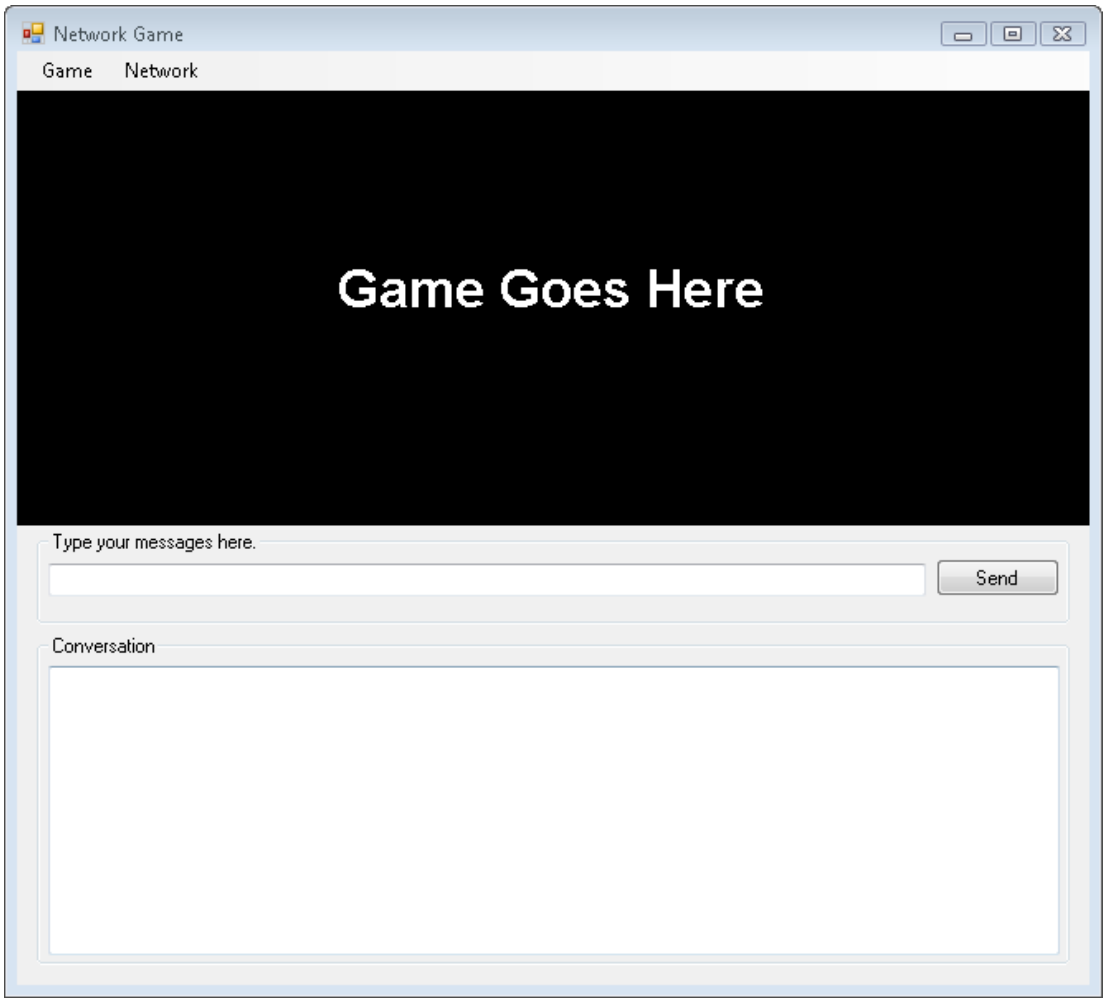
# Assignment – Asynchronous Network Client

For this assignment, you will be taking your learning from your Synchronous Chat console application and upgrading it into an Asynchronous library that will be used within the framework of a Windows Forms Application.

