Now we will start adding the code behind the form. To change to code view, select View → Code from the main menu. You will now have two windows, the *code view* (with the tab labeled “MainForm.cs”) and the *design view* (with the tab labeled “MainForm.cs [Design]”).

We'll use a 3 by 3 boolean array to keep track of our grid. Declare the constants, Grid array, and Random object with all the other data members of MainForm above the MainForm constructor like so:

```
private const int GRID_OFFSET = 25; // Distance from upper-left side of window
private const int GRID_LENGTH = 200; // Size in pixels of grid
private const int NUM_CELLS = 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
```

When the program is first executed, the entire grid should be “on.” All initialization and start-up processing should be done in the form’s constructor, so enter the following code in the MainForm’s constructor (after the call to `InitializeComponent()`) to initialize the random number generator and grid:

```
Rand = new Random(); // Initializes random number generator

Grid = new bool[NUM_CELLS, NUM_CELLS];

// Turn entire grid on
for (int r = 0; r < NUM_CELLS; r++)
    for (int c = 0; c < NUM_CELLS; c++)
        Grid[r,c] = true;
```

We need to add the logic to display the Grid to the screen. Just like programming the Win32 API, all of our painting should be done when processing the paint message. Add a **Paint** event handler to the MainForm by first selecting the form in the design view. Click the Event button in the Properties window (lightening bolt), and double-click in the space to the right of the Paint event and enter the following code:

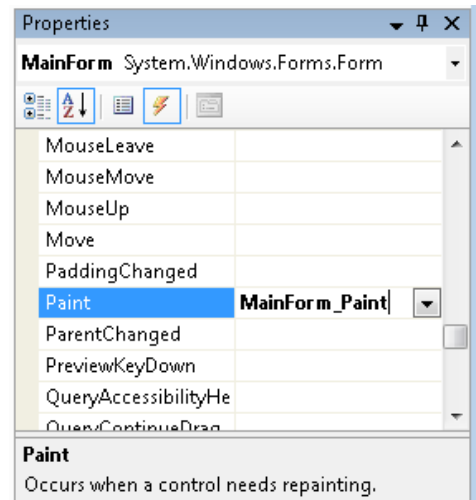
```
private void MainForm_Paint(object sender,
    System.Windows.Forms.PaintEventArgs e)
{
    Graphics g = e.Graphics;

    for (int r = 0; r < NUM_CELLS; r++)
        for (int c = 0; c < NUM_CELLS; c++)
        {
            // Get proper pen and brush for on/off
            // grid section
            Brush brush;
            Pen pen;

            if (Grid[r,c])
            {
                pen = Pens.Black;
                brush = Brushes.White; // On
            }
            else
            {
                pen = Pens.White;
                brush = Brushes.Black; // Off
            }

            // Determine (x,y) coord of row and col to draw rectangle
            int RectSize = GRID_LENGTH / NUM_CELLS;
            int x = c * RectSize + GRID_OFFSET;
            int y = r * RectSize + GRID_OFFSET;

            g.DrawRectangle(pen, x, y, RectSize, RectSize); // Rectangle outline
            g.FillRectangle(brush, x+1, y+1, RectSize-1, RectSize-1); // Solid rectangle
        }
}
```



Now run the application to verify that you’ve entered the code correctly so far. The grid should now appear all white. Clicking on it doesn’t do anything right now. Close the application and continue.

Let’s now add the logic to handle mouse clicks on the Grid. Select the form in the design view and create a callback for the **MouseDown** event. Enter the following code:

```
private void MainForm_MouseDown(object sender, System.Windows.Forms.MouseEventArgs e)
{
    // Find row, col of mouse press
    int RectSize = GRID_LENGTH / NUM_CELLS;
    int r = (e.Y - GRID_OFFSET) / RectSize;
    int c = (e.X - GRID_OFFSET) / RectSize;

    // Invert selected box and all surrounding boxes
```

```

for (int i = r-1; i <= r+1; i++)
    for (int j = c-1; j <= c+1; j++)
        if (i >= 0 && i < NUM_CELLS && j >= 0 && j < NUM_CELLS)
            Grid[i,j] = !Grid[i,j];

// Redraw grid
this.Invalidate();

// Check to see if puzzle has been solved
if (PlayerWon())
{
    // Display winner dialog box just inside window
    MessageBox.Show(this, "Congratulations! You've won!", "Lights Out!",
        MessageBoxButtons.OK, MessageBoxIcon.Information);
}
}
}

