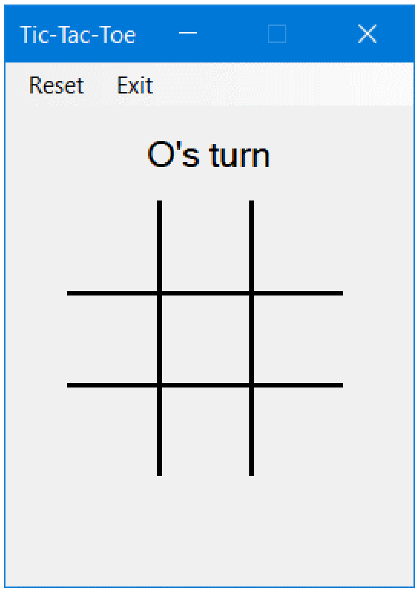
Assignment Purpose

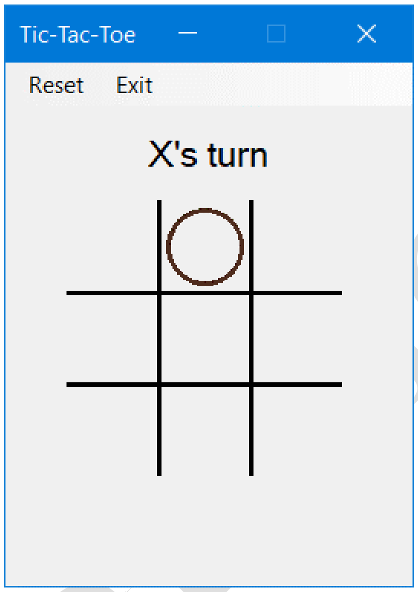
This lab will give you practice creating Windows Forms Applications, menus, drawing (GDI+).

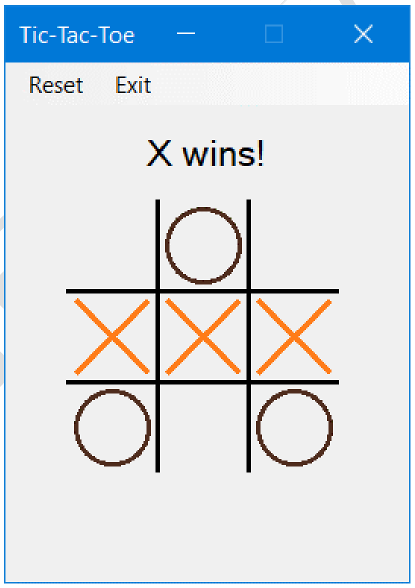
You can work in your groups (2 members). If your partner is not present in class, each will need to complete the assignment on your own by the due date/time.

Mandatory Instructions

In this assignment, you will implement the TicTacToe game as a Windows Forms application. The Game will allow two humans to play with each other, with the computer displaying the playing grid and the results. When the form initially loads, it looks like the diagram on the left below. After three moves, it would look like the diagram on the right.