The Windows application will be the skeleton of a fictional, networked game. The game interface will look similar to the following:



The area marked “Game Goes Here” is where the fictional game is played. The area below the game play area is where users can chat with another player of the game. For the purposes of this assignment you can ignore the game play area. This assignment deals with the chat ability.

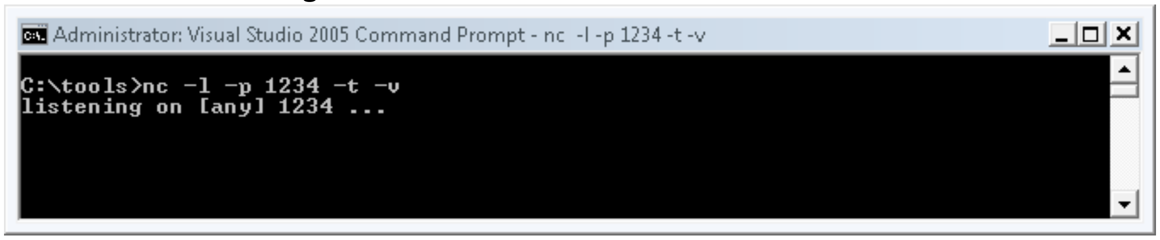
The chat function will work similarly to Assignment 1, except that this is a Windows application instead of a console application. Also, because of time constraints, you are going to focus on the **client** portion of the chat functionality. The **server** will be simulated using either your console server from Assignment #1 or the open-source utility “NetCat” (http://nc110.sourceforge.net/).

Note: For simplicity sake, you can again assume that both the client and server are running on the same machine.

