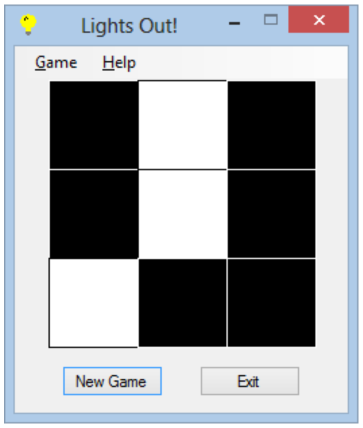
Lights Out! with Windows Forms

GUI Programming

10 points

Introduction

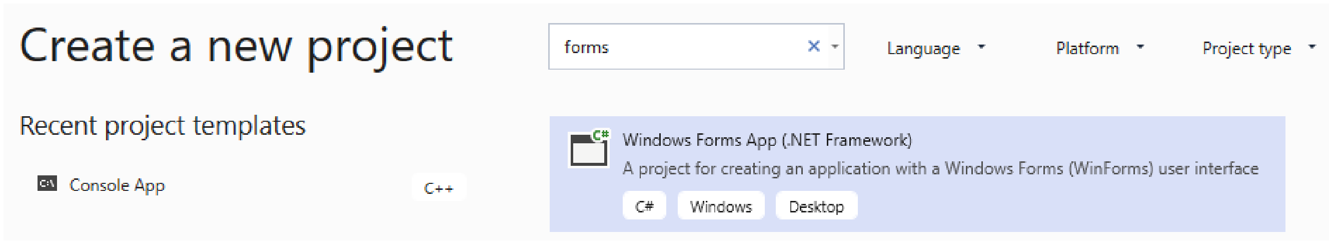


In this assignment, you will create a Windows Forms application in C# with a menu, some buttons, and an About dialog box. You will learn how to create a menu, handle button presses, handle menu selections, display child forms, use GDI+ functions, and display a message dialog box.

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 \\cs1\classes\Comp445\WinForms\LightsOut

Create a New Project

Start Visual Studio 2019 and click the **Create a new project** button on the right. Enter "forms" in the search box, click the **Windows Forms App (.NET Framework)** option, and click Next.



Create a new project

forms

Language Platform Project type

Recent project templates

Console App C++

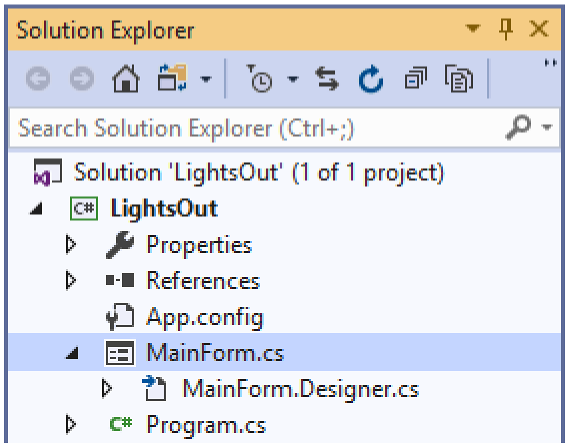
Windows Forms App (.NET Framework)

A project for creating an application with a Windows Forms (WinForms) user interface

C# Windows Desktop

Name the project **LightsOut.** The Solution will by default use the same name as the project. The default location will be in your source\repos folder. The most recent version of .NET Framework will already be selected. Click Create to create the solution and project.

Create the Main Form

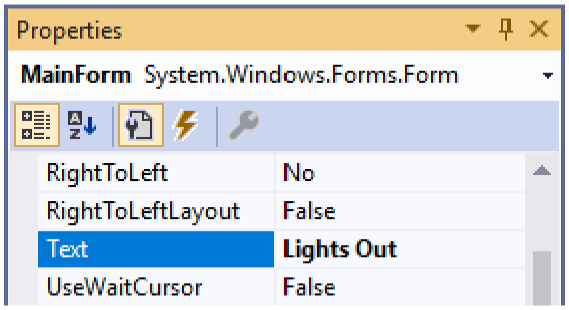


After creating the project, a single form called Form1 will be added to your project. Naming your primary window's form MainForm is good practice. Right-click on Form1.cs in the Solution Explorer and choose Rename from the popup menu. Change the name to MainForm.cs. Choose Yes, if asked if all references to Form1 should be changed.

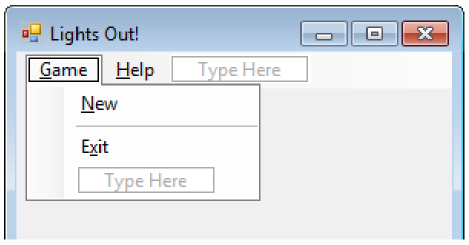
Once you have renamed the form, you will see three .cs files in your project from the **Solution Explorer**:

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()` and instantiates MainForm.

Change the MainForm's title bar text by selecting MainForm and changing the **Text** property to "Lights Out!" in the Properties window, as shown on the right.

