UNIVERSITY OF
BIRMINGHAM

Introductory Module

04 41480
Object Oriented Programming

Assignment 2009

Dr M. Spann

1. Aims and Objectives

The aim of this programming exercise is to design a system enabling a simple card game, gin rummy, to be played across a network. The assignment will involve the following aspects of C# programming that we have covered in the course.

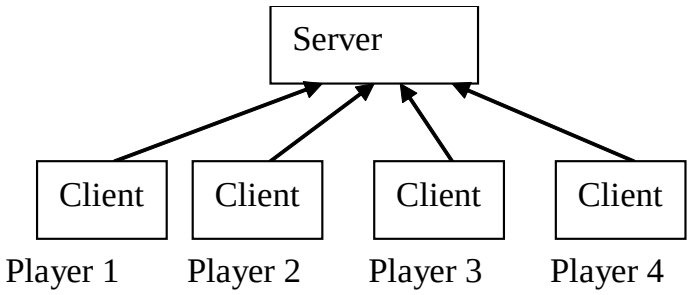
- Object oriented programming
- Graphical user interfaces
- Object serialization
- Multi-threading

In addition, the assignment will involve the development of a client/server system for handling the communications necessary to support the online game. This is covered in the appendix although it's very similar to code we looked at in the Files and Streams lecture.

2. Practical work

The assignment involves creating a system which allows 4 players to play the card game gin rummy across a network. The simplified rules of gin rummy are explained in Appendix IV.

One of the main tasks of this assignment is to develop a multi-threaded server, and associated client. Networking in C# is similar to that in Java and is about the easiest of any programming language. Essentially it involves reading or writing a Socket which you can regard as a network end point. A simple introduction is given in Appendix 1.



3. Design and Programming Hints

This is a demanding programming exercise and you should design your system in separate stages with each stage undergoing thorough testing. Note, it is expected that you include details of all testing in your report. My suggestion is as follows:

1. Design a system that includes all of the necessary classes to support a simple console based game of gin rummy. Package this up as a *dll* as you will need to make use of these classes in your final system. Include functionality in your system

allowing a game to be played *interactively* (in other words the user plays the hand) or *autonomously* (the computer plays the hand). In either case, the decision about whether a hand is a winning hand is determined by the computer. Thus you will have to design some simple algorithms to support this functionality. Use pseudo-code to express algorithm design in your report.

2. Design a graphical user interface to support the playing of the game. The GUI should enable each player's hand to be displayed and each player to play either interactively (which will involve user interaction with the playing card images to select cards to exchange) or automatically (which will involve a simple mouse click instructing the computer to play the hand). Obviously you will make extensive use of visual programming techniques to create your GUI.

3. Design the final system involving a server program which enables clients to connect, plays the card game and communicates the 'plays' to its clients. The client program will handle the update of its graphical user interface according to the information it receives from the server. A complication is that a player can play interactively so that a particular play is generated locally at a client. In this case the relevant information must be transmitted back to the server and then 'broadcast' out to all clients so that they can each update their GUIs accordingly. I wouldn't recommend you attempt a peer to peer communication strategy as it's considerably more complex.

   Another issue that you must consider in the design of the final system is the amount of network traffic produced in the playing of a game and at what stage in the game information can be passed from server to client. Imagine that this online card game is being played by registered players for money. Player A should not be able to cheat and be able to gain access to the hand held by player B either by examining object values in player A's program (for example through the use of a debugger) or by 'sniffing' network traffic. As a simple example, when the cards are dealt, only the hand of each player is sent to its appropriate client and not the hand of the other 3 players. Also the draw deck should only be held at the server and not at each client. The draw deck's top card should only be passed to the appropriate player when required to be shown face up and is then, as in the actual card game, visible to all players.

4. Resources

There are a set of playing card images which you can download from my web site playing card images. The image file names correspond in an obvious way to the card face value and suit. Also included are a couple of images of the card's back. All of the images are in gif format and to read the files and store the image into an *Image* object, use the static method *Image.FromFile(string filename)*.

5. Assessment

This coursework represents all of the assessment for this module. The assessment will be based on a submitted formal report as well as my assessment of your program's functionality. You will be expected to give a demonstration of your program to me and answer any questions about your program. Each student will be allocated a 5 minute slot to do this.

Please submit your program **written using VS2008 (.NET framework 3)** on CD to accompany your report. Please include all of the solution files under a single solution directory Make sure your CD has your name/ID on it in case it gets separated from your report. I randomly check submitted code using anti-plagiarism software (see below). **Your program must run on the School's networked Visual Studio 2008.** Even if you develop your application in Visual Express, it is your responsibility to check compatibility with the fully installed Visual Studio and please be aware we have had compatibility issues in the past.