The basic operation of the program is as follows…

1. The server application is started and begins waiting for clients to connect.

**Server simulated using NetCat**

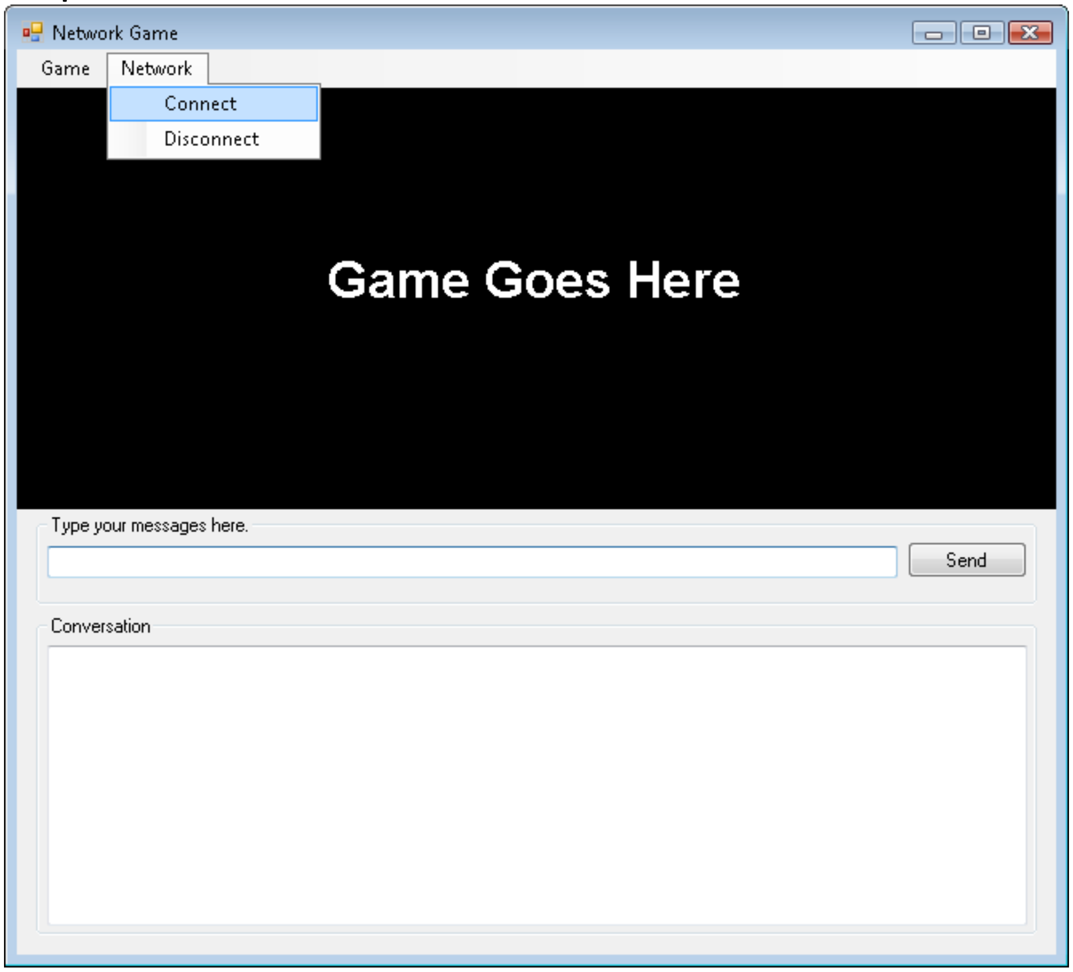


Administrator: Visual Studio 2005 Command Prompt - nc -l -p 1234 -t -v

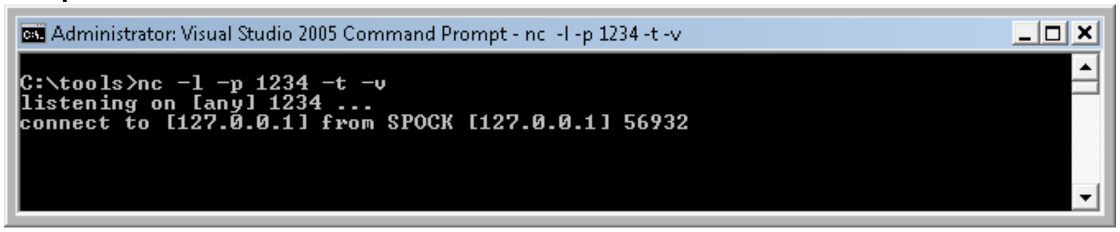
```
C:\tools>nc -l -p 1234 -t -v
listening on [any] 1234 ...
```