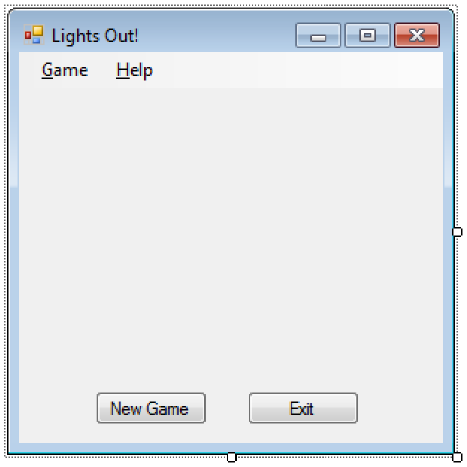


RightToLeft	No
RightToLeftLayout	False
Text	Lights Out
UseWaitCursor	False



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" with a separator (use a dash "-") as in the illustration on the left. Create an "About" item under "Help". Place an "&" character before each letter to make it underlined. This allows the user to press Alt-*x* (where *x* is an underlined letter) to access the menu without using their mouse; this is called a **menu accelerator**. Notice that VS gives each menu item a descriptive name that incorporates the label like `newToolStripMenuItem` for New.

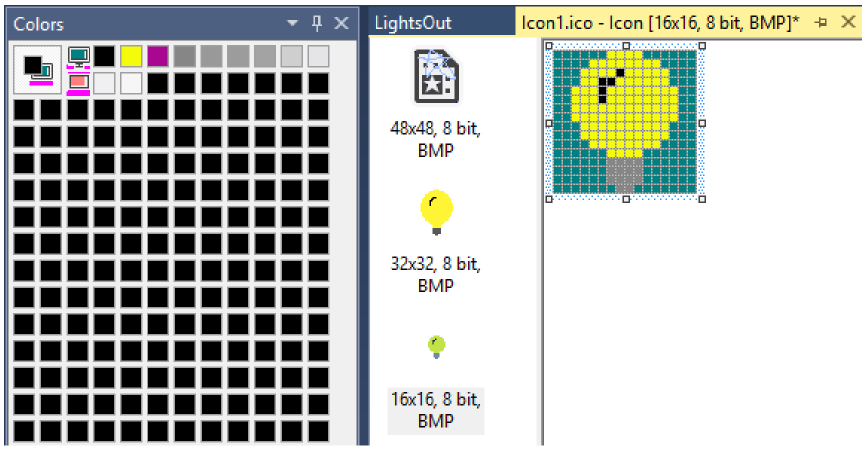
Add two **Button** objects to the form named `newGameButton` and `exitButton`, and place them at the bottom of the form as shown in the illustration below. Change the Text properties of the buttons to "New Game" and "Exit".



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 Visual Studio menu, select *Project →Add New Item…* Select **Icon File** from the dialog box, name the icon file anything you want, and click Add. The icon will display in edit mode, allowing you to modify it. There are multiple versions of the icon stored in the same file with different sizes (256x256, 64x64, etc.) and bits (32 bit and 8 bit).



Visual Studio's editor doesn't allow you to modify the 32 bit versions, so right-click each of the 32 bit images and select "Delete Image Type" from the context menu. You should only have the 48x48, 32x32, and 16x16 8 bit images remaining.

Draw a picture of a light bulb for **both** the 32x32 and 16x16 pixel icons. You can change the colors that are available by double-clicking a color and choosing a new one. Various drawing tools are available in the toolbar above the image.

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 may need to navigate to the project folder to locate the .ico file.

When you build and run the application, you should see the 16x16 icon in the window's title bar and the 32x32 icon in the task bar. 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 code that controls the game logic. To change to code view, select *View → Code* from the main menu. You will now have two tabs open, the *code view* (the tab labeled "MainForm.cs") and the *design view* (the tab labeled "MainForm.cs [Design]").

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

```
namespace LightsOut
{
    public partial class MainForm : Form
    {
        private const int GridOffset = 25;    // Distance from upper-left side of window
        private const int GridLength = 200;   // Size in pixels of grid
        private const int NumCells = 3;       // Number of cells in grid
        private const int CellLength = GridLength / NumCells;

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

        public MainForm()
        {
            InitializeComponent();
        }
    }
}
```

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 to initialize the random number generator and grid:

```
        public MainForm()
        {
            InitializeComponent();

            rand = new Random();    // Initializes 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;

        }
```

We need to add the logic to display the grid to the screen. To paint the screen, we need to handle the form's Paint event. 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. This will automatically create a name for the event handler (shown below) and transfer you to code view.