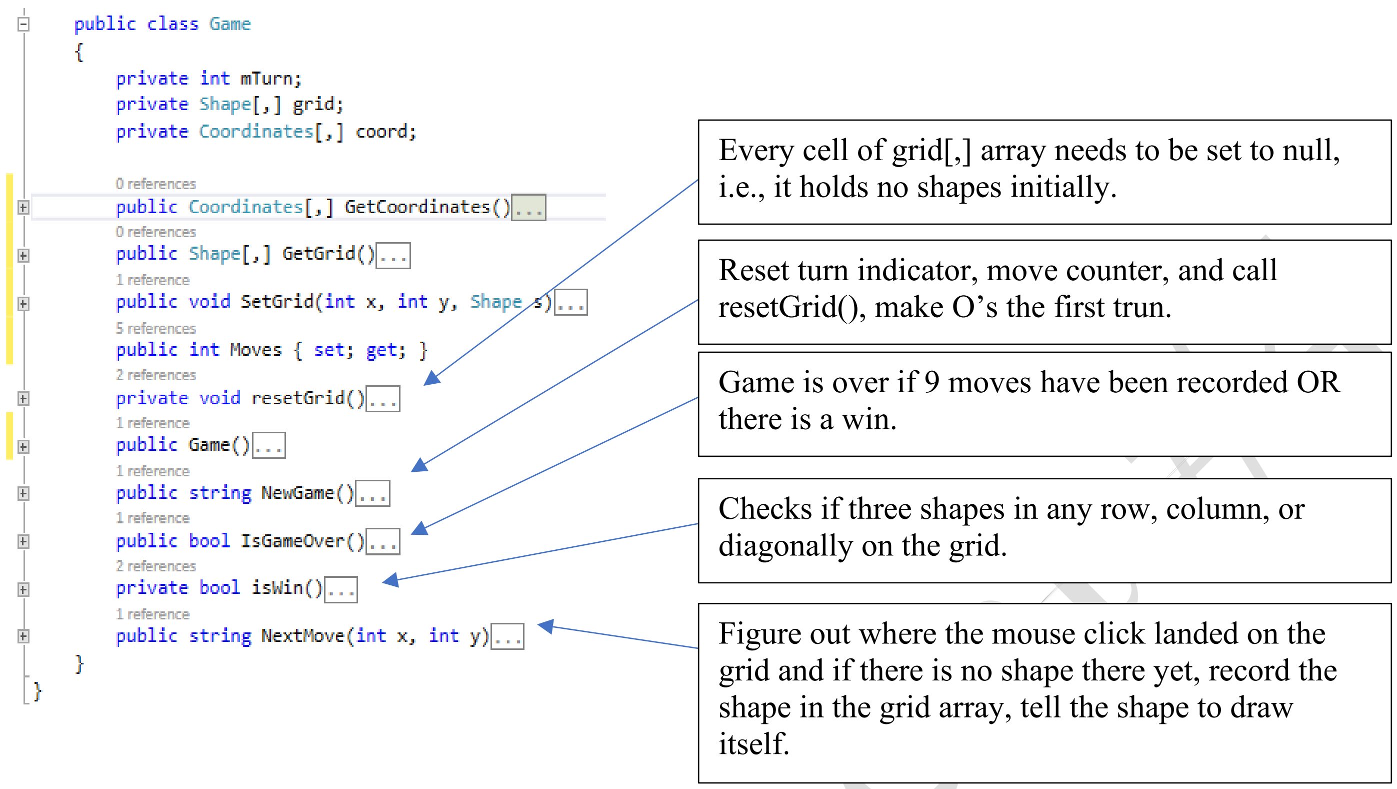






Your form should be set so that it can't be resized. The only controls in this form are one label (initially displaying "O's turn") and one menu containing two items (Reset, Exit). The grid must be perfectly square. I made my grid boxes 50 dialog units wide/tall. It should start just below the label and a little to the right of the window's left border. Creating the grid may be a bit of a trial and error exercise. You can use graph paper to lay out your grid on the form.

To support n-tier application design, define a class called Game that contains all of the TicTacToe logic. The form, i.e., the presentation tier, will capture the click events and other events and pass them to the Game class functions. Below are examples of some methods, properties, fields for the Game class.



```
public class Game
{
    private int mTurn;
    private Shape[,] grid;
    private Coordinates[,] coord;

    0 references
    public Coordinates[,] GetCoordinates()...
    0 references
    public Shape[,] GetGrid()...
    1 reference
    public void SetGrid(int x, int y, Shape s)...
    5 references
    public int Moves { set; get; }
    2 references
    private void resetGrid()...
    1 reference
    public Game()...
    1 reference
    public string NewGame()...
    1 reference
    public bool IsGameOver()...
    2 references
    private bool isWin()...
    1 reference
    public string NextMove(int x, int y)...
}
}
```

Every cell of grid[,] array needs to be set to null, i.e., it holds no shapes initially.

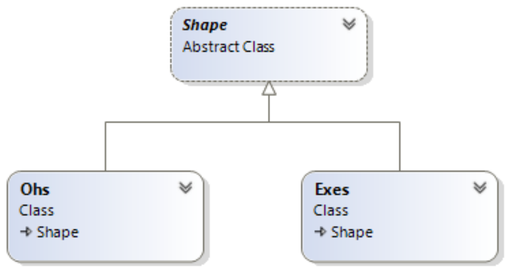
Reset turn indicator, move counter, and call resetGrid(), make O's the first trun.

Game is over if 9 moves have been recorded OR there is a win.

Checks if three shapes in any row, column, or diagonally on the grid.

Figure out where the mouse click landed on the grid and if there is no shape there yet, record the shape in the grid array, tell the shape to draw itself.

You will also want to create an abstract Shape class with derived classes for the X character and the O character.