2. The Windows client application is started.
3. The user then connects to the server using a menu selection:

**Sample network connection selection**



**Sample successful client connection**

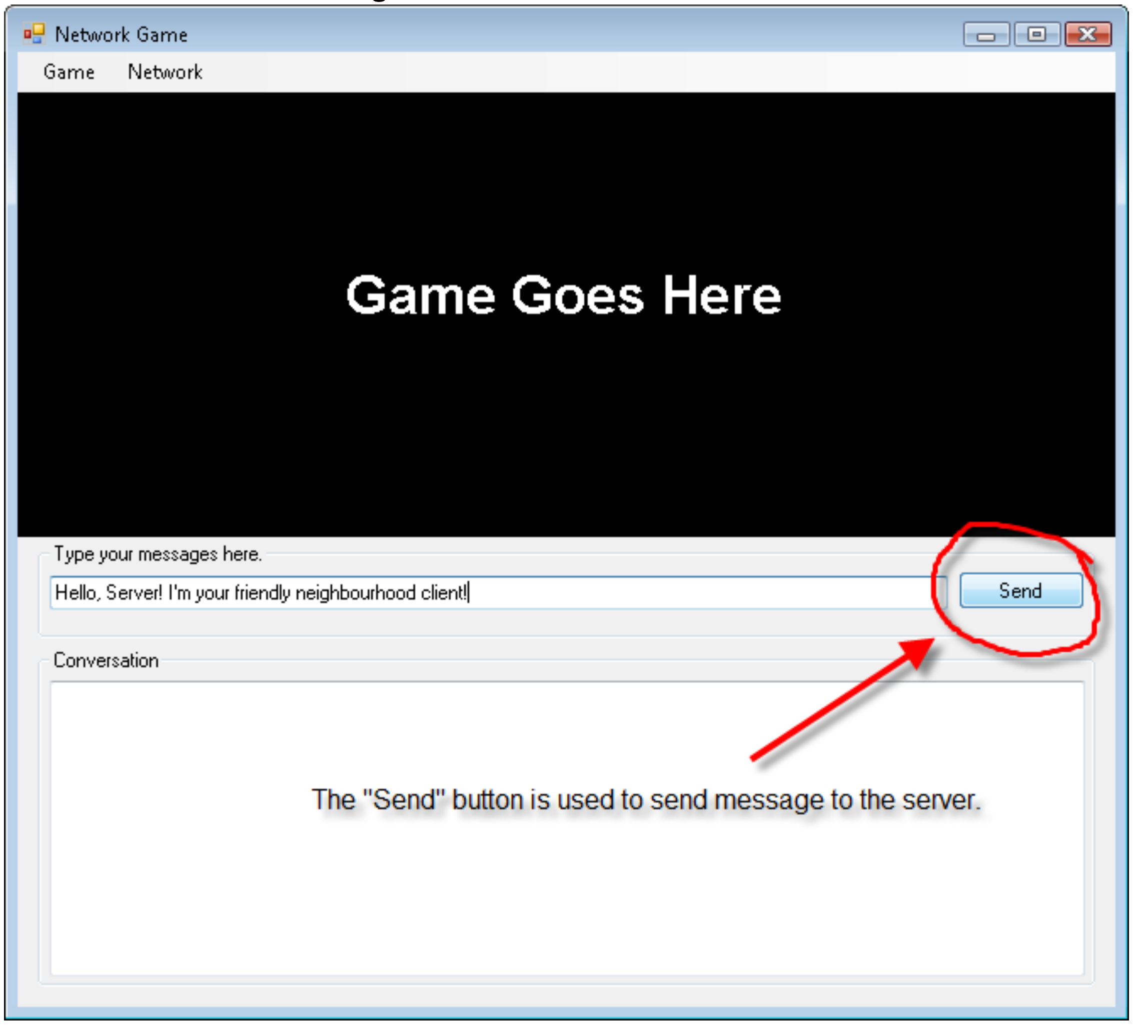


Administrator: Visual Studio 2005 Command Prompt - nc -l -p 1234 -t -v

```
C:\tools>nc -l -p 1234 -t -v
listening on [any] 1234 ...
connect to [127.0.0.1] from SPOCK [127.0.0.1] 56932
```

4. Once connected, either the server or the client may begin sending messages.

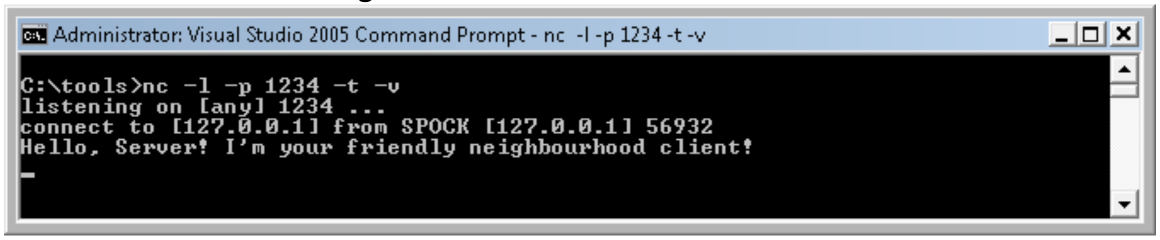
**View of client before message from client has been sent.**



Hello, Server! I'm your friendly neighbourhood client!

