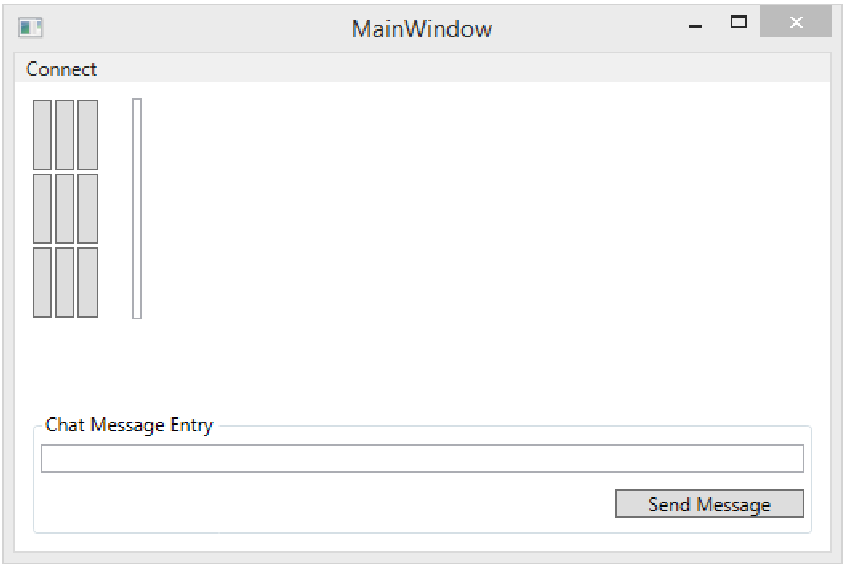
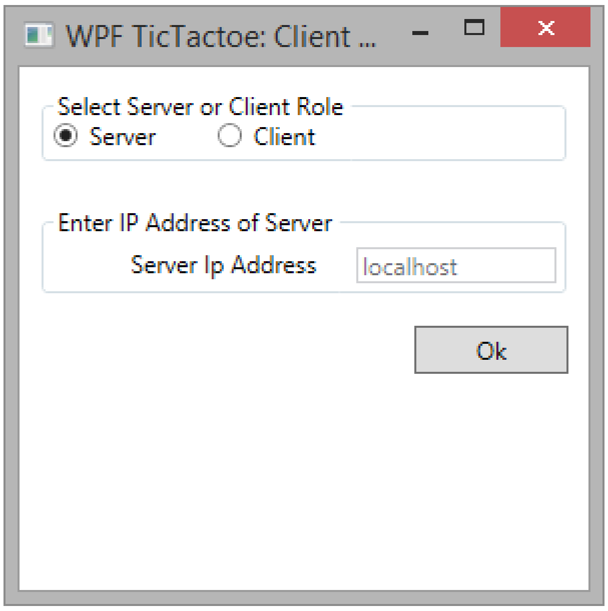
CSC 355 PROJECT 6
NETWORKED TIC TAC TOE WITH CHAT CAPABILITY

PROFESSOR GODFREY C. MUGANDA

Write a networked application that allows two players to play a game of Tic Tac Toe while exchanging chat messages. The main user interface should look like this when the game is first started.



At this point, a user clicking on the `Connect` menu item will bring up the following dialog box:


WPF TicTactoe: Client ...