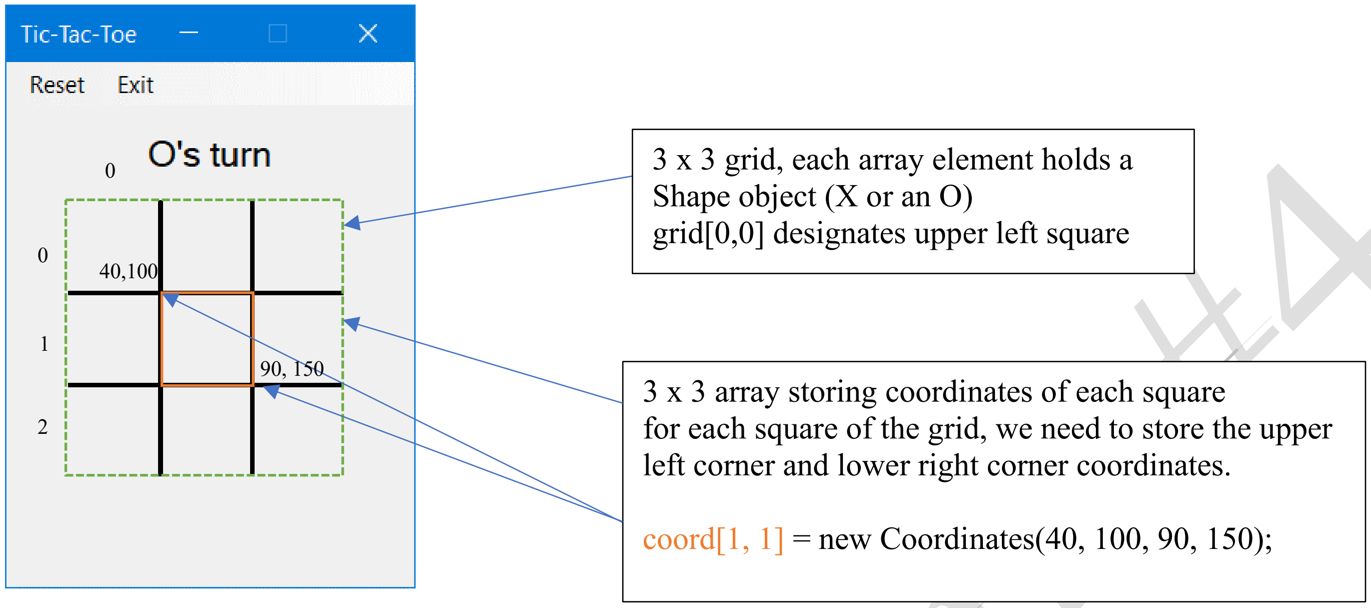


The shape classes will need access to the form's graphic context, so I suggest that you define a public static Graphics reference in the Shape class. It should be initialized by Form1_Load().

For the mouse up event, you will want to send the coordinates to a Game function that makes the next move. Ignore any mouse click that doesn't fall within one of the nine squares of the grid, and ignore any mouse click involving the right mouse button. The function that makes the next move returns a string response, and that string should be put into the label on the form. Some possible messages are:

```
O's turn
X's turn
Square already occupied--choose another
It's a tie!
X's win!
O's win!
```

Here are some hints:



3 x 3 grid, each array element holds a Shape object (X or an O)
grid[0,0] designates upper left square

3 x 3 array storing coordinates of each square for each square of the grid, we need to store the upper left corner and lower right corner coordinates.

coord[1, 1] = new Coordinates(40, 100, 90, 150);

You will want to define data structures containing all of the objects drawn on the form. They will be redrawn as necessary. For example, the Game class will probably have a 3x3 array of type Shape to hold all of the X's and O's on the form.

```
        private Shape[,] grid;
```

You will also want to save each square's coordinates in a 3x3 array so that these are readily available for calls to Draw for the X's and O's. This array of coordinates can also be used to determine which square a mouse click falls in. You might want to use the system class Rectangle to save these coordinates. I created a Coordinates class to represent the two points in the plane (i.e., upper left and lower right corners of my grid's square).

```
        private Coordinates[,] coord;
```

The form Paint event handler redraws the grid. The form Mouse click event record x,y of the click, calls IsGameOver() and if not, calls NextMove(x, y);

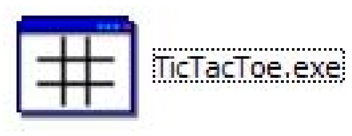
The abstract class called Shape will have an abstract method called Draw. The Draw function should take four arguments, representing upper-left and lower-right rectangle coordinates. If you prefer, Draw could accept only one argument of type Rectangle (a .NET class). Each derived class must implement its own Draw.

```
    public abstract class Shape
    {
        public string ItsShape { get; set; } // auto-property
        public static Graphics g;

        public Shape()
        {
            ItsShape = "";
        }
        public abstract void Draw(int x1, int x2, int y1, int y2);
    }

    public class Exes : Shape
    {
        public Exes()
        {
            ItsShape = "X";
        }
        public override void Draw(int x1, int x2, int y1, int y2){/* implement */}
    }

    public class Ohs : Shape
    {
        public Ohs()
        {
            ItsShape = "O";
        }
        public override void Draw(int x1, int x2, int y1, int y2){/* implement */}
    }
```

The Reset menu command should return the playing grid to its initial state. It should also change the label to say "X's turn". You might need to call the Invalidate() function from the event handler for this menu item to force the screen to be redrawn.

Once someone has won the Game, ignore all other mouse clicks on the form until the Game is reset. Do not allow two clicks into the same grid square, do not increment moves when that happens. Don't forget to recognize a tie game and display an appropriate message. It is sufficient to keep track of how many moves have been made and declare a "tie game" after nine moves (although you can get fancier than this if you like).

For this assignment, make a custom icon for your application by editing the App.ico file. Your icon might look something like this, but feel free to be more creative:



Code Documentation

Each source file should contain a documentation header with a description, author name, and class information (CS3160, Fall/Spring 20YY). Each function should have a documentation header with a minimally function name, description, parameters, return value. Use proper program style at all times, which means indentation, alignment, and whitespace. Utilize self-documenting code which means use mnemonic, i.e., meaningful variable/function/namespace/class names, etc.

What to turn in?

Submit a compressed (.zip) solution folder via Canvas.
Make sure that all necessary files to compile/link/execute your projects are provided in your solution folder.
Here are some files that may be required:

1. All required C# source files (.cs)
2. makefile (if the solution requires it)
3. The entire Visual Studio 2019 solution folder (if the solution requires it)