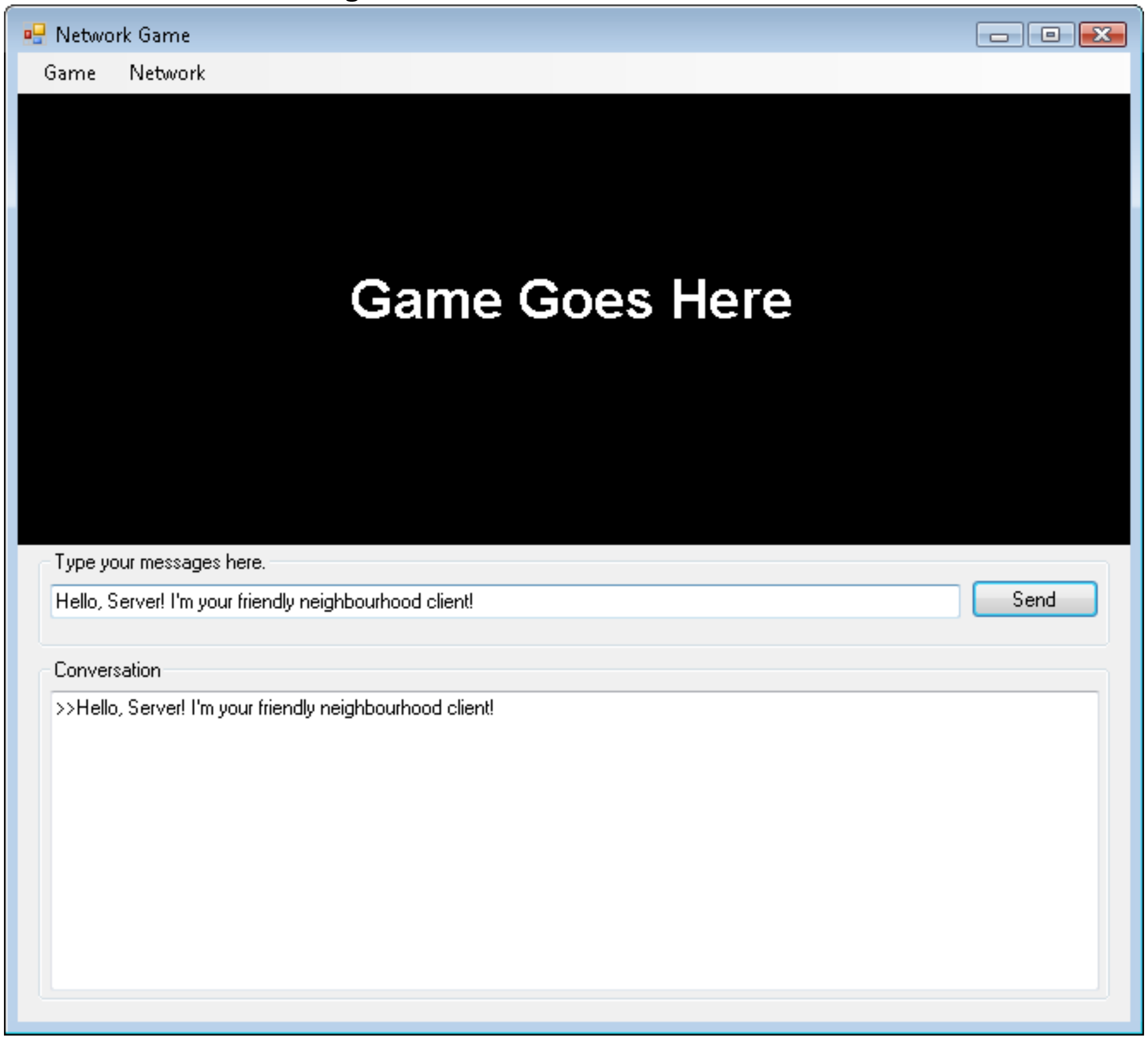
The "Send" button is used to send message to the server.