Select Server or Client Role

(● Server) ○ Client

Enter IP Address of Server

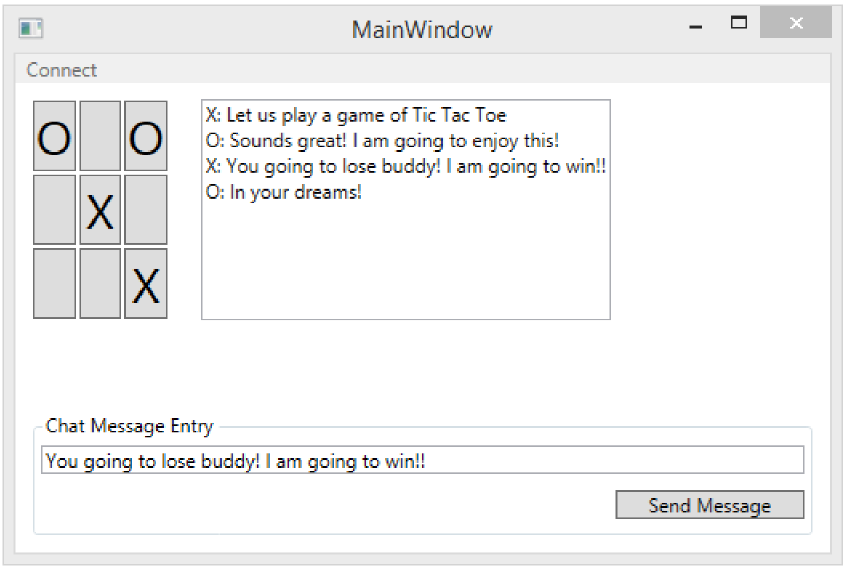
Server Ip Address: localhost

Ok

This modal dialog box allows the user to select one of two roles: server or client. The user selects one of the two radio buttons. If the client role is selected, the textbox for entering the IP address of the server is enabled; otherwise, the IP address text box is disabled. A user selecting the client role must also enter the IP address of the server. (It is assumed that the server is listening on port 50001). Once all the required information has been entered, the user clicks the Ok button to connect to the remote peer.

Once a connection has been established the game proceeds. A user can click on a cell on the Tic Tac Toe "board" to place a marker (X or O) when it is his or her turn. The application will display the local player's marker on the clicked cell and send notification of the move to the remote side, where the the move will be recorded in the corresponding cell. In addition, a user may type a chat message in the chat message text box at the bottom and click the `Send Message` button. This will place a copy of the typed message in the *chat history text box* at the top right and also send a copy of the message to the remote player, where it will also be recorded in the chat history textbox.

Here is a screen shot of the game in progress:

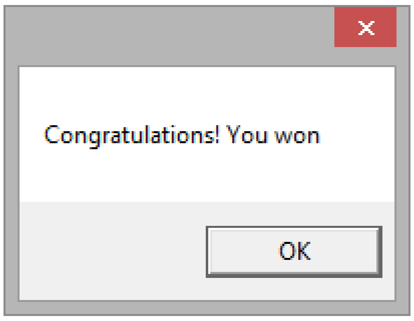


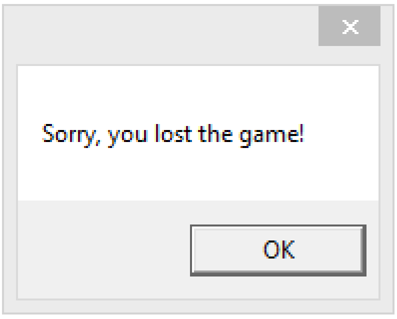
X: Let us play a game of Tic Tac Toe
O: Sounds great! I am going to enjoy this!
X: You going to lose buddy! I am going to win!!
O: In your dreams!

You going to lose buddy! I am going to win!!

The game should work as a user would expect: For example, the game should ignore attempts by a user to play out of turn, or to move to cell that is already marked.

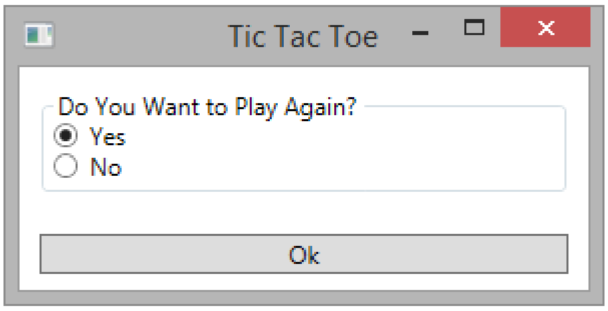
A game can end in a tie, or in a win for one of the player. If the game ends in a tie, a message box should pop up indicating the that the game is tied, otherwise, a message box should pop informing the local player that they have won or lost the game:


Congratulations! You won


Sorry, you lost the game!

