MATCHING COLOURS

COMP1564 – SYSTEM PROGRAMMING

Usman Basharat
000874782

Table of Contents

The OWNER of this report	Name: Usman Basharat	Student ID: 000874782
The student I worked with	Name: Yunus Hassan	Student ID: 000880204

Date started: 15/03/2016

The Game we have chosen to implement is: Matching Colours

Date this Game was discussed with the tutor and agreed by them: 06/04/2017

Pages: 13 excluding Code Listing

Introduction

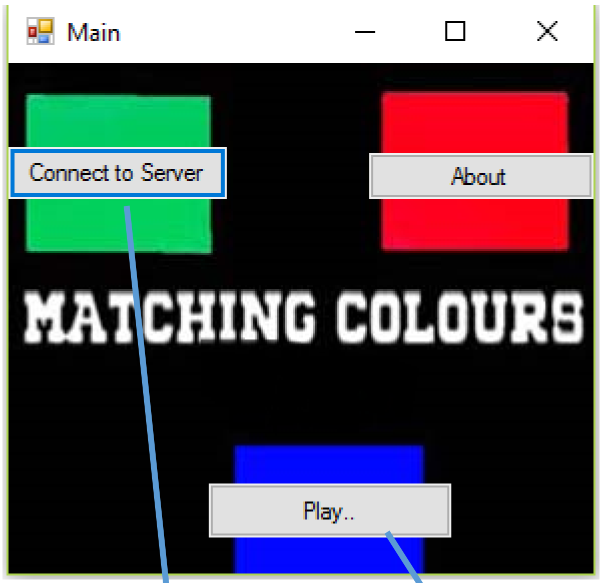
In this report, we worked as a peer to discuss how to create our game called matching colours. We will be discussing what network is my game and more importantly how my game works. My game is a simple game called Matching Colours. The game itself works over a network that considers two players that run on a network. The game will have an option of how difficult the guess can be. This can be done by adding more colours to make it harder for the opponent to guess each time. I would like for all of this to be in one project. You need another person to guess the colour, and the second player to actually guess the colour. The game's aim is easy to play as many times as the user can by the user it can. A plan to make it is that, the user can have an option to clear it each time they have played, to make it easier to guess the second time. It would not confuse any person of what they have chosen and what they have not chosen. Another feature that I would like to implement is clients talking to each other through the server and they would be able to send messages back and forth by communicating through the server.

The steps in which the following game discussed will work as the following:

1. The game start up by giving the users option to connect to the server, or directly play the game.
2. From this, either player one or player two connects to the server and connects to the client.
3. Player two straight away follows the same steps of difficulty and connects to the server by changing the IP Address with the same one as the server.
4. Once the two users have connected with each other, the player selects the chosen colour they would like to match automatically. [TCP]
5. This is acknowledged by the server, by stating which colour is selected by the user. [TCP]
6. This is then sent back to the both clients by which colour is stated [TCP]
7. This would be acknowledged by both clients and this would enable the result of the matching colours. [TCP]
8. The same process happens with the chats talking to each other on the client's side. [TCP]
9. The player sends a message to the server. [TCP]
10. The server acknowledges this message and automatically sends it to the both clients by sending this to the message box where they can both view it. [TCP]
11. Steps 4 to 11 can be repeated continuously as many times as the user would like to play the game. The user can also send messages continuously as many times as the user can like. There is no end game with this game. This can be a benefit as the game factor can be reusable as many times as it can be done.
12. Another option the user has whilst playing the game is to clear both sides if they feel necessary that it is getting confusing, or if the scoreboard is getting too long. [OPTION]
13. After the players have enough of playing matching the colour with each other, they have an option to exit the application. [OPTION]

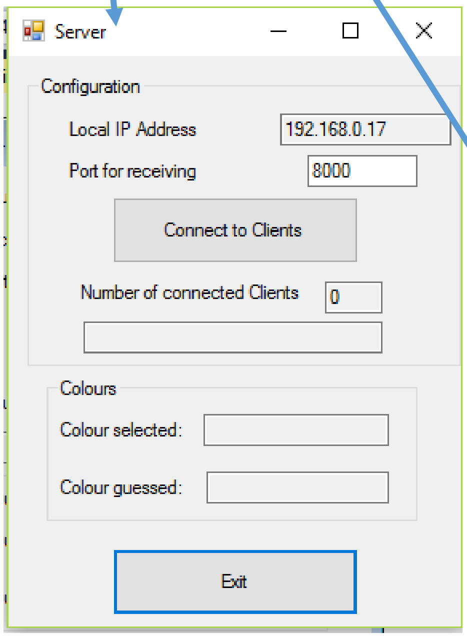
Requirement Analysis and Design

Design



This is the Main Menu for my game, called Matching Colours. As soon as the user the user runs the application, this screen appears by giving both clients to choose if one of them would want to be a server or client. By one of them choosing to be a server, they can start to play the game. Below is the server that shows the user to only click Connect to Clients and it enables binds, listens and accepts all in one. The colours are the communication between the clients and server. This shows it working in Figure 5. As soon as this is complete, the client must leave the server idle for it to communicate silently between the two. If this is shut, the communication between the two clients are broken and there is no game to play.

These arrows demonstrate which button shows which. The difficulty is about the game the user would like to choose. As you can see, they can choose between easy or hard.



Configuration

Local IP Address 192.168.0.17

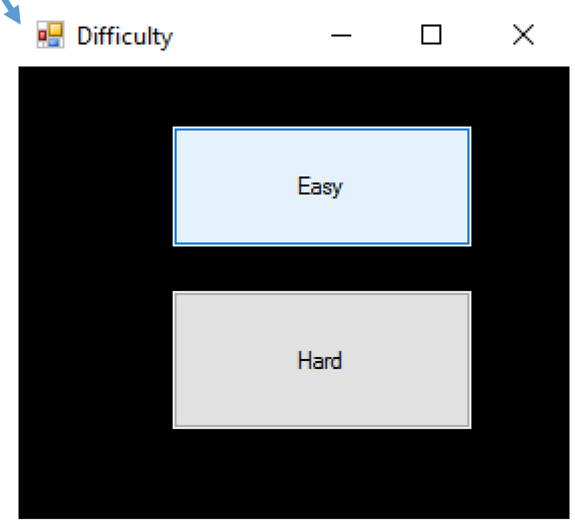
Port for receiving 8000

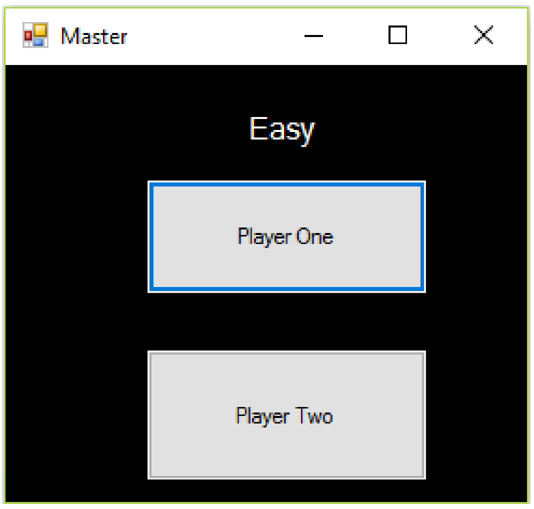
Number of connected Clients 0

Colours

Colour selected:

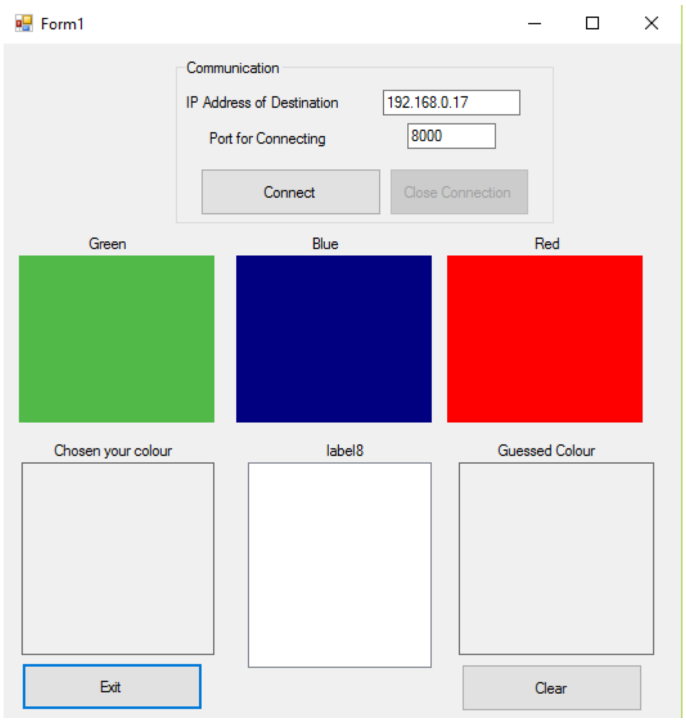
Colour guessed:





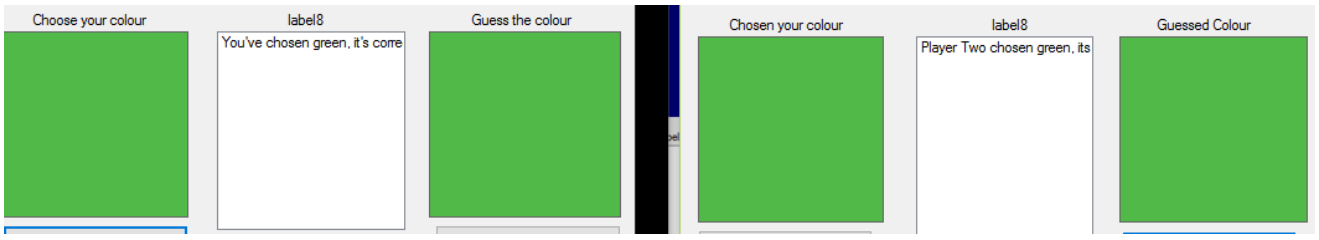
For difficulty, both clients need to be on the same page. For example, both clients need to pick hard and then connect to the server by choosing which player they would like to be.

Once the above steps are complete, both clients get to choose between what to pick between the two. As you can see, they can choose between player one and player two.



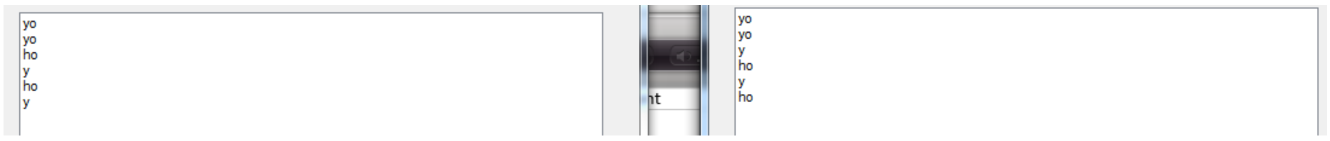
This is the screen that both clients should end up playing. For the users to connect, they should be able to choose between the two colours of what colours they would like to guess and match it.

Below shows the difference of matching the two together. Once they have been matched, they would be logged into the list box in between the two colours to let the users know if they have matched the colours or not.



You've chosen green, it's corre

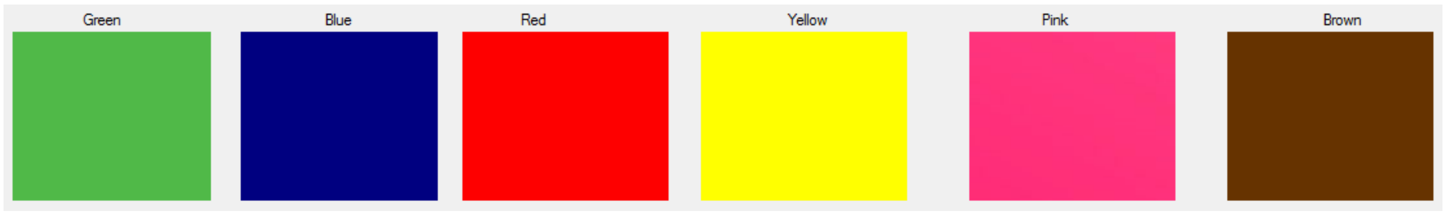
Player Two chosen green, its



yo
yo
ho
y
ho
y

yo
yo
y
ho
y
ho

This is below the two together. This shows the chat between the two clients communicating between the two. This shows that whilst the game is being played, the user can communicate by sending various messages between the two.



These images displayed above shows the different colours we have for the harder version. This makes the game much more interesting and harder to guess what colour one of the clients has chosen. We have two different to make it interesting and creativity. This game, as stated, is aimed at kids. This is not for adults.