```

Notice the call to PlayerWon. It's a boolean function that should return true if all the squares in the grid are off, false otherwise. Place the PlayerWon method anywhere you'd like in the MainForm class. Below I've provided the function skeleton. You must write the heart of the function yourself.

```

private bool PlayerWon()
{
    // Write the function code here
}

```

At this point, we have a working game except that we need to be able to start a new game by randomly turning each square of the Grid on or off. Create a callback for clicking the "New Game" button by double-clicking the "New Game" button in design mode. Use the Random object's Next method which returns a random integer to set each cell of the Grid to on or off. Enter the following code:

```

private void NewGameButton_Click(object sender, System.EventArgs e)
{
    // Fill grid with either white or black
    for (int r = 0; r < NUM_CELLS; r++)
        for (int c = 0; c < NUM_CELLS; c++)
            Grid[r,c] = Rand.Next(2) == 1 ? true : false;

    // Redraw grid
    this.Invalidate();
}

```

When "New" is selected from the "Game" menu, the same code should be executed. Instead of duplicating the same code you just entered, simply call the NewGameButton_Click function when receiving the **Click** event for the New menu item. Do this by double-clicking the New menu item in design mode and enter the following code:

```

private void NewMenuItem_Click(object sender, System.EventArgs e)
{
    NewGameButton_Click(sender, e);
}

```

Now let's add the logic to terminate the application when the Exit button is clicked or Exit is selected from the menu. Create a callback for each of these two events like you did for the new game selections. The Close() method closes the form which is calling it and terminates the program if the main form is calling it.

```

private void ExitButton_Click(object sender, System.EventArgs e)
{
    this.Close();
}

```

Run the application. First make sure the game can determine if you win by clicking the middle square of the grid which will invert all of the white squares to black. A dialog box should pop-up saying you've won. Now Click the "New Game" button a few times and select "New" from the "Game" menu to make sure the Grid is randomly lit each time. Play a game or two to verify the mouse clicks are working. Close the application using the "Exit" button or selecting "Exit" from the menu.

Creating an About Dialog Box

Now we want to add an about dialog box so the user will know something about the developer of this application. Select Project → Add Windows Form... from the menu. When the Add dialog appears, name the form's C# file AboutForm.cs and press Open. This will add AboutForm.cs to our project and display a blank Windows form.

The **Name** of the form should be set to "AboutForm". Change the **Text** property to "About". To make the dialog appear in the center of the MainForm, set the **StartPosition** to CenterParent. To remove the minimize and maximize buttons, set the **MaximizeBox** and **MinimizeBox** properties to False. To make the form fixed, change the **FormBorderStyle** to FixedDialog.

Instead of processing the paint event for the form and displaying text, it's often easier to use labels. Add some **Labels** to the AboutForm from the Toolbox. Try experimenting with different fonts and colors by modifying the **Font** and **ForeColor** properties. Display the name of the application ("Lights Out!"), your name, and some instructions on how to play the game.

Add a **PictureBox** control to the form from the Toolbox to display a light bulb image. Click on the "..." button next to the Image property of the PictureBox and select the light bulb icon (.ico file) you created earlier.

Add an OK **Button** to the form, and add a **Click** even handler to close the AboutForm.



Now lets add the logic to display the About dialog box. Because the AboutForm.cs is selected into your project, launching the AboutForm in your MainForm.cs program will just require you to create an AboutForm object and display it when About is selected from the menu. Add the declaration in MainForm.cs along with the other properties:

```
private AboutForm AboutBox;           // About this program
```

Add a **Click** event handler to your About menu item, and display the dialog modally like this:

```
if (AboutBox == null)
    AboutBox = new AboutForm();
AboutBox.ShowDialog(this);
```

Now make sure to change the AboutForm's FormBorderStyle, MinimizeBox, MaximizeBox, and StartPosition to appropriate values.

Finishing Up

If you've typed everything in correctly, you should now have a fully functional Lights Out! game that also displays a nifty About dialog box. After you've finished testing it, submit your working executable to Easel for testing.

*"...Let your light shine before men,
that they may see your good deeds and praise your Father in heaven."*
Matthew 5:16