Let us call these dialogs the game-over dialogs. Once the game-over dialogs are dismissed, the game pops up another dialog box asking the user if he / she would like to play again:

Upon the dismissal of this dialog, the game sends a message to the remote side stating whether or not the local player wants to play again. The game then processes the play-again message from the remote side. If both the local and remote players want to play again, the game resets and a new game begins. Otherwise, if at least one of the players does not want to play again, the game exits. If the local player wants to play again but the remote player does not want to play, a message box should be displayed to indicate the the remote player does not wish to play again, and the game will terminate when that dialog box is dismissed.



1. Additional Conditions

The server should have the X marker, while the client has the O marker. The server gets to go first for the first game. Thereafter, the players alternate as who gets to go first in subsequent games.

Use the following protocol messages:

(1) `move` *row col*: Here *row* and *col* are 32 bit integers. This message is sent to indicate that the local player has put their marker on the cell with the given row and column.
(2) `chat` *message*: Here *message* is a string to be sent to the other side as part of the chat function.
(3) `playagain` *no* : This message is sent to indicate that the local player does not with to play another game.
(4) `playagain` *yes*: This message is sent to indicate that the local player wants to play another game.

You must create a separate thread that stays in a perpetual loop, reading the network connection whenever a message arrives, decoding the protocol command received, and applying the command as needed. This means that a `move` protocol command will result in the remote player's marker being placed in the specified cell in the local GUI, and *chat* protocol commands will result in a copy of the received message being displayed in the local chat history text box.

Note that the network reading thread cannot directly access the GUI, so it must arrange to have the GUI update code run on the GUI Dispatcher thread.

2. Hints

You might find it useful to have the following variables to track the state of the game:

```
const int port = 50001;
PlayerId localPlayerId;
PlayerId remotePlayerId;
 // Initially, X, the server, always goes first
PlayerId playerTurn = PlayerId.X;
int numberTurnsRemaining = 9;
TcpClient tcpClient;
BinaryWriter netWriter;
BinaryReader netReader;
bool gameOver = false;
//  Message sent: Local player want to play again
```

```
        bool localPlayAgain = false;
        // Message Recieved: Remote player wants to play gain
        bool remotePlayAgain = false;
        int numberOfGamesPlayed = 0;

        TTTCell[,] board = new TTTCell[3, 3];
```

You may use the following class to represent an cell on the Tic Tac Toe board:

```
namespace WPFTicTacToe
{
    enum PlayerId { X, O, None };

    class TTTCell : Button
    {
      public readonly int row;
      public readonly int col;
      private PlayerId owner;
      public TTTCell(int r, int c)
      {
        row = r;
        col = c;
        owner = PlayerId.None;
        Content = (" ");
        this.FontSize = 30;
        this.Margin = new System.Windows.Thickness(20, 20, 20, 20);
      }

      public PlayerId getOwner() { return owner; }

      public void setOwner(PlayerId id)
      {
        owner = id;
        switch(owner)
        {
            case PlayerId.X : Content = "X"; break;
            case PlayerId.O: Content = "O"; break;
            case PlayerId.None: Content = " "; break;
        }
      }
    }
}
```

For the `ChatHistoryTextBox`, use a `TextBox` with the `AcceptsReturn` property set to true. You should also set its `VerticalScrollBarVisibility` property to `Auto` on this text box to make a scrollbar appear when the text box fills up.

Finally, you may want to look up two-dimensional array in C# online so you know how to represent the board.

This is due Saturday at the end of week 10.