Clients should be able to do the following

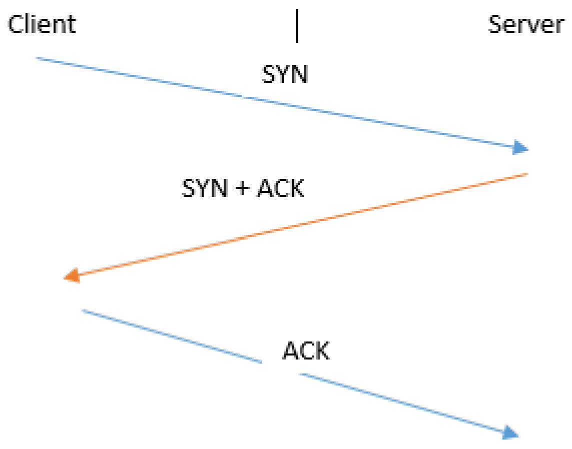
- Connect to the server using TCP Communication
- Type the IP Address to connect to the server to make sure its matching
- Receive opponents selection of the colour
- Receive opponents message
- Send messages to the server
- Show both chosen and guessed colour of both clients playing

Server should be able to do the following

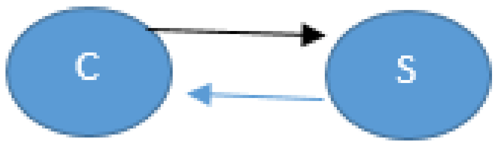
- Enable clients to be able to connect to the server
- Able to listen out to any clients trying to connect
- Send colours out to client
- Receive colours from client
- Receive message and send out messages from client

Application Level Communication

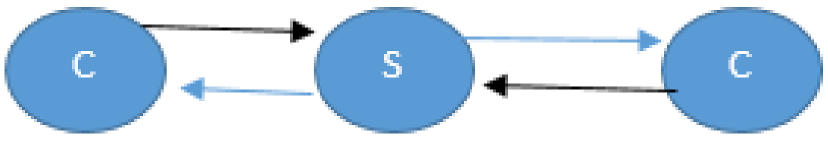
As you can see, the application communication level demonstrates the three way handshake. The three way handshake is before the communication has event began. The three way handshake is associated with TCP handshake. Before anything happens, the server needs to be able to create a socket which listens for any incoming connections. The server would then "bind" by listening to those connections. By accepting this, it would enable the incoming connections to be able to connect as soon as possible.



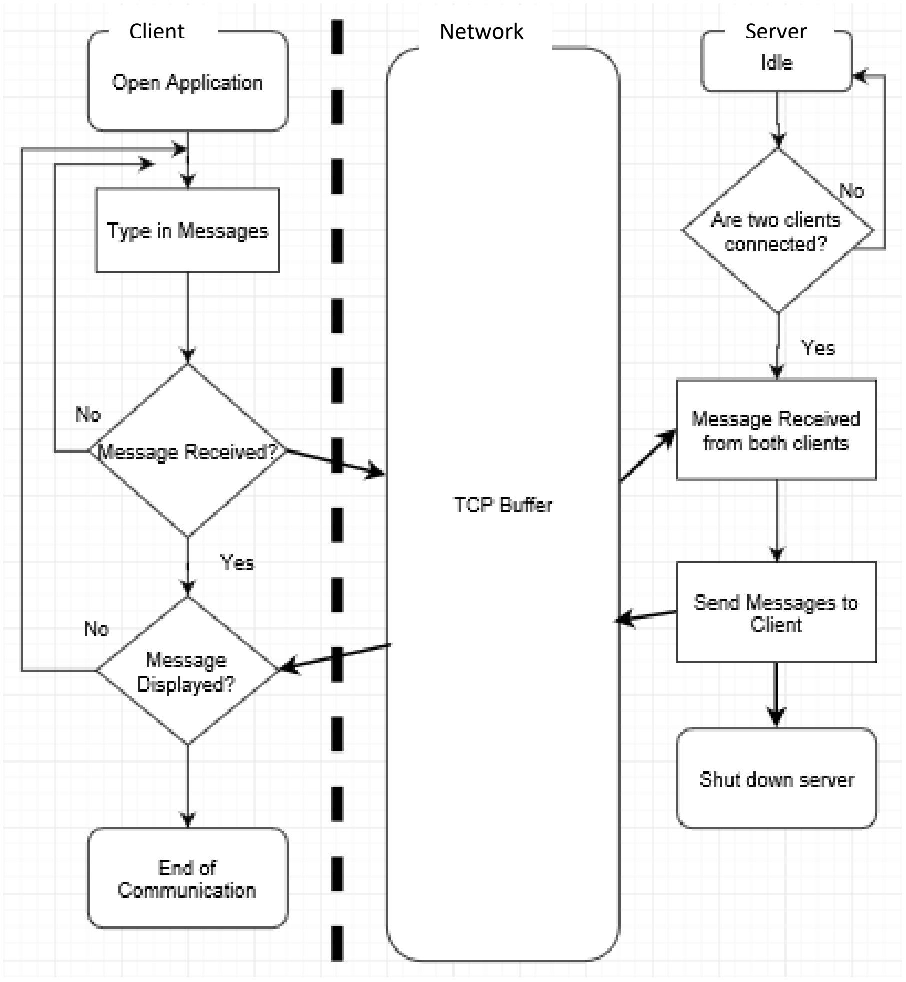
The image demonstrates the Synchronise packet sent to the server. Once this is complete, the server sends back a Synchronise and Acknowledgement to say to the server that they have received the request upon this. Once the client has received this, the client sends back an Acknowledgement to the server to let them know that this has happened. Once this is complete, the connection has been enabled and communication can be enabled upon the client and server. This is all demonstrated within the game clearly by the user typing in the IP Address for the client and the server connecting to each other. However, this is done before the communication is intact. In summary, this is all done before the game has begun. Any communication between the client and server has been demonstrated above.



This image above demonstrates in a diagram of how communication is between the client and server. S stands for Server and C stands for Client. These both have constantly communicating between the two. However, when the connecting is taking place, the server needs to be able to connect by typing the IP Address. Below is a demonstration of how two clients communicate between the server and client. Both of these images demonstrate how communication takes place effectively between the two. Below can be used within our game. For example, if C1 which is the first circle, sends Green to the server, it would save it and C2 which is the third circle sends out Green colour too. It would have both of them as Green as a match. The server would then send out a message to both clients at the same time letting them know that both colours have been matched. This is exactly the same for the chat that we have between the two clients as well.

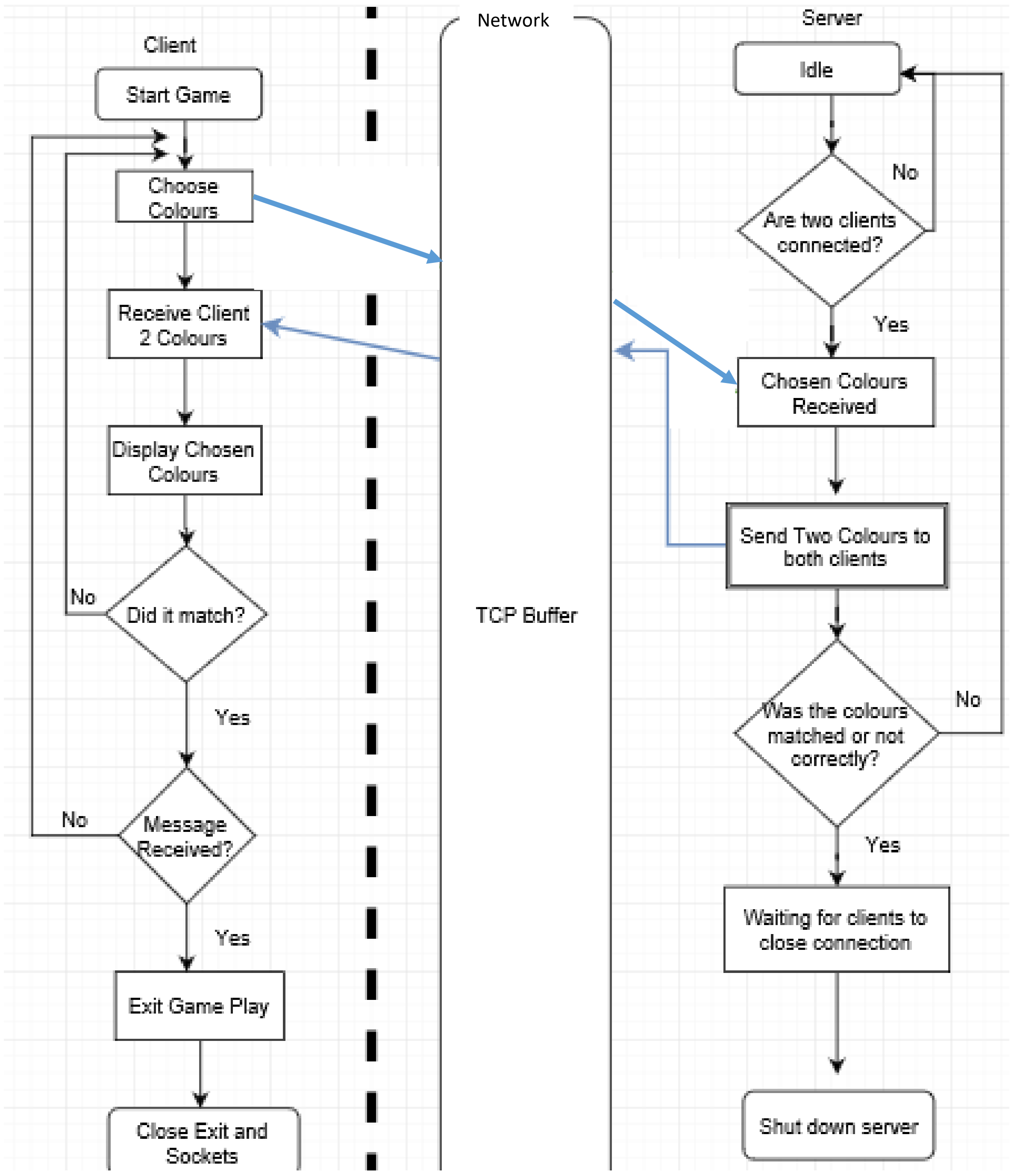


Chat Flowchart



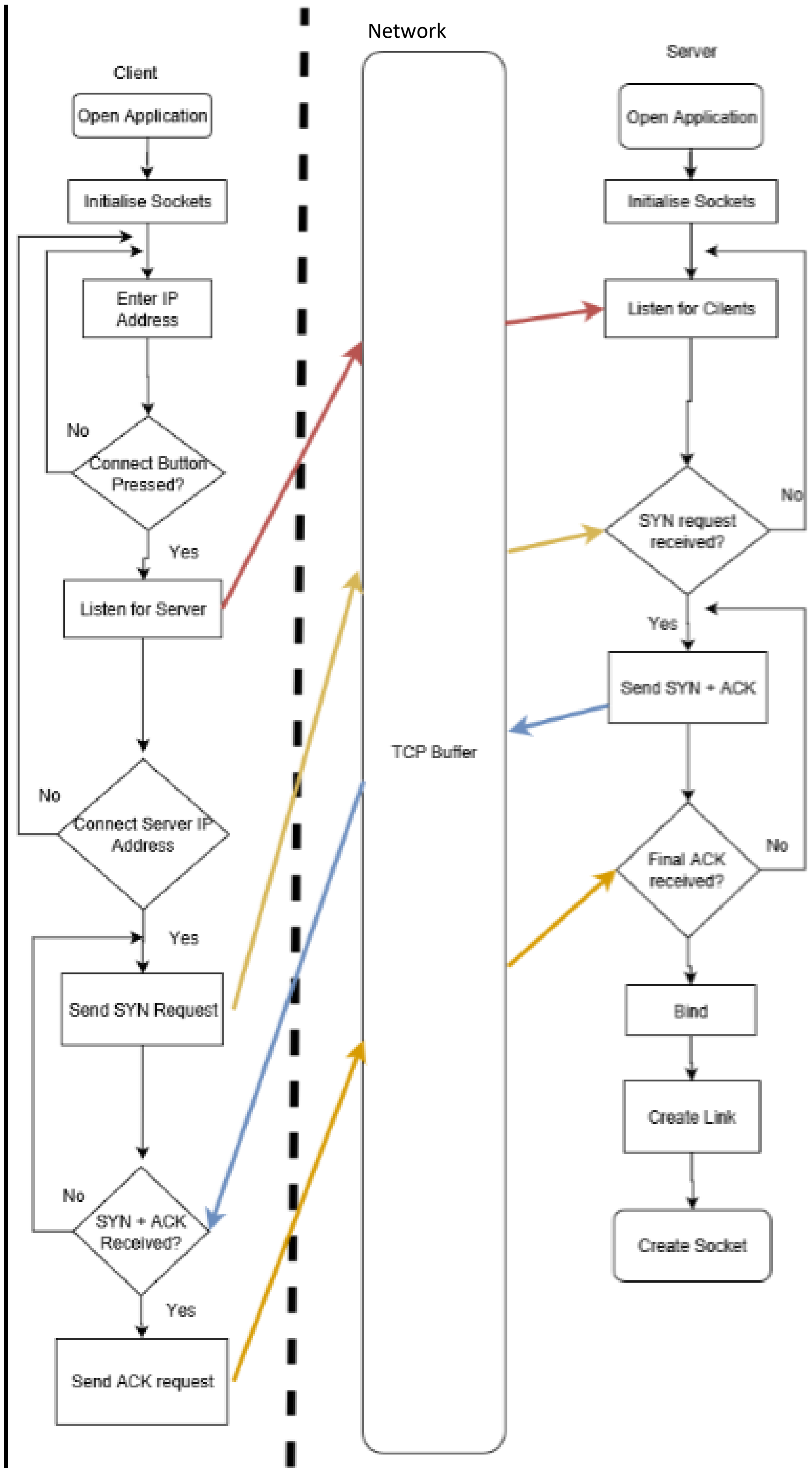
This flowchart demonstrates the communication between the server and client for the chat that is included within our program. As you can see, it shows the effectiveness of the client sending the server a clear message that is then sent back to the server to be displayed. The communication between the two demonstrates how it works between the two clients.

Game Play Flowchart



The above Flowchart demonstrates how the game communicates with the server and the client. As you can see, there is a communication between the server and the client. The clients has to choose to make sure they have chosen, they need to select the one they would like to pick. Once they have done this, it sends the colour to the server. This is retrieved and send back to the client by communicating between the two together.

