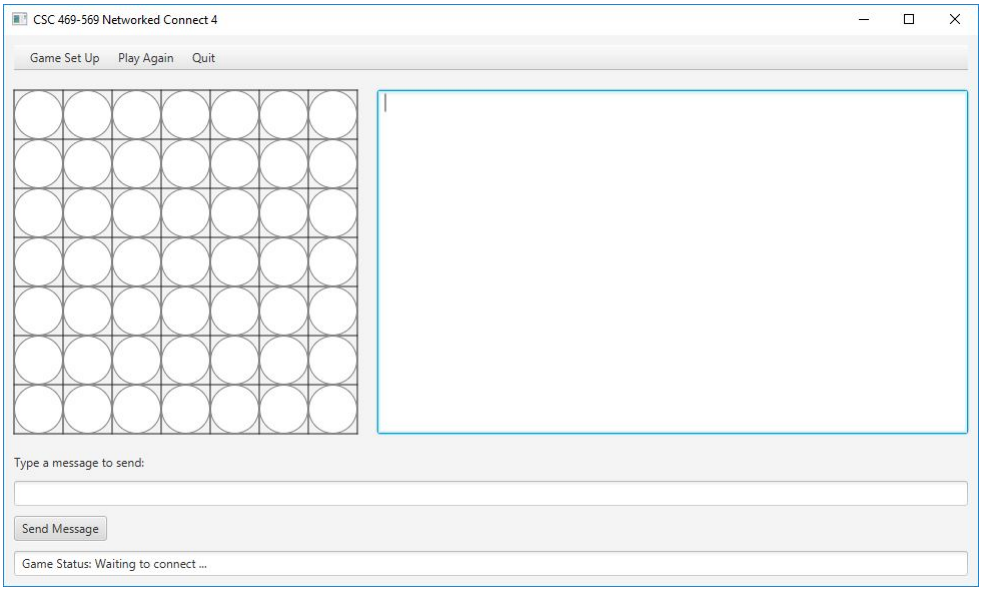
CSC 469/569 JAVA FX CONNECT 4 ASSIGNMENT
DUE MONDAY WEEK 6

GODFREY MUGANDA

Write a networked version of the Connect 4 game that allow players to exchange chat messages. The user interface should look as shown in the image below.



As explained in class, the user interface consists of a menubar with 3 menus, a *gameboard*, an chat history text areas, and a place at the bottom of the window to type in and send chat messages. There is also a status bar at the bottom.

1. Menus

The application has 3 menus on the menu bar

(1) Game Set Up
(2) Play Again
(3) Quit

The *Game Set Up* menu has 2 menu items that allow a player to select whether they will act as a server or a client during the initial set up phase:

(1) Server Role : the application sets up a server socket and waits for a connection request from a client. The serve must start first, and must run at port 50002.
(2) Client Role : the application pops up a dialog to ask the user to enter the IP address of a server. creates a regular socket and uses it to connect to a waiting server.

Initially, the only the *Game Set Up* menu is active. The *Game Set Up* menu becomes disabled as soon as network connections have been made. The *Quit* menu is enabled after the first game starts, and the *Play Again* menu is enabled only between games. This means that the *Play Again* menu is enabled as soon as a game ends, and becomes disabled when a new game starts.

2. How the Game Works

The player with the server role goes first when playing the first game. After the first game, players alternate who gets to go first.

The two players should use different colors for the markers. For example, the server might use blue markers while the client uses red.

3. Status Bar Messages

Following messages should be displayed on the status bar at the appropriate times:

(1) *Waiting for a network connection:* Displayed while waiting for the players to establish a network connection.
(2) *Make a move:* Displayed when it is the local player's turn to make a move.
(3) *Please wait for your turn:* Displayed when it is the remote player's turn to make a move.
(4) *Game over:* Displayed when a game is over, but before a new game starts.

4. End of a Game

An alert box announcing the result of the game will pop up as soon as a game ends. The alert box will display one of the following messages

(1) Congratulations: You won!
(2) So sorry you lost! Better luck next time.

After the user dismisses the alert box, he or she may use the *Play Again* menu to request another round, or the *Quit* menu to exit the game.

5. Project Design

An overall design approach was discussed in class. Avoid a design that puts everything in a single file. Set up the project files as follows:

(1) *JavaFXConnect4.java* : This file has the code that builds the user interface, instantiates the event handlers, and attaches the handlers to the appropriate components.
(2) *MenuItemHandlers.java* : This file contains the classes that implement event handlers for the menu items.
(3) *Globals.java* : This file contains static fields for all the variables that need to be accessed by code in separate files. These include variables that keep track of the state of the game, as well as references to certain user interface components.
(4) *Connect4Events.java* : This file contains classes that implement event handlers for all non-menu user interface components.

(5) *NetworkReader.java* : This file contains the classes that implement the *Runnable* interface and are used to read network input and post work assignments to the GUI application thread.

6. Protocol Messages

The game should use the following protocol messages:

(1) *chat* [*message*] where *message* represents a string that is the content of a chat message. Upon receiving this protocol command, the application updates the chat history by appending a line prefixed by *Remote:*.
(2) *play* [*row*] [*col*] where *row* and *col* are integers. Upon receiving this command, the application updates the game board by placing the remote player's marker in the given row and column of the game board. The application should then update game state and status bars so the local player knows if the game is over, or if it is now his/her turn to play. If the game is over, the application pops up an appropriate alert to announce the outcome of the game to the local user.
(3) *quit*. Upon receiving this command, the application pops up an alert to inform the local user that the remote player has quit the game. The application should close network connections and exit when the alert box is dismissed.
(4) *playagain*. Upon receiving this command, the application pops up an alert to inform the local user that the remote player wants to play another round. If the local player also wants another round, the application sends a *playagain* message to the remote player and resets the game. If the local player declines, the application sends a *quit* message to the remote player, closes network connections, and exit.

7. Event Handling

As you know, the majority of the work in a GUI program is accomplished in the event handlers. We have already discussed how to handle game set up menu events. Other events should be handled as follows:

(1) *Mouse Enter* on a `Circle` node in the top row of the game board: The circle fills with the local player's color.
(2) *Mouse Exit* on a `Circle` node in the top row of the game board: The circle fills with the color WHITE.
(3) *Mouse Click* on a `Circle` node in column c in the top row of the game board: The click is ignored if it is not the local player's turn, or there is no more room in the column. Otherwise, the application fills the circle in row r and column c with the local player's color, where r is the largest row the column that is still filled with WHITE (this will be the farthest row down); sends a
    play r c
    message to the remote side; and updates the game state and status bar.
(4) *Click* on the *SendMessage* button: The application retrieves the text from the message text field, clears the contents of that text field, and appends the text to the chat history with a prefix of "Me:". The application then sends a
    chat m

message to the remote side, where m is the text retrieved from the chat input text field.

(5) *Click* on the *Play Again* menu item : The application sends a
playagain
message to the remote side.

(6) *Click* on the *Quit* menu item : The application sends a
quit
message to the remote side.