Enter the following code which will draw rectangles to form the white and black grid:

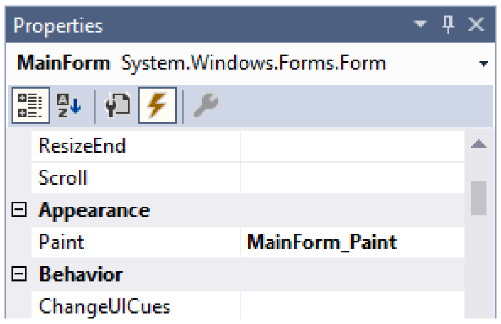
```
private void MainForm_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;

    for (int r = 0; r < NumCells; r++)
    {
        for (int c = 0; c < NumCells; 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 x = c * CellLength + GridOffset;
            int y = r * CellLength + GridOffset;

            // Draw outline and inner rectangle
            g.DrawRectangle(pen, x, y, CellLength, CellLength);
            g.FillRectangle(brush, x + 1, y + 1, CellLength - 1, CellLength - 1);
        }
    }
}
```



MainForm System.Windows.Forms.Form

ResizeEnd	
Scroll	
⊟ Appearance	
Paint	MainForm_Paint
⊟ Behavior	
ChangeUICues	

Run the application to verify that the grid appears all white. Clicking on it doesn't do anything. Close the application and continue.

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 (triggered when pressing a mouse button). Enter the following code:

```
private void MainForm_MouseDown(object sender, System.Windows.Forms.MouseEventArgs e)
{
   // Make sure click was inside the grid
   if (e.X < GridOffset || e.X > CellLength * NumCells + GridOffset ||
       e.Y < GridOffset || e.Y > CellLength * NumCells + GridOffset)
     return;

  // Find row, col of mouse press
  int r = (e.Y - GridOffset) / CellLength;
  int c = (e.X - GridOffset) / CellLength;
```

```
          // 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 < NumCells && j >= 0 && j < NumCells)
                 grid[i,j] = !grid[i,j];

          // Redraw grid
          this.Invalidate();

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

Notice the call to `PlayerWon()` above. 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, you have a working game except that starting a new game should randomly turn each square of the grid on or off. Create a **Click** event listener for 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 like so:

```
        private void newGameButton_Click(object sender, EventArgs e)
        {
            // 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;

            // 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 `newGameButton_Click()` 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 newToolStripMenuItem_Click(object sender, EventArgs e)
        {
            newGameButton_Click(sender, e);
        }
```

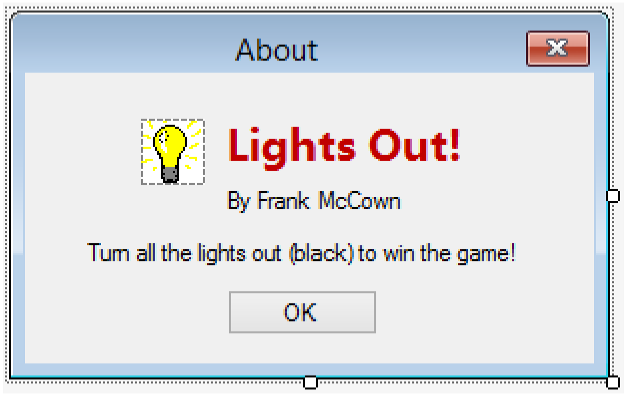
Add the logic to terminate the application when the Exit button is clicked or Exit is selected from the menu. Create a callback for each of these two events like you did for the new game selections. The `Close()` method closes the form and terminates the program if the main form is calling it.

```
        private void exitButton_Click(object sender, EventArgs e)
        {
            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 main menu. When the dialog box appears, select Windows Form and name the form’s AboutForm.cs and press Add. This will add AboutForm.cs to our project and display a blank Windows form. Note that you could also select AboutBox from the list of new items which creates a standard-looking about dialog box.



Turn all the lights out (black) to win the game!

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 a fixed size, change the **FormBorderStyle** to FixedDialog.

To display text, 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. Create a .bmp or .png with a light bulb using your favorite image editor and save it in the LightsOut project directory. Then click on the “…” button next to the Image property of the PictureBox, select “Local resource”, click Import, and select the light bulb file you just created. Then click OK, and you should see the image in the PictureBox.

Add an OK **Button** to the form, and set its **DialogResult** property to OK. This will cause the button to dismiss the dialog box when pressed.

Now let’s 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 require you to instantiate an AboutForm object and display it when About is selected from the menu.

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

```
AboutForm aboutBox = new AboutForm();
aboutBox.ShowDialog(this);
```

The ShowDialog() method will not return until the user presses the OK button on the dialog box to dismiss it.

Extra Credit

1. (1 pt) Allow the user to change the size of the grid by adding a main menu item called “Size” which has three options underneath: 3x3, 4x4, and 5x5. The 3x3 item should be initially checked. Selecting 4x4 causes the board to be displayed with 4 rows and 4 columns, and selecting 5x5 changes to 5 rows and 5 columns. The selected menu item should also now be the only item checked. This will require some reworking of the code. You’ll have to do some googling to figure out how to make a menu item checked.

2. (1 pt) Allow the window to be resized, and make the game board resize along with it. The buttons should remain anchored at the bottom of the window as it is resized, and the board should remain a square. This will require you to modify some properties of the two buttons and to create a Resize event

handler that recalculates the grid length based on the new width or height of the form. The handler will also need to call the form’s Invalidate() method so that it triggers the form’s Paint event handler which repaints the board.

Finishing Up

If you’ve typed everything in correctly, you should now have a fully functional Lights Out! game that also displays an About dialog box. After you’ve finished testing it, zip your entire project and submit the zip file to Easel. If you worked in pairs, only one person needs to submit the solution.