TCP Establishment Flowchart



This flowchart demonstrates the TCP connection between the server and client. As you can see, we do have to enter the IP Address manually by the user entering this. If this is not correct, the user would need to make sure the IP Address is correct with the server for it to connect and play the game. We understand that by not entering IP Addresses can demonstrate a better idea of communication, however, due to time; we did not have that much for it to work with a broadcast. However, UDP broadcast is one of the big improvements that we can make for it.

Development

During this time, we left the original code as it is. We did try numerous of ways of enhancing this to UDP Broadcast, however, it kept crashing as stated in testing. When we got his code, the first thing we did try to get a communication between the server and client. By inputting the code below, it shows that it would get the messages from the server whatever the client has been sent to it. By getting the messages from the client, it would be able to get both of them and try to get a match between the two together. By putting this, it is done automatically as soon as both of the clients have been selected.

```
EndPoint RemoteEndPoint = (EndPoint)m_remoteEndPoint;
int iReceiveByteCount;
iReceiveByteCount = m_ClientSocket.ReceiveFrom(ReceiveBuffer, ref RemoteEndPoint);

string szReceivedMessage;
if (0 < iReceiveByteCount)
{   // Copy the number of bytes received, from the message buffer to the text control
    szReceivedMessage = Encoding.ASCII.GetString(ReceiveBuffer, 0, iReceiveByteCount);
    listBox2.Items.Add(szReceivedMessage);
    guessedcolour = szReceivedMessage;


    if (chosencolour == guessedcolour)
    {
        if (chosencolour == "Green")
        {
            MessageBox.Show("Player Two guessed it correctly");
            guessedpicturebox.Image = greenPictureBox.Image;
            listBox1.Items.Add("Player Two chosen green, its correct");
        }
```

The code below shows that whenever the first client is connected, it sends the same message to the first textbox. Once the second one is connected, it shows it in the second message box. This is implemented so that the clients can see that the communication is in full effect. Both of these codes show the development of where we needed to put both of them to have full effect of the code and how we managed to do it.

```
if (iIndex == 0)
{
    Message_textBox.Text = szReceivedMessage;
}
else if (iIndex == 1)
{
    Message_textBox2.Text = szReceivedMessage;
}

if (Message_textBox.Text != "" && Message_textBox2.Text != "")
{
    SendReplyToClients();
}
```

Testing

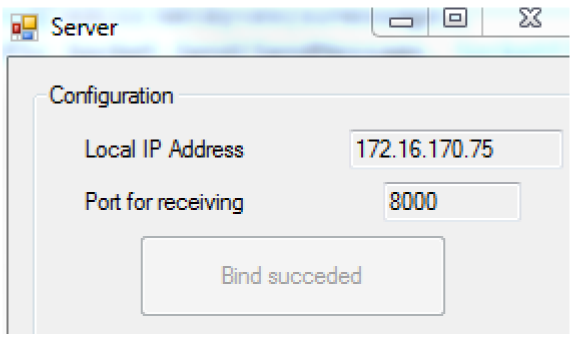
Test No	*Test Description*	*Expected Results*	*Actual Result*	*Actions Required*	*PASS/FAIL*	*Screenshot*
1	Server accepts to listen to any other clients.	Expecting the server to accept and listen out for any potential clients to accept to the server	The server did as expected by listening out for potential clients to accept and communicate with the server	No action was required.	**PASS**	Figure 1
2	Validation – editing textboxes	Expecting the textboxes not to be editable after it is not needed	The textboxes as expected was not editable. For example, when the user types in the address, after this, it is unable to edit.	No action was required.	**PASS**	Figure 2
3	Validation – clear specific area	Expecting the area selected to be cleared as soon as the button has been pressed	The area that was specified was cleared and it did the job it was necessary	No action required	**PASS**	Figure 3 and Figure 4
4	Broadcast of the IP Address with the TCP Connection	Expecting the broadcast to be sent to the client from the server and it automatically connects	The broadcast was sent and this was tried to be done, however, it did not manage to send the broadcast, nor connect automatically.	No action needed	**FAIL**	No screenshot needed
5	Colour selected sent to server	Expecting the colour selected to be sent to the server	The colour was sent to the server of the player's chose. For example, it could have been Red.	No Action required	**PASS**	Figure 5

6	Server sent back to the client acknowledging if it is correct	Expecting the server to send back to the client if the colour matches or not	The server sent back to the client as a message saying this is wrong and it is not correct.	No Action Required	**PASS**	Figure 6
7	Buttons – Exit	Expecting the exit button to close the application	The application as expected closed it	No Action is Required	**PASS**	No screenshot needed
8	Logging	Expecting the list box to log everything of it is matching or not	The logbook as expected has logged all the details of the games	No Action Required	**PASS**	Figure 6
9	Matching the IP Address	Expecting the IP Address to change and connect to the game	As expected, the IP Address changed and it connected with the game.	No Action Required	**PASS**	Figure 1 and Figure 2
10	Recognizing the server	Expecting the server to recognise that the client has been connected	The server recognises the client has been connected and recognised.	No Action Required	**PASS**	Figure 7
11	Disabling IP address after its use	Expecting the IP address to be disabled as soon as this has been changed or not.	The IP address is disabled as soon as it has been connected	No Action Required	**PASS**	Figure 1
12	The chat sends it back and forth	Expecting the chat to be sent from the client to the server and back and forth	As expected, the chat works perfectly by the messages appear as soon as it has been sent	No Action Required	**PASS**	Figure 8

Test Problems/Failures

We tried to do this a numerous of different ways by automatically connecting the client to the server by sending out a broadcast. One of the ways we tried to change the IP Address to, “172.16.170.255” and using a UDP Broadcast, but it did not acknowledge this either way. We felt that this would of improved by game numerous of ways by automatically connecting the IP Address. We feel that by not putting this is the only thing that we feel within my program is missing. We know the acknowledgement of missing this crucial part of the game out. I feel that it is necessary to write this down as one big improvement for next time. We know that what can be done by sending out a UDP Broadcast from the server side of the part and changing the default IP Address, so that it recognises a the current client would want a broadcast.

Screenshots


Server

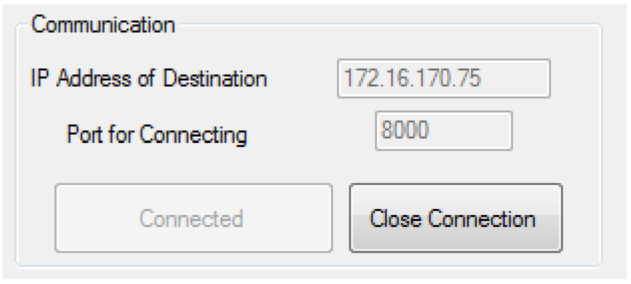
Configuration

Local IP Address: 172.16.170.75

Port for receiving: 8000

Bind succeded

Figure 2 shows that the server accepts any by the bind being succeeded.


Communication

IP Address of Destination: 172.16.170.75

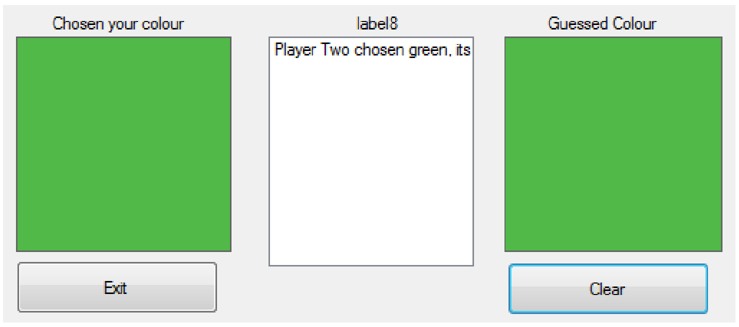
Port for Connecting: 8000

Connected

Close Connection

Figure 1 shows that it is not able to edit after it has been connected with the server

Figure 1 and Figure 2 demonstrates the connection between the two by enabling the server first to enable the client to join the two together. This is important to join the two together to start the connection and communication between the two together. Another way is that the other client that wants to join the server, the client needs to change the IP Address to the server's IP Address. Once this is complete, the other clients can join in together.



Player Two chosen green, its

Figure 4 shows the content that needs to be cleared

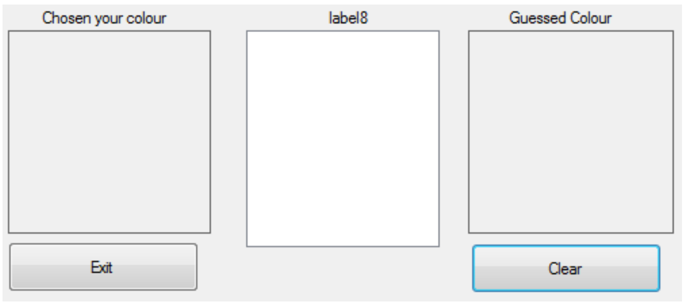
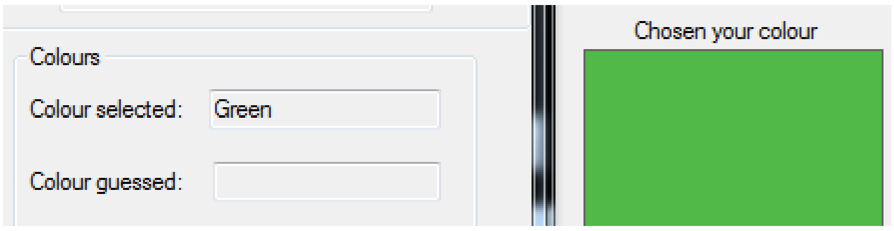


Figure 3 shows the area cleared

Figure 3 and Figure 4 shows the testing of each game. This shows that the clear button works effectively by once both clients have chosen each of their colours. Once this is complete, they can have an option to clear it all together.

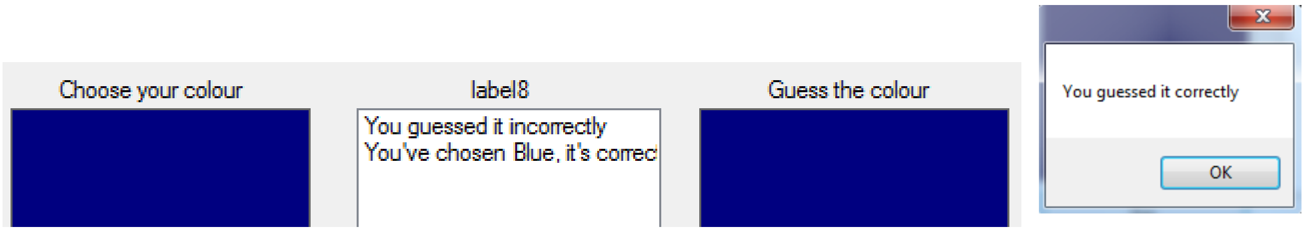

Colours

Colour selected: Green

Colour guessed:

Figure 5 shows the chosen colour sent to the server

Figure 5 is to show that the communication between the client and server is in effect. As you can see, the client has chosen Green as a colour. Once this is complete, it sends that acknowledgement to the server and it shows us that the communication between the two is intact.



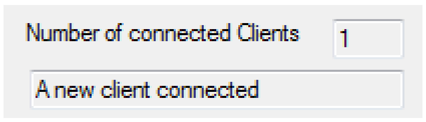
You guessed it incorrectly
You've chosen Blue, it's correc

You guessed it correctly

OK

Figure 6 shows the colour that has been chosen is correct and it's matching alongside its message sent.

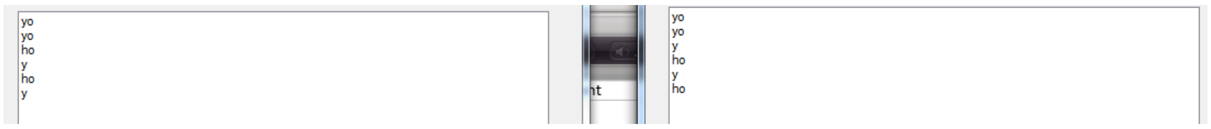
As you can see, Figure 6 shows that the colour of both chosen colours have been correctly matched. This shows that the colour that has been chosen from both clients have been correctly matched by it been logged in to effect with a message to show that it is correct.


Number of connected Clients 1

A new client connected

Figure 7 shows that one client has been recognised and connected

This shows that one client has been connected and recognised. This can be increased to 2 clients connected once they have been through each step of connecting the server and client together. Once this is complete, Number of connected Clients would be automatically updated.


```
yo
yo
ho
y
ho
y
```

```
yo
yo
y
ho
y
ho
```

Figure 8 shows the chat between the two clients

This shows the communication between the two clients together. The two way handshake communication is intact by sending messages between one to another. Messages that have been sent from one end to another has been chosen on the receiving end too as you can see above.