The assessment form that I use is in appendix V so this should give you an idea of the criteria I will use in marking your report. You should be aiming for a report length of around 15 to 20 pages excluding appendices. I am happy for you to include your code listing in an appendix but it is not obligatory. I do not expect you to use formal design notation (such as UML) but you can if you are familiar with it. Use pseudo-code to explain algorithm implementation (and not flow charts!) **and do not include explicit code snippets in your main report**.

Finally, I am sure you are aware there is a lot of published code on the internet for just about every application imaginable**. If you are going to use downloaded code for any part of this exercise, make sure you attribute it in your report (referencing the URL is sufficient)**. Obviously your mark will reflect the amount of original code in your program but you will not be penalised for using small amounts of attributed downloaded code. **If you use code from the internet (or code from a colleague) without an adequate reference in the source text, this will count as plagiarism. Any significant plagiarism will result in a zero mark for the exercise. Also, if you submit the same or similar code to a colleague, you will both receive a zero mark.**

Key dates

Report deadline: **Monday 14th December**. Please hand in to the General Office by 12 noon.

Program demonstration: **Monday 14th December**. Time and venue tbd.

Appendix I

Client/Server Networking

Questions and Answers

1) What is a client… and what is a server?

Server: Sits around waiting for a client to connect.
Client: Connects to a server.

2) How does the client find the server?

Every server has an IP address (every PC in the lab has a different one), and a port number. The client must know both of these in order to find the server.

3) So what is my IP address?

Type 'ipconfig' in a command window ('DOS prompt') and it will tell you. Alternatively if your client and server are **on the same machine** you can use the IP address 127.0.0.1 which means 'on this machine'.

4) What port number should I use?

Any number you like above 1023 and below 65536. Just make sure that your client and server are both using the same port. Also be aware that you cannot have two servers listening on the same port.

5) What happens after I connect?

After you connect you use streams to send and receive messages between the client and the server.

6) So how do I write this in C#?

Have a look at this (client and server) code in the next two appendices.

7) What about if there are multiple clients?

Good question! The above code will only work for a single client - if you read it you should be able to work out why. You will need a *multi-threaded server* to allow multiple simultaneous connections (because you have multiple clients) - you will need to work out how to do this. Also you will need a multi-threaded client so that code which 'talks' to the server runs in a separate thread.

8) Exactly what is the sequence of events in the clients connecting to the server?

It's important to be clear about which bit is doing what, and when. Essentially you start your server and multiple clients (each player runs a client). The clients then all connect to the server, and the game starts. It might be helpful to consider a

typical sequence of events. We are assuming each client supports a GUI which has a 'Connect' button, pressed when the client is ready to connect to the server.

Connecting phase

1) Server is started on PC1
2) Server waits for connections
3) Someone starts a client program on PC2
4) User on PC2 enters the IP address of the server (otherwise the client won't know where the server is)
5) User on PC2 pressed 'Connect' button
6) Client on PC2 tries to connect to server on PC1
7) Server on PC1 gets the connection from PC2, starts a new thread (it's a multi-threading server) and goes back to listening for incoming connections
8) Someone starts a client program on PC3, enters the IP address and pressed 'Connect'
9) Server on PC1 gets the connection from PC3, starts a new thread and goes back to listening for incoming connections
10) When the server has enough players, the game can start.

Obviously, once all 4 clients have connected, the server will then enter its 'Game phase' and the game can then commence. The sequence is exactly the same if all clients run on the same machine as the server.

9) What data is actually communicated?

The server could send a string indicating that the data being sent to the client was a particular card drawn from the top of the draw deck. A 7 of hearts could be sent as the string "D7H" for example. The 'D' could be the code indicating it's the drawdeck top card. As another example, the client could send a string back to the server indicating that its corresponding player has selected the card from the top of the discard deck and exchanged it for a particular card in his hand. The server would then have to communicate this to all other clients so that they could update their GUI's accordingly.

10) Is this particularly object oriented?

No! Obviously you could design a protocol which involves simply passing messages as strings between the client and server programs and each program interpreting these strings in terms of the current status of the game. However, I would recommend you make use of the immense power of object serialization (as discussed in the Files and Streams lecture) to pass serialized objects between the client and server. Obviously these objects could be individual playing cards, an array of playing cards representing a player's hand or an object representing the current 'play' (which cards to exchange and where they come from). It's pretty easy to adapt the client and server programs shown in appendices II and III so that objects and not strings are communicated (although, as a special case, a string is an object!)

Use the *TcpClient.getStream()* method to return a *NetworkStream* object *ns*. Create a *BinaryFormatter* object *bf* and call *bf .Serialize(ns, obj)* which passes the serialized object *obj* (of class *Object)* into the network stream. It's also possible to do this with xml and SOAP serializers for more specialized applications but using the binary serializer is sufficient for this application. At the reading end, use the *Deserialize()* method in the same way. This technique still allows simple strings to be passed as objects. The type of the object read using the simple *if (inputObject is Type)* test will determine exactly what object has been sent and the use of that object at the receiving end.