**View of server after message from client has been sent.**


Administrator: Visual Studio 2005 Command Prompt - nc -l -p 1234 -t -v

```
C:\tools>nc -l -p 1234 -t -v
listening on [any] 1234 ...
connect to [127.0.0.1] from SPOCK [127.0.0.1] 56932
Hello, Server! I'm your friendly neighbourhood client!
_
```

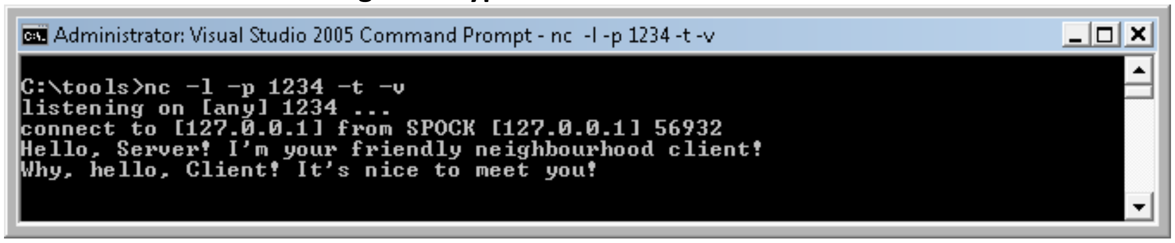
**View of client after message from client has been sent.**



Hello, Server! I'm your friendly neighbourhood client!

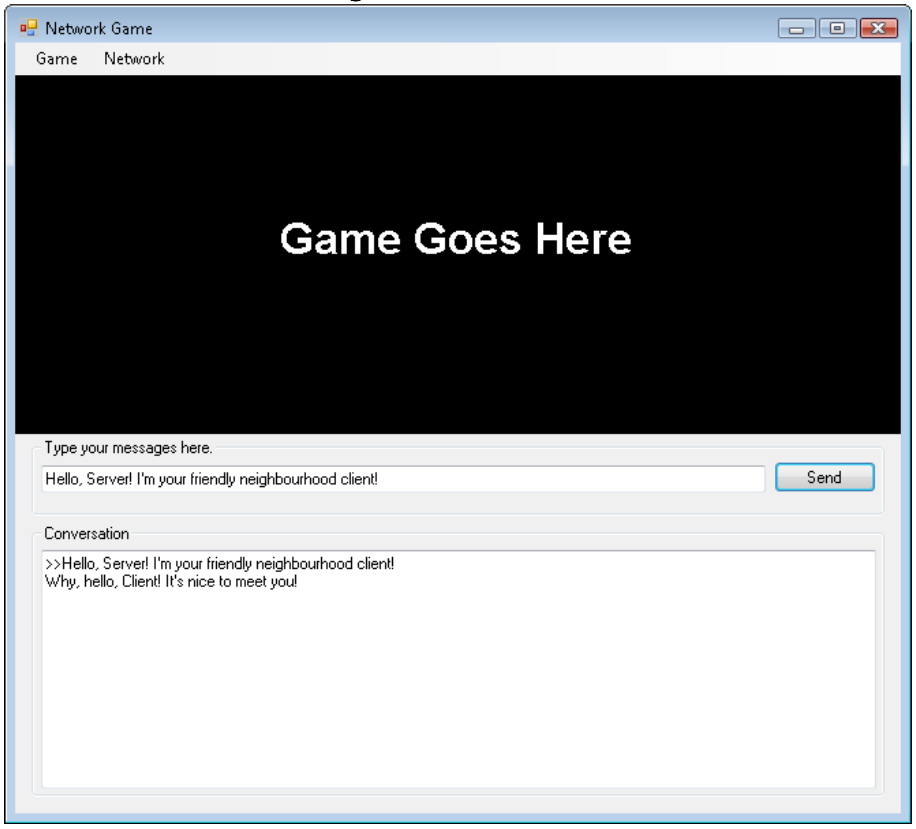
>>Hello, Server! I'm your friendly neighbourhood client!