Evaluation

In conclusion, I would evaluate it so that what we intended to do from the start has worked out well. We wanted it so that it works from one application. We wanted this to happen so that it would be all in one application and the user does not need to execute different applications in order for the game to run. This makes it easier for the user to use the application. Another feature that I felt is good in our game, consists of being the game being reusable. We felt that the user can and will use it as many times as they can, because it does not have an end limit to this game. Not only the game that can be reusable; however, the chat communication between both clients can be used as many times as the user feels necessary. We felt that in terms of reusability, our game is important and outstanding. Another feature within our game that fits the requirement is the three way handshake that communicates between the client and server. We felt that this is necessary to be included as this is the key ingredient in any game that communicates on a network. We had complex ideas from the start and, due to the amount of time we started this, we felt that it is necessary we get the game working on a network running together. Our game includes two players running by guessing the colour by this, we only need two players running on the network. We have been using TCP since the start of the network. The big statement that is missing in our coursework is that, since it is aimed at children, we felt that the missing piece is the UDP Broadcast. We felt that by missing this out makes the game incomplete. This is because we know that children would not understand what IP Addresses are. Broadcast would automatically connect with the client and it would have been much easier for both client and server. We did try a numerous amounts of ways to do this game, however, we felt that by doing all of the fit requirements was necessary at the time. Another idea that I had beforehand is to by improving the game, we would have implemented a computer based idea where the computer chooses the colour and the player chooses the colour and see if it matches. We felt that this could have worked, but we did not have enough working together to do this and implement both of these ideas if we was to go back and do it again.

Initially, we only had the game running on. After receiving feedback from our program on the demo, we felt that by missing out on UDP Broadcast, we needed to implement a chatting system that can run on the network by communicating between the two. I felt that this was necessary to do so. By implementing this, I felt that it expands and makes the communication between the two clients much better and effective.

In terms of functionality and robustness, we felt that it has achieved both targets. Functionality is easy for the user to go back and forth. The reason for this is that the application being between the two can be effective. As stated before, we felt that the user would have no problems by not being able to use the application itself. We felt the same for robustness. We felt that the quality is satisfied and in good condition to run as demonstrated on the demo this week.

Reflection

During my time doing this coursework, I felt that I have learnt much more than I expected and I felt that this became much more fun as the weeks went on. I feel that it was necessary to have a partner to work with too. I feel that I gained much more experience by learning beside him. Without the commitment and motivation of my peer, we would not have achieved the things that we have today. Overall, I am pleased with the result of the game of how it worked. However, I am not pleased with the fact that I did not implement the things that we both wanted to put into this game. I felt that it was necessary to put these things in as it would have been advanced and much better. Finally, I would end up by recommending this course to the students next year and I hope that they would enjoy it as much as I have this year.

Program Listing

[HIGHLIGHTED PARTS ARE THE ONES THAT HAVE BEEN CHANGED]

Server

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApplication7
{
    public partial class Server : Form
    {

        private const int m_iMaxConnections = 2;
        Socket m_ClientSocket;

        struct Connection_Struct    // Define a structure to hold details about a single connection
        {
            public Socket ClientSpecific_Socket;
            public bool bInUse;
        };

        Socket m_ListenSocket;
        Connection_Struct[] m_Connection_Array = new Connection_Struct[m_iMaxConnections]; // Define an array
to hold a number of connections

        System.Net.IPEndPoint m_LocalIPEndPoint;
        static int m_iNumberOfConnectedClients;
        private static System.Windows.Forms.Timer m_CommunicationActivity_Timer;

        public Server()
        {
            InitializeComponent();
            Initialise_ConnectionArray();
            m_CommunicationActivity_Timer = new System.Windows.Forms.Timer(); // Check for communication
activity on Non-Blocking sockets every 200ms
            m_CommunicationActivity_Timer.Tick                    +=                    new
EventHandler(OnTimedEvent_PeriodicCommunicationActivityCheck); // Set event handler method for timer
            m_CommunicationActivity_Timer.Interval = 100;  // Timer interval is 1/10 second
            m_CommunicationActivity_Timer.Enabled = false;
            string szLocalIPAddress = GetLocalIPAddress_AsString(); // Get local IP address as a default value
            IP_Address_textBox.Text = szLocalIPAddress;           // Place local IP address in IP address field
            ReceivePort_textBox.Text = "8000";  // Default port number
            m_iNumberOfConnectedClients = 0;
            NumberOfClients_textBox.Text = System.Convert.ToString(m_iNumberOfConnectedClients);
            try
            {   // Create the Listen socket, for TCP use
                m_ListenSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
                m_ListenSocket.Blocking = false;
            }
            catch (SocketException se)
            {   // If an exception occurs, display an error message
                MessageBox.Show(se.Message);
            }
        }

        private void Initialise_ConnectionArray()
        {
            int iIndex;
            for (iIndex = 0; iIndex < m_iMaxConnections; iIndex++)
            {
                m_Connection_Array[iIndex].bInUse = false;
            }
        }

        private int GetnextAvailable_ConnectionArray_Entry()
        {
            int iIndex;
            for (iIndex = 0; iIndex < m_iMaxConnections; iIndex++)
            {
                if (false == m_Connection_Array[iIndex].bInUse)
                {
                    return iIndex;  // Return the index value of the first not-in-use entry found
                }
            }
            return -1;      // Signal that there were no available entries
        }

        private void Bind_button_Click(object sender, EventArgs e)
        {   // Bind to the selected port and start listening / receiving
            try
            {
                // Get the Port number from the appropriate text box
                String szPort = ReceivePort_textBox.Text;
                int iPort = System.Convert.ToInt16(szPort, 10);
                // Create an Endpoint that will cause the listening activity to apply to all the local node's interfaces
                m_LocalIPEndPoint = new System.Net.IPEndPoint(IPAddress.Any, iPort);
                // Bind to the local IP Address and selected port
                m_ListenSocket.Bind(m_LocalIPEndPoint);
                Bind_button.Enabled = false;
                Bind_button.Text = "Bind succeded";
                // Prevent any further changes to the port number
                ReceivePort_textBox.ReadOnly = true;
                m_ListenSocket.Listen(2); // Listen for connections, with a backlog / queue maximum of 2
                m_CommunicationActivity_Timer.Start();  // Start the timer to perform periodic checking for connection
requests
            }

            catch // Catch any errors
            {   // If an exception occurs, display an error message
                Bind_button.Text = "Bind failed";
            }
        }

        public string GetLocalIPAddress_AsString()
        {
            string szHost = Dns.GetHostName();
            string szLocalIPaddress = "127.0.0.1";  // Default is local loopback address
            IPHostEntry IPHost = Dns.GetHostEntry(Dns.GetHostName());
            foreach (IPAddress IP in IPHost.AddressList)
            {
                if (IP.AddressFamily == AddressFamily.InterNetwork) // Match only the IPv4 address
                {
                    szLocalIPaddress = IP.ToString();
                    break;
                }
            }
            return szLocalIPaddress;
        }

        private void Done_button_Click(object sender, EventArgs e)
        {
            Close_And_Quit();
        }

        private void Close_And_Quit()
        {   // Close the sockets and exit the application
            try
            {
                m_ListenSocket.Close();
            }
            catch
            {
            }
            try
            {
                int iIndex;
                for (iIndex = 0; iIndex < m_iMaxConnections; iIndex++)
                {
                    m_Connection_Array[iIndex].ClientSpecific_Socket.Shutdown(SocketShutdown.Both);
                    m_Connection_Array[iIndex].ClientSpecific_Socket.Close();
                }
            }
            catch
            {
            }
            try
            {
                Close();
            }
            catch
            {
            }
        }

        private void OnTimedEvent_PeriodicCommunicationActivityCheck(Object myObject, EventArgs
myEventArgs)
        {   // Periodic check whether a connection request is pending or a message has been received on a connected
socket

            // First, check for pending connection requests
            int iIndex;
            iIndex = GetnextAvailable_ConnectionArray_Entry(); // Find an available array entry for next connection
request
            if (-1 != iIndex)
            {   // Only continue with Accept if there is an array entry available to hold the details

                try
                {
                    m_Connection_Array[iIndex].ClientSpecific_Socket = m_ListenSocket.Accept();    // Accept a
connection (if pending) and assign a new socket to it (AcceptSocket)
```

```
                                        // Will 'catch' if NO connection was pending, so
statements below only occur when a connection WAS pending
          m_Connection_Array[iIndex].bInUse = true;
          m_Connection_Array[iIndex].ClientSpecific_Socket.Blocking = false;      // Make the new socket
operate in non-blocking mode
          m_iNumberOfConnectedClients++;
          NumberOfClients_textBox.Text = System.Convert.ToString(m_iNumberOfConnectedClients);
          Status_textBox.Text = "A new client connected";


          //  SendUpdateMesageToAllConnectedclients();
        }
        catch (SocketException se) // Handle socket-related exception
        {   // If an exception occurs, display an error message
          if (10053 == se.ErrorCode || 10054 == se.ErrorCode) // Remote end closed the connection
          {
            CloseConnection(iIndex);
          }
          else if (10035 != se.ErrorCode)
          {   // Ignore error messages relating to normal behaviour of non-blocking sockets
            MessageBox.Show(se.Message);
          }
        }
        catch // Silently handle any other exception
        {
        }
      }

      // Second, check for received messages on each connected socket
      for (iIndex = 0; iIndex < m_iMaxConnections; iIndex++)
      {
        if (true == m_Connection_Array[iIndex].bInUse)
        {
          try
          {
            EndPoint localEndPoint = (EndPoint)m_LocalIPEndPoint;
            byte[] ReceiveBuffer = new byte[1024];
            int iReceiveByteCount;
            iReceiveByteCount                                                        =
m_Connection_Array[iIndex].ClientSpecific_Socket.ReceiveFrom(ReceiveBuffer, ref localEndPoint);

            string szReceivedMessage;
            if (0 < iReceiveByteCount)
            {   // Copy the number of bytes received, from the message buffer to the text control
              szReceivedMessage = Encoding.ASCII.GetString(ReceiveBuffer, 0, iReceiveByteCount);
              if ("QuitConnection" == szReceivedMessage)
              {
                CloseConnection(iIndex);
              }
              else
              {
                //   Message_textBox.Text = szReceivedMessage;

                if (iIndex == 0)
                {
                  Message_textBox.Text = szReceivedMessage;
                }
                else if (iIndex == 1)
                {
                  Message_textBox2.Text = szReceivedMessage;
```

```
                    }

                    if (Message_textBox.Text != "" && Message_textBox2.Text != "")
                    {
                        SendReplyToClients();
                    }

                }
            }
        }
        catch (SocketException se) // Handle socket-related exception
        {   // If an exception occurs, display an error message
            if (10053 == se.ErrorCode || 10054 == se.ErrorCode) // Remote end closed the connection
            {
                CloseConnection(iIndex);
            }
            else if (10035 != se.ErrorCode)
            {   // Ignore error messages relating to normal behaviour of non-blocking sockets
                MessageBox.Show(se.Message);
            }
        }
        catch // Silently handle any other exception
        {
        }
    }
}
}

private void SendUpdateMesageToAllConnectedclients()
{   // Send message to each connected client informing of the total number of connected clients
    int iIndex;
    for (iIndex = 0; iIndex < m_iMaxConnections; iIndex++)
    {
        if (true == m_Connection_Array[iIndex].bInUse)
        {
            string szMessage;
            if (1 == m_iNumberOfConnectedClients)
            {
                szMessage = string.Format("There is now {0} client connected", m_iNumberOfConnectedClients);
            }
            else
            {
                szMessage = string.Format("There are now {0} clients connected", m_iNumberOfConnectedClients);
            }
            byte[] SendMessage = System.Text.Encoding.ASCII.GetBytes(szMessage);
            m_Connection_Array[iIndex].ClientSpecific_Socket.Send(SendMessage, SocketFlags.None);
        }
    }
}


private void SendReplyToClients()
{
    // Send message to each connected client informing of the total number of connected clients
    int iIndex;
    for (iIndex = 0; iIndex < m_iMaxConnections; iIndex++)
    {
        if (true == m_Connection_Array[iIndex].bInUse)
        {
            string szMessage = "";
```

```
                if (iIndex == 0)
                {
                    szMessage = Message_textBox2.Text;
                }
                else if (iIndex == 1)
                {
                    szMessage = Message_textBox.Text;
                }
                byte[] SendMessage = System.Text.Encoding.ASCII.GetBytes(szMessage);
                m_Connection_Array[iIndex].ClientSpecific_Socket.Send(SendMessage, SocketFlags.None);
              }
            }
            clearTextBoxes();
        }

        void clearTextBoxes()
        {
            Message_textBox.Text = "";
            Message_textBox2.Text = "";
        }



        private void CloseConnection(int iIndex)
        {
            try
            {
                m_Connection_Array[iIndex].bInUse = false;
                m_Connection_Array[iIndex].ClientSpecific_Socket.Shutdown(SocketShutdown.Both);
                m_Connection_Array[iIndex].ClientSpecific_Socket.Close();
                m_iNumberOfConnectedClients--;
                NumberOfClients_textBox.Text = System.Convert.ToString(m_iNumberOfConnectedClients);
                Status_textBox.Text = "A Connection was closed";
                SendUpdateMesageToAllConnectedclients();
            }
            catch // Silently handle any exceptions
            {
            }
        }

        private void Bind_button_Click_1(object sender, EventArgs e)
        {

            try
            {
                // Get the Port number from the appropriate text box
                String szPort = ReceivePort_textBox.Text;
                int iPort = System.Convert.ToInt16(szPort, 10);
                // Create an Endpoint that will cause the listening activity to apply to all the local node's interfaces
                m_LocalIPEndPoint = new System.Net.IPEndPoint(IPAddress.Any, iPort);
                // Bind to the local IP Address and selected port
                m_ListenSocket.Bind(m_LocalIPEndPoint);
                Bind_button.Enabled = false;
                Bind_button.Text = "Bind succeded";
                // Prevent any further changes to the port number
                ReceivePort_textBox.ReadOnly = true;
                m_ListenSocket.Listen(2); // Listen for connections, with a backlog / queue maximum of 2
                m_CommunicationActivity_Timer.Start();  // Start the timer to perform periodic checking for connection
requests
```