Appendix II. A simple server

```
using System;
using System.Net;
using System.Net.Sockets;
using System.IO;

namespace TestServer
{
    class MyTcpListener
    {
        public static void Main()
        {
            TcpListener server = null;
            try
            {
                // Set the TcpListener on port 13000.
                Int32 port = 10;
                IPAddress localAddr = IPAddress.Parse("127.0.0.1");

                // TcpListener server = new TcpListener(port);
                server = new TcpListener(localAddr, port);

                // Start listening for client requests.
                server.Start();

                Console.WriteLine("Waiting for client to connect");

                TcpClient socketForClient = server.AcceptTcpClient();
                Console.WriteLine("Connected!");
                if (socketForClient.Connected)
                {
                    Console.WriteLine("Client connected");

                    NetworkStream networkStream =
                              socketForClient.GetStream();

                    System.IO.StreamWriter streamWriter =
                    new System.IO.StreamWriter(networkStream);
                    System.IO.StreamReader streamReader =
                    new System.IO.StreamReader(networkStream);
                    string theString = "Server Message";
                    streamWriter.WriteLine(theString);
                    streamWriter.Flush();
                    theString = streamReader.ReadLine();
                    Console.WriteLine("Server: " + theString);
                    streamReader.Close();
                    networkStream.Close();
                    streamWriter.Close();
                 }
                 socketForClient.Close();
                 Console.WriteLine("Exiting...");
             }
            catch (SocketException e)
            {
                Console.WriteLine("SocketException: {0}", e);
            }

            Console.WriteLine("\nHit enter to continue...");
            Console.Read();
        }
    }
}
```

Appendix III. A simple client

```
using System;
using System.Net;
using System.Net.Sockets;
using System.IO;

namespace TestClient
{
    public class MyTcpClient
    {
        static void Main(string[] args)
        {
            TcpClient socketForServer;
            try
            {
                Int32 port = 10;
                socketForServer = new TcpClient("127.0.0.1", port);
            }
            catch
            {
                Console.WriteLine("Failed to connect to server at
                                  {0}:999", "127.0.0.1");
                return;
            }
            NetworkStream networkStream =
                  socketForServer.GetStream();
            System.IO.StreamReader streamReader = new
                  System.IO.StreamReader(networkStream);
            System.IO.StreamWriter streamWriter = new
                  System.IO.StreamWriter(networkStream);
            try
            {
                string outputString;
                // read the data from the host and display it
                {
                    outputString = streamReader.ReadLine();
                    Console.WriteLine("Client: " + outputString);
                    streamWriter.WriteLine("Client Message");
                    streamWriter.Flush();
                }
            }
            catch
            {
                Console.WriteLine("Exception reading from Server");
            }

            // tidy up
            networkStream.Close();
        }
    }
}
```

Appendix IV

The rules for a simplified game of gin rummy are as follows.

The game is for 4 players. Each player is dealt 7 cards from the pack. The top card from the remaining deck is then drawn and placed face up as the first card in the 'discard pile'. Each player takes it in turns and can either draw the top card in the remaining deck, which is face down, or the top card from the discard pile which is face up. The objective is to arrange your 7 cards into one group of 4 and one group of 3 where a group can either be 3 or 4 cards with the same face values (eg. three 7's or four queens) or a run of cards of the same suit (eg 2,3,4 and 5 of spades). Normally the ace is taken as a one. The first player who gets to this stage has won the game. Obviously the key to the game is to judiciously change a card in your hand with the one drawn from the pack or the discard pile. Also other players can see the cards you take from the discard pile which may give them a clue as to other cards that you may need. You may or may not exchange the drawn card with one from your hand depending on the suitability of the card drawn. In either case, one card is discarded and put on the discard pile (face up for all the other players to see) so each player is always holding 7 cards. If all of the cards from the deck have been drawn and a player has yet to win, the discard pile is re-shuffled and placed face down and the game continues.

Appendix V. Assessment Form

Introductory Module
04 41480
Object Oriented Programming

Report Presentation Cover page Page numbering Contents page Grammar and spelling Section layout Figure labelling and clarity Correct use of references		/10
Program Design Effective use of classes and object interactions Discussion of object oriented issues relating to design Effective use of clear formal or semi formal design diagrams		/20
Program Implementation Code layout including use of comments Effective use of dll's Algorithm efficiency and correctness Minimization of network traffic Effective use of object serialization for client server communication Effective use of multi threading		/20
Program Functionality No, limited, full or extended operation Ease of use of the graphical user interface Useful information displayed during course of game		/30
Testing Use of a systematic approach to sub system and full system testing Testing for unexpected user interactions and inputs Use of verifiable outputs such as screenshots		/10
Conclusions Discussion of possible design and implementation improvements Discussion of how well the program meets the specification and if not, why not Overall summing up of what has been achieved and what has been learnt		/10