**View of server after message was typed and sent from the server**


Administrator: Visual Studio 2005 Command Prompt - nc -l -p 1234 -t -v

```
C:\tools>nc -l -p 1234 -t -v
listening on [any] 1234 ...
connect to [127.0.0.1] from SPOCK [127.0.0.1] 56932
Hello, Server! I'm your friendly neighbourhood client!
Why, hello, Client! It's nice to meet you!
```

**View of client after message was received from the server**

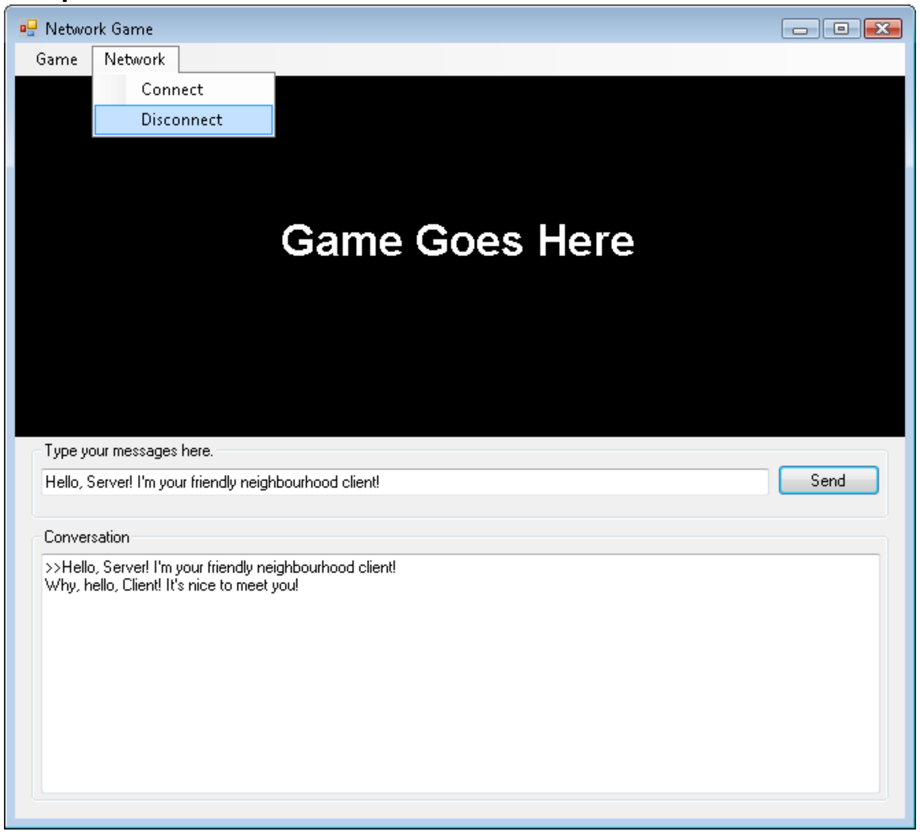


Hello, Server! I'm your friendly neighbourhood client!

```
>>Hello, Server! I'm your friendly neighbourhood client!
Why, hello, Client! It's nice to meet you!
```

5. After the game is completed and the user wishes to disconnect from the server, the user clicks another menu selection.