```
            }

            catch // Catch any errors
            {   // If an exception occurs, display an error message
                Bind_button.Text = "Bind failed";
            }
        }
    }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;

namespace WindowsFormsApplication7
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
     //    [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Main());
        }
    }
}
```

Master 2

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApplication7
{
    public partial class Master2 : Form
    {
        public Master2()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Form3 form = new WindowsFormsApplication7.Form3();
            form.Show();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            Form4 form = new Form4();
            form.Show();
        }
    }
}
```

Master

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApplication7
{
    public partial class Master : Form
    {
        public Master()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Form1 form = new Form1();
            form.Show();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            Form2 form = new Form2();
            form.Show();
        }
    }
}
```

Main

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApplication7
{
    public partial class Main : Form
    {
        public Main()
        {
            InitializeComponent();
        }

        private void button3_Click(object sender, EventArgs e)
        {

        }

        private void button2_Click(object sender, EventArgs e)
        {
            Difficulty form = new Difficulty();
            form.Show();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Server form = new Server();
            form.Show();
        }
    }
}
```

Difficulty

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApplication7
{
    public partial class Difficulty : Form
    {
        public Difficulty()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Master form = new Master();
            form.Show();
            this.Hide();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            Master2 form = new Master2();
            form.Show();
            this.Hide();
        }
    }
}
```

Client 1 [Easy]

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;

namespace WindowsFormsApplication7
{
  public partial class Form1 : Form
  {

    Socket m_ClientSocket;
    System.Net.IPEndPoint m_remoteEndPoint;
    private static System.Windows.Forms.Timer m_CommunicationActivity_Timer;
    string guessedcolour;
    string chosencolour;
    byte[] ReceiveBuffer = new byte[1024];

    public Form1()
    {
      InitializeComponent();

      m_CommunicationActivity_Timer = new System.Windows.Forms.Timer(); // Check for communication
activity on Non-Blocking sockets every 200ms
      m_CommunicationActivity_Timer.Tick                                    +=                              new
EventHandler(OnTimedEvent_PeriodicCommunicationActivityCheck); // Set event handler method for timer
      m_CommunicationActivity_Timer.Interval = 100;  // Timer interval is 1/10 second
      m_CommunicationActivity_Timer.Enabled = false;
      CloseConnection_button.Enabled = false;
      greenPictureBox.Enabled = false;
      bluePictureBox.Enabled = false;
      redPictureBox.Enabled = false;
      chosenpictureBox.Enabled = false;
      guessedpicturebox.Enabled = false;
      CloseConnection_button.Text = "Close Connection";
      string szLocalIPAddress = GetLocalIPAddress_AsString(); // Get local IP address as a default value
      IP_Address_textBox.Text = szLocalIPAddress;          // Place local IP address in IP address field
      SendPort_textBox.Text = "8000"; // Default port number

    }

    private string GetLocalIPAddress_AsString()
    {
      string szHost = Dns.GetHostName();
      string szLocalIPaddress = "127.0.0.1";  // Default is local loopback address
      IPHostEntry IPHost = Dns.GetHostEntry(Dns.GetHostName());
      foreach (IPAddress IP in IPHost.AddressList)
      {
        if (IP.AddressFamily == AddressFamily.InterNetwork) // Match only the IPv4 address
        {
          szLocalIPaddress = IP.ToString();
          break;
        }
      }
      return szLocalIPaddress;
    }


    private void btnClear_Click(object sender, EventArgs e)
    {
      guessedpicturebox.Image = null;
      chosenpictureBox.Image = null;
      panelPlayer.BackColor = Color.LightGray;
      panelComputer.BackColor = Color.LightGray;
    }


    private void Connect_button_Click(object sender, EventArgs e)
    {
      try
      {
        // Create the socket, for TCP use
        m_ClientSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
        m_ClientSocket.Blocking = true; // Socket operates in Blocking mode initially
      }
      catch // Handle any exceptions
      {
        Close_Socket_and_Exit();
      }
      try
      {
        // Get the IP address from the appropriate text box
        String szIPAddress = IP_Address_textBox.Text;
        System.Net.IPAddress DestinationIPAddress = System.Net.IPAddress.Parse(szIPAddress);

        // Get the Port number from the appropriate text box
        String szPort = SendPort_textBox.Text;
        int iPort = System.Convert.ToInt16(szPort, 10);

        // Combine Address and Port to create an Endpoint
        m_remoteEndPoint = new System.Net.IPEndPoint(DestinationIPAddress, iPort);

        m_ClientSocket.Connect(m_remoteEndPoint);
        m_ClientSocket.Blocking = false;    // Socket is now switched to Non-Blocking mode for send/ receive
activities
        Connect_button.Text = "Connected";
        Connect_button.Enabled = false;
        IP_Address_textBox.Enabled = false;
        SendPort_textBox.Enabled = false;
        CloseConnection_button.Enabled = true;
        greenPictureBox.Enabled = true;
        bluePictureBox.Enabled = true;
        redPictureBox.Enabled = true;
        chosenpictureBox.Enabled = true;
        guessedpicturebox.Enabled = true;
        CloseConnection_button.Text = "Close Connection";
        m_CommunicationActivity_Timer.Start();  // Start the timer to perform periodic checking for received
messages
      }
      catch // Catch all exceptions
```

```
            {   // If an exception occurs, display an error message
                Connect_button.Text = "(Connect attempt failed)\nRetry Connect";
            }
        }

        private void OnTimedEvent_PeriodicCommunicationActivityCheck(Object myObject, EventArgs
myEventArgs)
        {   // Periodic check whether a message has been received
            try
            {
                EndPoint RemoteEndPoint = (EndPoint)m_remoteEndPoint;
                int iReceiveByteCount;
                iReceiveByteCount = m_ClientSocket.ReceiveFrom(ReceiveBuffer, ref RemoteEndPoint);

                string szReceivedMessage;
                if (0 < iReceiveByteCount)
                {   // Copy the number of bytes received, from the message buffer to the text control
                    szReceivedMessage = Encoding.ASCII.GetString(ReceiveBuffer, 0, iReceiveByteCount);
                    listBox2.Items.Add(szReceivedMessage);
                    guessedcolour = szReceivedMessage;


                    if (chosencolour == guessedcolour)
                    {
                        if (chosencolour == "Green")
                        {
                            MessageBox.Show("Player Two guessed it correctly");
                            guessedpicturebox.Image = greenPictureBox.Image;
                            listBox1.Items.Add("Player Two chosen green, its correct");
                        }
                        else if (chosencolour == "Blue")
                        {
                            MessageBox.Show("Player Two guessed it correctly");
                            guessedpicturebox.Image = bluePictureBox.Image;
                            listBox1.Items.Add("Player Two chosen Blue, its correct");

                        }
                        else if (chosencolour == "Red")
                        {
                            MessageBox.Show("Player Two guessed it correctly");
                            guessedpicturebox.Image = redPictureBox.Image;
                            listBox1.Items.Add("Player Two chosen Red, its correct");


                        }
                    }
                    //lost
                    else if (chosencolour == "Red" && guessedcolour == "Green")
                    {
                        MessageBox.Show("Player Two guessed it incorrectly");
                        guessedpicturebox.Image = greenPictureBox.Image;
                        listBox1.Items.Add("Player Two chosen incorrect");


                    }
                    else if (chosencolour == "Red" && guessedcolour == "Blue")
                    {
                        MessageBox.Show("Player Two guessed it incorrectly");
                        guessedpicturebox.Image = bluePictureBox.Image;
                        listBox1.Items.Add("Player Two chosen incorrect");
                    }
                    else if (chosencolour == "Blue" && guessedcolour == "Red")
                    {
                        MessageBox.Show("Player Two guessed it incorrectly");
                        guessedpicturebox.Image = redPictureBox.Image;
                        listBox1.Items.Add("Player Two chosen incorrect");

                    }
                    else if (chosencolour == "Blue" && guessedcolour == "Green")
                    {
                        MessageBox.Show("Player Two guessed it incorrectly");
                        guessedpicturebox.Image = greenPictureBox.Image;
                        listBox1.Items.Add("Player Two chosen incorrect");

                    }
                    else if (chosencolour == "Green" && guessedcolour == "Red")
                    {
                        MessageBox.Show("Player Two guessed it incorrectly");
                        guessedpicturebox.Image = redPictureBox.Image;
                        listBox1.Items.Add("Player Two chosen incorrect");

                    }
                    else if (chosencolour == "Green" && guessedcolour == "Blue")
                    {
                        MessageBox.Show("Player Two guessed it incorrectly");
                        guessedpicturebox.Image = bluePictureBox.Image;
                        listBox1.Items.Add("Player Two chosen incorrect");

                    }
                }
            }
            catch // Silently handle any exceptions
            {
            }
        }

        private void Close_Socket_and_Exit()
        {
            try
            {
                m_ClientSocket.Shutdown(SocketShutdown.Both);
            }
            catch // Silently handle any exceptions
            {
            }
            try
            {
                m_ClientSocket.Close();
            }
            catch // Silently handle any exceptions
            {
            }
            this.Close();
        }

        private void CloseConnection_button_Click(object sender, EventArgs e)
        {
            try
            {
                String szData = "QuitConnection"; // Special code to signal 'close connection' to the server
                // This ensures that the server is aware the Client wants to close the connection
                // (TCP should otherwise automatically detect disconnection, but this approach ensures a clean disconnect)
                byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
                m_ClientSocket.Send(byData, SocketFlags.None);
                m_ClientSocket.Shutdown(SocketShutdown.Both);
                m_ClientSocket.Close();
                CloseConnection_button.Enabled = false;
                CloseConnection_button.Text = "Connection Closed";
                Connect_button.Enabled = true;
                Connect_button.Text = "Connect";
            }
            catch // Silently handle any exceptions
            {
            }
        }

        private void redPictureBox_Click_1(object sender, EventArgs e)
        {

            chosenpictureBox.Image = redPictureBox.Image;

            try
            {
                String szData = "Red";
                chosencolour = szData;

                byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
                m_ClientSocket.Send(byData, SocketFlags.None);
            }
            catch // Silently handle any exceptions
            {
            }
        }

        private void bluePictureBox_Click_1(object sender, EventArgs e)
        {

            chosenpictureBox.Image = bluePictureBox.Image;

            try
            {
                String szData = "Blue";
                chosencolour = szData;

                byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
                m_ClientSocket.Send(byData, SocketFlags.None);
            }
            catch // Silently handle any exceptions
            {
            }
        }

        private void greenPictureBox_Click_1(object sender, EventArgs e)
        {

            chosenpictureBox.Image = greenPictureBox.Image;

            try
            {
                String szData = "Green";
                chosencolour = szData;
                byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
                m_ClientSocket.Send(byData, SocketFlags.None);
            }
            catch // Silently handle any exceptions
            {
            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            chosenpictureBox.Image = null;
            guessedpicturebox.Image = null;
            listBox1.Items.Clear();
        }



        private void button3_Click(object sender, EventArgs e)
        {
            try
            {
                String szData = textBox1.Text;
                listBox2.Items.Add(szData);
                byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
                m_ClientSocket.Send(byData, SocketFlags.None);
            }
            catch // Silently handle any exceptions
            {
            }
            try
            {
                EndPoint RemoteEndPoint = (EndPoint)m_remoteEndPoint;
                byte[] ReceiveBuffer = new byte[1024];
                int iReceiveByteCount;
                iReceiveByteCount = m_ClientSocket.ReceiveFrom(ReceiveBuffer, ref RemoteEndPoint);
                string szReceivedMessage; // received answer from server
                if (0 < iReceiveByteCount)
                {   // Copy the number of bytes received, from the message buffer to the text control
                    szReceivedMessage = Encoding.ASCII.GetString(ReceiveBuffer, 0, iReceiveByteCount);
                    listBox2.Items.Add(szReceivedMessage);
                }
            }
            catch
            {
            }
        }
    }
}
```

Client 2 [Easy]

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;

namespace WindowsFormsApplication7
{
    public partial class Form2 : Form
    {

        Socket m_ClientSocket;
        System.Net.IPEndPoint m_remoteEndPoint;
        private static System.Windows.Forms.Timer m_CommunicationActivity_Timer;
        string guessedcolour;
        string chosencolour;


        public Form2()
        {
            InitializeComponent();

            m_CommunicationActivity_Timer = new System.Windows.Forms.Timer(); // Check for communication
activity on Non-Blocking sockets every 200ms
            m_CommunicationActivity_Timer.Tick += new
EventHandler(OnTimedEvent_PeriodicCommunicationActivityCheck); // Set event handler method for timer
            m_CommunicationActivity_Timer.Interval = 100;  // Timer interval is 1/10 second
            m_CommunicationActivity_Timer.Enabled = false;
            CloseConnection_button.Enabled = false;
            // getting them to be false as soon as it starts up
            greenPictureBox.Enabled = false;
            bluePictureBox.Enabled = false;
            redPictureBox.Enabled = false;
            chosenpictureBox.Enabled = false;
            guessedpicturebox.Enabled = false;
            CloseConnection_button.Text = "Close Connection";
            string szLocalIPAddress = GetLocalIPAddress_AsString(); // Get local IP address as a default value
            IP_Address_textBox.Text = szLocalIPAddress;          // Place local IP address in IP address field
            SendPort_textBox.Text = "8000"; // Default port number

        }

        private string GetLocalIPAddress_AsString()
        {
            string szHost = Dns.GetHostName();
            string szLocalIPaddress = "127.0.0.1";  // Default is local loopback address
            IPHostEntry IPHost = Dns.GetHostEntry(Dns.GetHostName());
            foreach (IPAddress IP in IPHost.AddressList)
            {
```

```
            if (IP.AddressFamily == AddressFamily.InterNetwork) // Match only the IPv4 address
            {
                szLocalIPaddress = IP.ToString();
                break;
            }
        }
        return szLocalIPaddress;
    }


    private void Connect_button_Click(object sender, EventArgs e)
    {
        try
        {
            // Create the socket, for TCP use
            m_ClientSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
            m_ClientSocket.Blocking = true; // Socket operates in Blocking mode initially
        }
        catch // Handle any exceptions
        {
            Close_Socket_and_Exit();
        }
        try
        {
            // Get the IP address from the appropriate text box
            String szIPAddress = IP_Address_textBox.Text;
            System.Net.IPAddress DestinationIPAddress = System.Net.IPAddress.Parse(szIPAddress);

            // Get the Port number from the appropriate text box
            String szPort = SendPort_textBox.Text;
            int iPort = System.Convert.ToInt16(szPort, 10);

            // Combine Address and Port to create an Endpoint
            m_remoteEndPoint = new System.Net.IPEndPoint(DestinationIPAddress, iPort);

            m_ClientSocket.Connect(m_remoteEndPoint);
            m_ClientSocket.Blocking = false;    // Socket is now switched to Non-Blocking mode for send/ receive
activities
            Connect_button.Text = "Connected";
            Connect_button.Enabled = false;
//getting them to be active and IP Address to be false as soon as its connected
            IP_Address_textBox.Enabled = false;
            SendPort_textBox.Enabled = false;
            CloseConnection_button.Enabled = true;
            greenPictureBox.Enabled = true;
            bluePictureBox.Enabled = true;
            redPictureBox.Enabled = true;
            chosenpictureBox.Enabled = true;
            guessedpicturebox.Enabled = true;
            CloseConnection_button.Text = "Close Connection";
            m_CommunicationActivity_Timer.Start();  // Start the timer to perform periodic checking for received
messages
        }
        catch // Catch all exceptions
        {   // If an exception occurs, display an error message
            Connect_button.Text = "(Connect attempt failed)\nRetry Connect";
        }
    }
    private void OnTimedEvent_PeriodicCommunicationActivityCheck(Object myObject, EventArgs myEventArgs)
    {   // Periodic check whether a message has been received
        try
        {
            EndPoint RemoteEndPoint = (EndPoint)m_remoteEndPoint;
            byte[] ReceiveBuffer = new byte[1024];
            int iReceiveByteCount;
            iReceiveByteCount = m_ClientSocket.ReceiveFrom(ReceiveBuffer, ref RemoteEndPoint);

            string szReceivedMessage; // received answer from server
            if (0 < iReceiveByteCount)
            {   // Copy the number of bytes received, from the message buffer to the text control
                szReceivedMessage = Encoding.ASCII.GetString(ReceiveBuffer, 0, iReceiveByteCount);
                listBox2.Items.Add(szReceivedMessage);
                guessedcolour = szReceivedMessage;


                if (chosencolour == guessedcolour)
                {
                    //guessed correctly
                    if (chosencolour == "Green")
                    {
                        MessageBox.Show("You Guessed it correctly");
                        guessedpicturebox.Image = greenPictureBox.Image;
                        listBox1.Items.Add("You've chosen green, it's correct");

                    }
                    else if (chosencolour == "Blue")
                    {
                        MessageBox.Show("You guessed it correctly");
                        guessedpicturebox.Image = bluePictureBox.Image;
                        listBox1.Items.Add("You've chosen Blue, it's correct");

                    }
                    else if (chosencolour == "Red")
                    {
                        MessageBox.Show("You guessed it correctly");
                        guessedpicturebox.Image = redPictureBox.Image;
                        listBox1.Items.Add("You've chosen Red, it's correct");
                    }
                }

                //lost
                else if (chosencolour == "Red" && guessedcolour == "Green")
                {
                    MessageBox.Show("You guessed it incorrectly");
                    guessedpicturebox.Image = greenPictureBox.Image;
                    listBox1.Items.Add("You've chosen incorrect");

                }
                else if (chosencolour == "Red" && guessedcolour == "Blue")
                {
                    MessageBox.Show("You guessed it incorrectly");
                    guessedpicturebox.Image = bluePictureBox.Image;
                    listBox1.Items.Add("You guessed it incorrectly");

                }
                else if (chosencolour == "Blue" && guessedcolour == "Red")
                {
                    MessageBox.Show("You guessed it incorrectly");
                    guessedpicturebox.Image = redPictureBox.Image;
                    listBox1.Items.Add("You guessed it incorrectly");

                }
                else if (chosencolour == "Blue" && guessedcolour == "Green")
                {
                    MessageBox.Show("You guessed it incorrectly");
                    guessedpicturebox.Image = greenPictureBox.Image;
                    listBox1.Items.Add("You guessed it incorrectly");
                }
                else if (chosencolour == "Green" && guessedcolour == "Red")
                {
                    MessageBox.Show("You guessed it incorrectly");
                    guessedpicturebox.Image = redPictureBox.Image;
                    listBox1.Items.Add("You guessed it incorrectly");

                }
                else if (chosencolour == "Green" && guessedcolour == "Blue")
                {
                    MessageBox.Show("You guessed it incorrectly");
                    guessedpicturebox.Image = bluePictureBox.Image;
                    listBox1.Items.Add("You guessed it incorrectly");

                }
            }
        }
        catch // Silently handle any exceptions
        {
        }
    }

    private void Close_Socket_and_Exit()
    {
        try
        {
            m_ClientSocket.Shutdown(SocketShutdown.Both);
        }
        catch // Silently handle any exceptions
        {
        }
        try
        {
            m_ClientSocket.Close();
        }
        catch // Silently handle any exceptions
        {
        }
        this.Close();
    }

    private void CloseConnection_button_Click_1(object sender, EventArgs e)
    {

        try
        {
            String szData = "QuitConnection"; // Special code to signal 'close connection' to the server
            // This ensures that the server is aware the Client wants to close the connection
            // (TCP should otherwise automatically detect disconnection, but this approach ensures a clean disconnect)
            byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
            m_ClientSocket.Send(byData, SocketFlags.None);
            m_ClientSocket.Shutdown(SocketShutdown.Both);
            m_ClientSocket.Close();
            CloseConnection_button.Enabled = false;
            CloseConnection_button.Text = "Connection Closed";
            Connect_button.Enabled = true;
            Connect_button.Text = "Connect";
        }
        catch // Silently handle any exceptions
        {
        }
    }

    private void ExitButton_Click_1(object sender, EventArgs e)
    {
        Application.Exit();
    }

    private void clearButton_Click(object sender, EventArgs e)
    {
        guessedpicturebox.Image = null;
        chosenpictureBox.Image = null;
        panelPlayer.BackColor = Color.LightGray;
        panelComputer.BackColor = Color.LightGray;
    }

    private void greenPictureBox_Click(object sender, EventArgs e)
    {
        chosenpictureBox.Image = greenPictureBox.Image;

        try
        {
            String szData = "Green"; //chosen colour to select
            chosencolour = szData; //sent to socket
            byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
            m_ClientSocket.Send(byData, SocketFlags.None);
        }
        catch // Silently handle any exceptions
        {
        }
    }

    private void bluePictureBox_Click(object sender, EventArgs e)
    {
        chosenpictureBox.Image = bluePictureBox.Image;

        try
        {
            String szData = "Blue"; //chosen colour to select
            chosencolour = szData; //sent to socket

            byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
            m_ClientSocket.Send(byData, SocketFlags.None);
        }
        catch // Silently handle any exceptions
        {
        }
    }

    private void redPictureBox_Click(object sender, EventArgs e)
```

```
        {
            chosenpictureBox.Image = redPictureBox.Image;
            try
            {
                String szData = "Red"; //chosen colour to select
                chosencolour = szData; //sent to socket

                byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
                m_ClientSocket.Send(byData, SocketFlags.None);
            }
            catch // Silently handle any exceptions
            {
            }
        }

        private void button3_Click(object sender, EventArgs e)
        {
            try
            {
                String szData = textBox1.Text;
                listBox2.Items.Add(szData);
                byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
                m_ClientSocket.Send(byData, SocketFlags.None);
            }
            catch // Silently handle any exceptions
            {
            }
            try
            {
                EndPoint RemoteEndPoint = (EndPoint)m_remoteEndPoint;
                byte[] ReceiveBuffer = new byte[1024];
                int iReceiveByteCount;
                iReceiveByteCount = m_ClientSocket.ReceiveFrom(ReceiveBuffer, ref RemoteEndPoint);
                string szReceivedMessage; // received answer from server
                if (0 < iReceiveByteCount)
                {   // Copy the number of bytes received, from the message buffer to the text control
                    szReceivedMessage = Encoding.ASCII.GetString(ReceiveBuffer, 0, iReceiveByteCount);
                    listBox2.Items.Add(szReceivedMessage);
                }
            }
            catch
            {
            }
        }
    }
}
```

Client 3 [Hard]

```
using System;
   using System.Collections.Generic;
   using System.ComponentModel;
   using System.Data;
   using System.Drawing;
   using System.Linq;
   using System.Text;
   using System.Threading.Tasks;
   using System.Windows.Forms;
   using System.Net;
   using System.Net.Sockets;

   namespace WindowsFormsApplication7
   {
       public partial class Form3 : Form
       {

           Socket m_ClientSocket;
           System.Net.IPEndPoint m_remoteEndPoint;
           private static System.Windows.Forms.Timer m_CommunicationActivity_Timer;
           string guessedcolour;
           string chosencolour;


           public Form3()
           {
               InitializeComponent();

               m_CommunicationActivity_Timer = new System.Windows.Forms.Timer(); // Check for communication
activity on Non-Blocking sockets every 200ms
               m_CommunicationActivity_Timer.Tick += new
EventHandler(OnTimedEvent_PeriodicCommunicationActivityCheck); // Set event handler method for timer
               m_CommunicationActivity_Timer.Interval = 100;  // Timer interval is 1/10 second
               m_CommunicationActivity_Timer.Enabled = false;
               CloseConnection_button.Enabled = false;
               greenPictureBox.Enabled = false;
               bluePictureBox.Enabled = false;
               redPictureBox.Enabled = false;
               chosenpictureBox.Enabled = false;
               guessedpicturebox.Enabled = false;
               CloseConnection_button.Text = "Close Connection";
               string szLocalIPAddress = GetLocalIPAddress_AsString(); // Get local IP address as a default value
               IP_Address_textBox.Text = szLocalIPAddress;          // Place local IP address in IP address field
               SendPort_textBox.Text = "8000"; // Default port number

           }

           private string GetLocalIPAddress_AsString()
           {
               string szHost = Dns.GetHostName();
               string szLocalIPaddress = "127.0.0.1";  // Default is local loopback address
               IPHostEntry IPHost = Dns.GetHostEntry(Dns.GetHostName());
               foreach (IPAddress IP in IPHost.AddressList)
```

```
            {
                if (IP.AddressFamily == AddressFamily.InterNetwork) // Match only the IPv4 address
                {
                    szLocalIPaddress = IP.ToString();
                    break;
                }
            }
            return szLocalIPaddress;
        }


        private void OnTimedEvent_PeriodicCommunicationActivityCheck(Object myObject, EventArgs
myEventArgs)
        {   // Periodic check whether a message has been received
            try
            {
                EndPoint RemoteEndPoint = (EndPoint)m_remoteEndPoint;
                byte[] ReceiveBuffer = new byte[1024];
                int iReceiveByteCount;
                iReceiveByteCount = m_ClientSocket.ReceiveFrom(ReceiveBuffer, ref RemoteEndPoint);

                string szReceivedMessage; // received answer from server
                if (0 < iReceiveByteCount)
                {   // Copy the number of bytes received, from the message buffer to the text control
                    szReceivedMessage = Encoding.ASCII.GetString(ReceiveBuffer, 0, iReceiveByteCount);
                listBox2.Items.Add(szReceivedMessage);
                guessedcolour = szReceivedMessage;


                if (chosencolour == guessedcolour)
                {
                    //guessed correctly
                    if (chosencolour == "Green")
                    {
                        MessageBox.Show("Player Two guessed it correctly");
                        guessedpicturebox.Image = greenPictureBox.Image;
                        listBox1.Items.Add("Player Two chosen Green, its correct");

                    }
                    else if (chosencolour == "Blue")
                    {
                        MessageBox.Show("Player Two guessed it correctly");
                        guessedpicturebox.Image = bluePictureBox.Image;
                        listBox1.Items.Add("Player Two chosen Blue, its correct");

                    }
                    else if (chosencolour == "Red")
                    {
                        MessageBox.Show("Player Two guessed it correctly");
                        guessedpicturebox.Image = redPictureBox.Image;
                        listBox1.Items.Add("Player Two chosen Red, its correct");
                    }
                    else if (chosencolour == "Yellow")
                    {
                        MessageBox.Show("Player Two guessed it correctly");
                        guessedpicturebox.Image = yellowpictureBox.Image;
                        listBox1.Items.Add("Player Two chosen Yellow, its correct");
                    }
                    else if (chosencolour == "Pink")
                    {
```

```
                MessageBox.Show("Player Two guessed it correctly");
                guessedpicturebox.Image = yellowpictureBox.Image;
                listBox1.Items.Add("Player Two chosen Pink, its correct");
            }
            else if (chosencolour == "Brown")
            {
                MessageBox.Show("Player Two guessed it correctly");
                guessedpicturebox.Image = yellowpictureBox.Image;
                listBox1.Items.Add("Player Two chosen Brown, its correct");
            }
        }
        //lost
        //lost red
        else if (chosencolour == "Red" && guessedcolour == "Green")
        {
            MessageBox.Show("Player Two guessed it incorrectly");
            guessedpicturebox.Image = greenPictureBox.Image;
            listBox1.Items.Add("Player Two chosen incorrect");