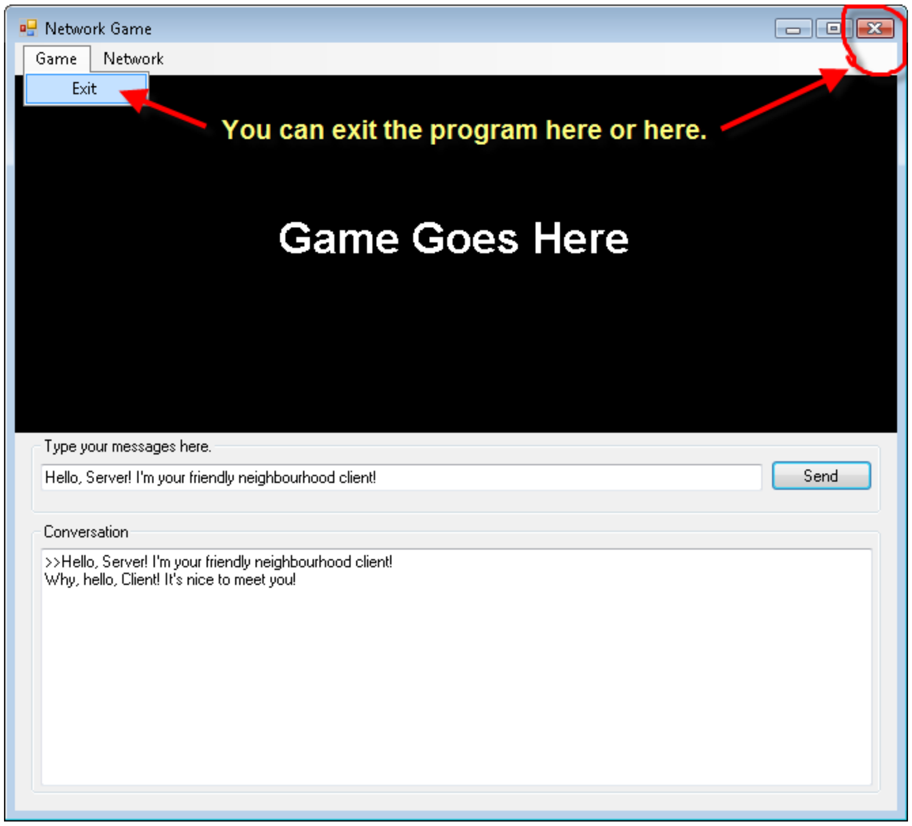
**Sample network disconnect selection**



Hello, Server! I'm your friendly neighbourhood client!

```
>>Hello, Server! I'm your friendly neighbourhood client!
Why, hello, Client! It's nice to meet you!
```

6. And finally the application can be closed using either the standard “X” button in the upper right corner, or another menu selection.



Hello, Server! I'm your friendly neighbourhood client!

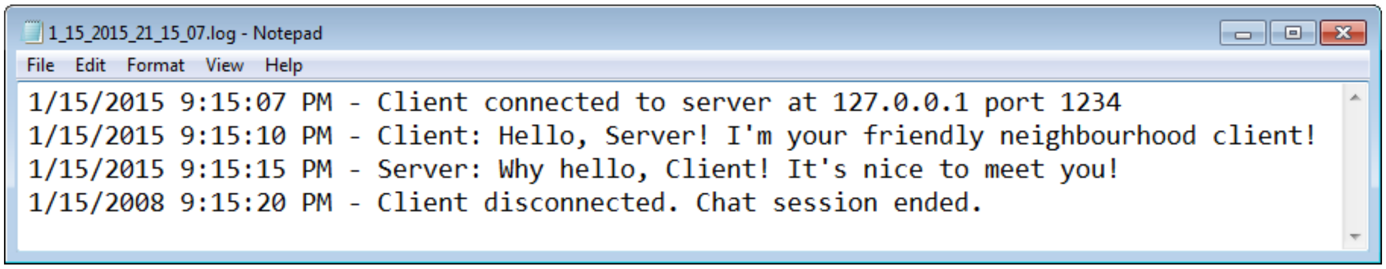
>>Hello, Server! I'm your friendly neighbourhood client!
Why, hello, Client! It's nice to meet you!

## Chat Session Logging

Your client application will also be required to record entire chat sessions by logging them to a text file. You will create a separate class library as part of your overall solution whose sole responsibility will be to log sessions to a text-based log file.

For each chat session, a new log file should be created and all contents of the chat should be written to the log file as the chat session progresses.

**Example of log file based on chat screenshots above**



```
1/15/2015 9:15:07 PM - Client connected to server at 127.0.0.1 port 1234
1/15/2015 9:15:10 PM - Client: Hello, Server! I'm your friendly neighbourhood client!
1/15/2015 9:15:15 PM - Server: Why hello, Client! It's nice to meet you!
1/15/2008 9:15:20 PM - Client disconnected. Chat session ended.
```