        }
        else if (chosencolour == "Red" && guessedcolour == "Blue")
        {
            MessageBox.Show("Player Two guessed it incorrectly");
            guessedpicturebox.Image = bluePictureBox.Image;
            listBox1.Items.Add("Player Two chosen incorrect");
        }


        else if (chosencolour == "Red" && guessedcolour == "Pink")
        {
            MessageBox.Show("Player Two guessed it incorrectly");
            guessedpicturebox.Image = pinkpictureBox.Image;
            listBox1.Items.Add("Player Two chosen incorrect");

        }
        else if (chosencolour == "Red" && guessedcolour == "Brown")
        {
            MessageBox.Show("Player Two guessed it incorrectly");
            guessedpicturebox.Image = brownpictureBox.Image;
            listBox1.Items.Add("Player Two chosen incorrect");

        }
        else if (chosencolour == "Red" && guessedcolour == "Yellow")
        {
            MessageBox.Show("Player Two guessed it incorrectly");
            guessedpicturebox.Image = yellowpictureBox.Image;
            listBox1.Items.Add("Player Two chosen incorrect");

        }
        // lost blue
        else if (chosencolour == "Blue" && guessedcolour == "Red")
        {
            MessageBox.Show("Player Two guessed it incorrectly");
            guessedpicturebox.Image = redPictureBox.Image;
            listBox1.Items.Add("Player Two chosen incorrect");

        }
        else if (chosencolour == "Blue" && guessedcolour == "Green")
        {
            MessageBox.Show("Player Two guessed it incorrectly");
```

```
                guessedpicturebox.Image = greenPictureBox.Image;
                listBox1.Items.Add("Player Two chosen incorrect");
            }
            else if (chosencolour == "Blue" && guessedcolour == "Pink")
            {
                MessageBox.Show("Player Two guessed it incorrectly");
                guessedpicturebox.Image = pinkpictureBox.Image;
                listBox1.Items.Add("Player Two chosen incorrect");

            }
            else if (chosencolour == "Blue" && guessedcolour == "Brown")
            {
                MessageBox.Show("Player Two guessed it incorrectly");
                guessedpicturebox.Image = brownpictureBox.Image;
                listBox1.Items.Add("Player Two chosen incorrect");

            }
            else if (chosencolour == "Blue" && guessedcolour == "Yellow")
            {
                MessageBox.Show("Player Two guessed it incorrectly");
                guessedpicturebox.Image = yellowpictureBox.Image;
                listBox1.Items.Add("Player Two chosen incorrect");

            }
            //lost green
            else if (chosencolour == "Green" && guessedcolour == "Red")
            {
                MessageBox.Show("Player Two guessed it incorrectly");
                guessedpicturebox.Image = redPictureBox.Image;
                listBox1.Items.Add("Player Two chosen incorrect");

            }
            else if (chosencolour == "Green" && guessedcolour == "Blue")
            {
                MessageBox.Show("Player Two guessed it incorrectly");
                guessedpicturebox.Image = bluePictureBox.Image;
                listBox1.Items.Add("Player Two chosen incorrect");
            }
            else if (chosencolour == "Green" && guessedcolour == "Pink")
            {
                MessageBox.Show("Player Two guessed it incorrectly");
                guessedpicturebox.Image = pinkpictureBox.Image;
                listBox1.Items.Add("Player Two chosen incorrect");

            }
            else if (chosencolour == "Green" && guessedcolour == "Brown")
            {
                MessageBox.Show("Player Two guessed it incorrectly");
                guessedpicturebox.Image = brownpictureBox.Image;
                listBox1.Items.Add("Player Two chosen incorrect");

            }
            else if (chosencolour == "Green" && guessedcolour == "Yellow")
            {
                MessageBox.Show("Player Two guessed it incorrectly");
                guessedpicturebox.Image = yellowpictureBox.Image;
                listBox1.Items.Add("Player Two chosen incorrect");

            } //lost yellow
            else if (chosencolour == "Yellow" && guessedcolour == "Red")
            {
              MessageBox.Show("Player Two guessed it incorrectly");
              guessedpicturebox.Image = redPictureBox.Image;
              listBox1.Items.Add("Player Two chosen incorrect");

            }
            else if (chosencolour == "Yellow" && guessedcolour == "Blue")
            {
              MessageBox.Show("Player Two guessed it incorrectly");
              guessedpicturebox.Image = bluePictureBox.Image;
              listBox1.Items.Add("Player Two chosen incorrect");
            }
            else if (chosencolour == "Yellow" && guessedcolour == "Pink")
            {
              MessageBox.Show("Player Two guessed it incorrectly");
              guessedpicturebox.Image = pinkpictureBox.Image;
              listBox1.Items.Add("Player Two chosen incorrect");

            }
            else if (chosencolour == "Yellow" && guessedcolour == "Brown")
            {
              MessageBox.Show("Player Two guessed it incorrectly");
              guessedpicturebox.Image = brownpictureBox.Image;
              listBox1.Items.Add("Player Two chosen incorrect");

            }
            else if (chosencolour == "Yellow" && guessedcolour == "Green")
            {
              MessageBox.Show("Player Two guessed it incorrectly");
              guessedpicturebox.Image = greenPictureBox.Image;
              listBox1.Items.Add("Player Two chosen incorrect");

            }
            //lost brown
            else if (chosencolour == "Brown" && guessedcolour == "Red")
            {
              MessageBox.Show("Player Two guessed it incorrectly");
              guessedpicturebox.Image = redPictureBox.Image;
              listBox1.Items.Add("Player Two chosen incorrect");

            }
            else if (chosencolour == "Brown" && guessedcolour == "Blue")
            {
              MessageBox.Show("Player Two guessed it incorrectly");
              guessedpicturebox.Image = bluePictureBox.Image;
              listBox1.Items.Add("You guessed it incorrectly");
            }
            else if (chosencolour == "Brown" && guessedcolour == "Pink")
            {
              MessageBox.Show("Player Two guessed it incorrectly");
              guessedpicturebox.Image = pinkpictureBox.Image;
              listBox1.Items.Add("Player Two chosen incorrect");

            }
            else if (chosencolour == "Brown" && guessedcolour == "Green")
            {
              MessageBox.Show("Player Two guessed it incorrectly");
              guessedpicturebox.Image = greenPictureBox.Image;
              listBox1.Items.Add("Player Two chosen incorrect");
```

```
                }
                else if (chosencolour == "Brown" && guessedcolour == "Yellow")
                {
                    MessageBox.Show("Player Two guessed it incorrectly");
                    guessedpicturebox.Image = yellowpictureBox.Image;
                    listBox1.Items.Add("Player Two chosen incorrect");

                }
                //lost pink
                else if (chosencolour == "Pink" && guessedcolour == "Red")
                {
                    MessageBox.Show("Player Two guessed it incorrectly");
                    guessedpicturebox.Image = redPictureBox.Image;
                    listBox1.Items.Add("Player Two chosen incorrect");

                }
                else if (chosencolour == "Pink" && guessedcolour == "Blue")
                {
                    MessageBox.Show("Player Two guessed it incorrectly");
                    guessedpicturebox.Image = bluePictureBox.Image;
                    listBox1.Items.Add("Player Two chosen incorrect");
                }
                else if (chosencolour == "Pink" && guessedcolour == "Green")
                {
                    MessageBox.Show("Player Two guessed it incorrectly");
                    guessedpicturebox.Image = greenPictureBox.Image;
                    listBox1.Items.Add("Player Two chosen incorrect");

                }
                else if (chosencolour == "Pink" && guessedcolour == "Yellow")
                {
                    MessageBox.Show("Player Two guessed it incorrectly");
                    guessedpicturebox.Image = yellowpictureBox.Image;
                    listBox1.Items.Add("Player Two chosen incorrect");

                }
                else if (chosencolour == "Pink" && guessedcolour == "Brown")
                {
                    MessageBox.Show("Player Two guessed it incorrectly");
                    guessedpicturebox.Image = brownpictureBox.Image;
                    listBox1.Items.Add("Player Two chosen incorrect");
                }
            }
        }
        catch // Silently handle any exceptions
            {
            }
        }

        private void Close_Socket_and_Exit()
        {
            try
            {
                m_ClientSocket.Shutdown(SocketShutdown.Both);
            }
            catch // Silently handle any exceptions
            {
            }
            try
            {
```

```
        m_ClientSocket.Close();
    }
    catch // Silently handle any exceptions
    {
    }
    this.Close();
}



private void greenPictureBox_Click(object sender, EventArgs e)
{
    chosenpictureBox.Image = greenPictureBox.Image;

    try
    {
        String szData = "Green"; //chosen colour to select
        chosencolour = szData; //sent to socket
        byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
        m_ClientSocket.Send(byData, SocketFlags.None);
    }
    catch // Silently handle any exceptions
    {
    }
}

private void redPictureBox_Click(object sender, EventArgs e)
{
    chosenpictureBox.Image = redPictureBox.Image;
    try
    {
        String szData = "Red"; //chosen colour to select
        chosencolour = szData; //sent to socket

        byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
        m_ClientSocket.Send(byData, SocketFlags.None);
    }
    catch // Silently handle any exceptions
    {
    }
}

private void bluePictureBox_Click_1(object sender, EventArgs e)
{
    chosenpictureBox.Image = bluePictureBox.Image;

    try
    {
        String szData = "Blue"; //chosen colour to select
        chosencolour = szData; //sent to socket

        byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
        m_ClientSocket.Send(byData, SocketFlags.None);
    }
    catch // Silently handle any exceptions
    {
    }
}

private void clearButton_Click(object sender, EventArgs e)
```

```
        {
            guessedpicturebox.Image = null;
            chosenpictureBox.Image = null;
            panelPlayer.BackColor = Color.LightGray;
            panelComputer.BackColor = Color.LightGray;
        }

        private void ExitButton_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void CloseConnection_button_Click(object sender, EventArgs e)
        {
            try
            {
                String szData = "QuitConnection"; // Special code to signal 'close connection' to the server
                // This ensures that the server is aware the Client wants to close the connection
                // (TCP should otherwise automatically detect disconnection, but this approach ensures a clean
disconnect)
                byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
                m_ClientSocket.Send(byData, SocketFlags.None);
                m_ClientSocket.Shutdown(SocketShutdown.Both);
                m_ClientSocket.Close();
                CloseConnection_button.Enabled = false;
                CloseConnection_button.Text = "Connection Closed";
                Connect_button.Enabled = true;
                Connect_button.Text = "Connect";
            }
            catch // Silently handle any exceptions
            {
            }
        }

        private void Connect_button_Click(object sender, EventArgs e)
        {
            try
            {
                // Create the socket, for TCP use
                m_ClientSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
                m_ClientSocket.Blocking = true; // Socket operates in Blocking mode initially
            }
            catch // Handle any exceptions
            {
                Close_Socket_and_Exit();
            }
            try
            {
                // Get the IP address from the appropriate text box
                String szIPAddress = IP_Address_textBox.Text;
                System.Net.IPAddress DestinationIPAddress = System.Net.IPAddress.Parse(szIPAddress);

                // Get the Port number from the appropriate text box
                String szPort = SendPort_textBox.Text;
                int iPort = System.Convert.ToInt16(szPort, 10);

                // Combine Address and Port to create an Endpoint
                m_remoteEndPoint = new System.Net.IPEndPoint(DestinationIPAddress, iPort);

                m_ClientSocket.Connect(m_remoteEndPoint);
                m_ClientSocket.Blocking = false;    // Socket is now switched to Non-Blocking mode for send/ receive
activities
                Connect_button.Text = "Connected";
                Connect_button.Enabled = false;
                IP_Address_textBox.Enabled = false;
                SendPort_textBox.Enabled = false;
                CloseConnection_button.Enabled = true;
                greenPictureBox.Enabled = true;
                bluePictureBox.Enabled = true;
                redPictureBox.Enabled = true;
                chosenpictureBox.Enabled = true;
                guessedpicturebox.Enabled = true;
                CloseConnection_button.Text = "Close Connection";
                m_CommunicationActivity_Timer.Start();  // Start the timer to perform periodic checking for received
messages
            }
            catch // Catch all exceptions
            {   // If an exception occurs, display an error message
                Connect_button.Text = "(Connect attempt failed)\nRetry Connect";
            }
        }

        private void pictureBox1_Click(object sender, EventArgs e)
        {
            chosenpictureBox.Image = yellowpictureBox.Image;

            try
            {
                String szData = "Yellow"; //chosen colour to select
                chosencolour = szData; //sent to socket

                byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
                m_ClientSocket.Send(byData, SocketFlags.None);
            }
            catch // Silently handle any exceptions
            {
            }
        }

        private void pinkpictureBox_Click(object sender, EventArgs e)
        {
            chosenpictureBox.Image = pinkpictureBox.Image;

            try
            {
                String szData = "Pink"; //chosen colour to select
                chosencolour = szData; //sent to socket

                byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
                m_ClientSocket.Send(byData, SocketFlags.None);
            }
            catch // Silently handle any exceptions
            {
            }
        }

        private void brownpictureBox_Click(object sender, EventArgs e)
        {
            chosenpictureBox.Image = brownpictureBox.Image;
            try
            {
                String szData = "Brown"; //chosen colour to select
                chosencolour = szData; //sent to socket

                byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
                m_ClientSocket.Send(byData, SocketFlags.None);
            }
            catch // Silently handle any exceptions
            {
            }
        }

    private void button3_Click(object sender, EventArgs e)
    {
        try
        {
            String szData = textBox1.Text;
            listBox2.Items.Add(szData);
            byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
            m_ClientSocket.Send(byData, SocketFlags.None);
        }
        catch // Silently handle any exceptions
        {
        }
        try
        {
            EndPoint RemoteEndPoint = (EndPoint)m_remoteEndPoint;
            byte[] ReceiveBuffer = new byte[1024];
            int iReceiveByteCount;
            iReceiveByteCount = m_ClientSocket.ReceiveFrom(ReceiveBuffer, ref RemoteEndPoint);
            string szReceivedMessage; // received answer from server
            if (0 < iReceiveByteCount)
            {   // Copy the number of bytes received, from the message buffer to the text control
                szReceivedMessage = Encoding.ASCII.GetString(ReceiveBuffer, 0, iReceiveByteCount);
                listBox2.Items.Add(szReceivedMessage);
            }
        }
        catch
        {
        }
    }
  }
  }
```

Client 4 [Hard]

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;

namespace WindowsFormsApplication7
{
    public partial class Form4 : Form
    {

        Socket m_ClientSocket;
        System.Net.IPEndPoint m_remoteEndPoint;
        private static System.Windows.Forms.Timer m_CommunicationActivity_Timer;
        string guessedcolour;
        string chosencolour;


        public Form4()
        {
            InitializeComponent();

            m_CommunicationActivity_Timer = new System.Windows.Forms.Timer(); // Check for communication
activity on Non-Blocking sockets every 200ms
            m_CommunicationActivity_Timer.Tick += new
EventHandler(OnTimedEvent_PeriodicCommunicationActivityCheck); // Set event handler method for timer
            m_CommunicationActivity_Timer.Interval = 100;  // Timer interval is 1/10 second
            m_CommunicationActivity_Timer.Enabled = false;
            CloseConnection_button.Enabled = false;
            greenPictureBox.Enabled = false;
            bluePictureBox.Enabled = false;
            redPictureBox.Enabled = false;
            chosenpictureBox.Enabled = false;
            guessedpicturebox.Enabled = false;
            CloseConnection_button.Text = "Close Connection";
            string szLocalIPAddress = GetLocalIPAddress_AsString(); // Get local IP address as a default value
            IP_Address_textBox.Text = szLocalIPAddress;         // Place local IP address in IP address field
            SendPort_textBox.Text = "8000"; // Default port number

        }

        private string GetLocalIPAddress_AsString()
        {
            string szHost = Dns.GetHostName();
            string szLocalIPaddress = "127.0.0.1";  // Default is local loopback address
            IPHostEntry IPHost = Dns.GetHostEntry(Dns.GetHostName());
            foreach (IPAddress IP in IPHost.AddressList)
            {
                if (IP.AddressFamily == AddressFamily.InterNetwork) // Match only the IPv4 address
                {
                    szLocalIPaddress = IP.ToString();
                    break;
                }
            }
            return szLocalIPaddress;
        }
```

```
    private void OnTimedEvent_PeriodicCommunicationActivityCheck(Object myObject, EventArgs myEventArgs)
    {   // Periodic check whether a message has been received
        try
        {
            EndPoint RemoteEndPoint = (EndPoint)m_remoteEndPoint;
            byte[] ReceiveBuffer = new byte[1024];
            int iReceiveByteCount;
            iReceiveByteCount = m_ClientSocket.ReceiveFrom(ReceiveBuffer, ref RemoteEndPoint);

            string szReceivedMessage; // received answer from server
            if (0 < iReceiveByteCount)
            {   // Copy the number of bytes received, from the message buffer to the text control
                szReceivedMessage = Encoding.ASCII.GetString(ReceiveBuffer, 0, iReceiveByteCount);
                listBox2.Items.Add(szReceivedMessage);
                guessedcolour = szReceivedMessage;

                if (chosencolour == guessedcolour)
                {
                    //guessed correctly
                    if (chosencolour == "Green")
                    {
                        MessageBox.Show("You Guessed it correctly");
                        guessedpicturebox.Image = greenPictureBox.Image;
                        listBox1.Items.Add("You've chosen green, it's correct");

                    }
                    else if (chosencolour == "Blue")
                    {
                        MessageBox.Show("You guessed it correctly");
                        guessedpicturebox.Image = bluePictureBox.Image;
                        listBox1.Items.Add("You've chosen Blue, it's correct");

                    }
                    else if (chosencolour == "Red")
                    {
                        MessageBox.Show("You guessed it correctly");
                        guessedpicturebox.Image = redPictureBox.Image;
                        listBox1.Items.Add("You've chosen Red, it's correct");
                    }
                    else if (chosencolour == "Yellow")
                    {
                        MessageBox.Show("You guessed it correctly");
                        guessedpicturebox.Image = yellowpictureBox.Image;
                        listBox1.Items.Add("You've chosen yellow, it's correct");
                    }
                    else if (chosencolour == "Pink")
                    {
```

```
                MessageBox.Show("You guessed it correctly");
                guessedpicturebox.Image = yellowpictureBox.Image;
                listBox1.Items.Add("You've chosen Pink, it's correct");
            }
            else if (chosencolour == "Brown")
            {
                MessageBox.Show("You guessed it correctly");
                guessedpicturebox.Image = yellowpictureBox.Image;
                listBox1.Items.Add("You've chosen Brown, it's correct");
            }
        }

        //lost
        //lost red
        else if (chosencolour == "Red" && guessedcolour == "Green")
        {
            MessageBox.Show("You guessed it incorrectly");
            guessedpicturebox.Image = greenPictureBox.Image;
            listBox1.Items.Add("You've chosen incorrect");

        }
        else if (chosencolour == "Red" && guessedcolour == "Blue")
        {
            MessageBox.Show("You guessed it incorrectly");
            guessedpicturebox.Image = bluePictureBox.Image;
            listBox1.Items.Add("You guessed it incorrectly");
        }


        else if (chosencolour == "Red" && guessedcolour == "Pink")
        {
            MessageBox.Show("You guessed it incorrectly");
            guessedpicturebox.Image = pinkpictureBox.Image;
            listBox1.Items.Add("You've chosen incorrect");

        }
        else if (chosencolour == "Red" && guessedcolour == "Brown")
        {
            MessageBox.Show("You guessed it incorrectly");
            guessedpicturebox.Image = brownpictureBox.Image;
            listBox1.Items.Add("You guessed it incorrectly");

        }
        else if (chosencolour == "Red" && guessedcolour == "Yellow")
        {
            MessageBox.Show("You guessed it incorrectly");
            guessedpicturebox.Image = yellowpictureBox.Image;
            listBox1.Items.Add("You guessed it incorrectly");

        }
        // lost blue
        else if (chosencolour == "Blue" && guessedcolour == "Red")
        {
            MessageBox.Show("You guessed it incorrectly");
            guessedpicturebox.Image = redPictureBox.Image;
            listBox1.Items.Add("You guessed it incorrectly");

        }
        else if (chosencolour == "Blue" && guessedcolour == "Green")
        {
                MessageBox.Show("You guessed it incorrectly");
                guessedpicturebox.Image = greenPictureBox.Image;
                listBox1.Items.Add("You guessed it incorrectly");
            }
            else if (chosencolour == "Blue" && guessedcolour == "Pink")
            {
                MessageBox.Show("You guessed it incorrectly");
                guessedpicturebox.Image = pinkpictureBox.Image;
                listBox1.Items.Add("You've chosen incorrect");

            }
            else if (chosencolour == "Blue" && guessedcolour == "Brown")
            {
                MessageBox.Show("You guessed it incorrectly");
                guessedpicturebox.Image = brownpictureBox.Image;
                listBox1.Items.Add("You guessed it incorrectly");

            }
            else if (chosencolour == "Blue" && guessedcolour == "Yellow")
            {
                MessageBox.Show("You guessed it incorrectly");
                guessedpicturebox.Image = yellowpictureBox.Image;
                listBox1.Items.Add("You guessed it incorrectly");

            }
            //lost green
            else if (chosencolour == "Green" && guessedcolour == "Red")
            {
                MessageBox.Show("You guessed it incorrectly");
                guessedpicturebox.Image = redPictureBox.Image;
                listBox1.Items.Add("You guessed it incorrectly");

            }
            else if (chosencolour == "Green" && guessedcolour == "Blue")
            {
                MessageBox.Show("You guessed it incorrectly");
                guessedpicturebox.Image = bluePictureBox.Image;
                listBox1.Items.Add("You guessed it incorrectly");
            }
            else if (chosencolour == "Green" && guessedcolour == "Pink")
            {
                MessageBox.Show("You guessed it incorrectly");
                guessedpicturebox.Image = pinkpictureBox.Image;
                listBox1.Items.Add("You've chosen incorrect");

            }
            else if (chosencolour == "Green" && guessedcolour == "Brown")
            {
                MessageBox.Show("You guessed it incorrectly");
                guessedpicturebox.Image = brownpictureBox.Image;
                listBox1.Items.Add("You guessed it incorrectly");

            }
            else if (chosencolour == "Green" && guessedcolour == "Yellow")
            {
                MessageBox.Show("You guessed it incorrectly");
                guessedpicturebox.Image = yellowpictureBox.Image;
                listBox1.Items.Add("You guessed it incorrectly");

            }
            //lost yellow
            else if (chosencolour == "Yellow" && guessedcolour == "Red")
            {
                MessageBox.Show("You guessed it incorrectly");
                guessedpicturebox.Image = redPictureBox.Image;
                listBox1.Items.Add("You guessed it incorrectly");

            }
            else if (chosencolour == "Yellow" && guessedcolour == "Blue")
            {
                MessageBox.Show("You guessed it incorrectly");
                guessedpicturebox.Image = bluePictureBox.Image;
                listBox1.Items.Add("You guessed it incorrectly");
            }
            else if (chosencolour == "Yellow" && guessedcolour == "Pink")
            {
                MessageBox.Show("You guessed it incorrectly");
                guessedpicturebox.Image = pinkpictureBox.Image;
                listBox1.Items.Add("You've chosen incorrect");

            }
            else if (chosencolour == "Yellow" && guessedcolour == "Brown")
            {
                MessageBox.Show("You guessed it incorrectly");
                guessedpicturebox.Image = brownpictureBox.Image;
                listBox1.Items.Add("You guessed it incorrectly");

            }
            else if (chosencolour == "Yellow" && guessedcolour == "Green")
            {
                MessageBox.Show("You guessed it incorrectly");
                guessedpicturebox.Image = greenPictureBox.Image;
                listBox1.Items.Add("You guessed it incorrectly");

            }
            //lost brown
            else if (chosencolour == "Brown" && guessedcolour == "Red")
            {
                MessageBox.Show("You guessed it incorrectly");
                guessedpicturebox.Image = redPictureBox.Image;
                listBox1.Items.Add("You guessed it incorrectly");

            }
            else if (chosencolour == "Brown" && guessedcolour == "Blue")
            {
                MessageBox.Show("You guessed it incorrectly");
                guessedpicturebox.Image = bluePictureBox.Image;
                listBox1.Items.Add("You guessed it incorrectly");
            }
            else if (chosencolour == "Brown" && guessedcolour == "Pink")
            {
                MessageBox.Show("You guessed it incorrectly");
                guessedpicturebox.Image = pinkpictureBox.Image;
                listBox1.Items.Add("You've chosen incorrect");

            }
            else if (chosencolour == "Brown" && guessedcolour == "Green")
            {
                MessageBox.Show("You guessed it incorrectly");
                guessedpicturebox.Image = greenPictureBox.Image;
```

```
                    listBox1.Items.Add("You guessed it incorrectly");

                }
                else if (chosencolour == "Brown" && guessedcolour == "Yellow")
                {
                    MessageBox.Show("You guessed it incorrectly");
                    guessedpicturebox.Image = yellowpictureBox.Image;
                    listBox1.Items.Add("You guessed it incorrectly");

                }
                //lost pink
                else if (chosencolour == "Pink" && guessedcolour == "Red")
                {
                    MessageBox.Show("You guessed it incorrectly");
                    guessedpicturebox.Image = redPictureBox.Image;
                    listBox1.Items.Add("You guessed it incorrectly");

                }
                else if (chosencolour == "Pink" && guessedcolour == "Blue")
                {
                    MessageBox.Show("You guessed it incorrectly");
                    guessedpicturebox.Image = bluePictureBox.Image;
                    listBox1.Items.Add("You guessed it incorrectly");
                }
                else if (chosencolour == "Pink" && guessedcolour == "Green")
                {
                    MessageBox.Show("You guessed it incorrectly");
                    guessedpicturebox.Image = greenPictureBox.Image;
                    listBox1.Items.Add("You guessed it incorrectly");

                }
                else if (chosencolour == "Pink" && guessedcolour == "Yellow")
                {
                    MessageBox.Show("You guessed it incorrectly");
                    guessedpicturebox.Image = yellowpictureBox.Image;
                    listBox1.Items.Add("You guessed it incorrectly");

                }
                else if (chosencolour == "Pink" && guessedcolour == "Brown")
                {
                    MessageBox.Show("You guessed it incorrectly");
                    guessedpicturebox.Image = brownpictureBox.Image;
                    listBox1.Items.Add("You guessed it incorrectly");
                }
            }
        }
        catch // Silently handle any exceptions
        {
        }
    }

    private void Close_Socket_and_Exit()
    {
        try
        {
            m_ClientSocket.Shutdown(SocketShutdown.Both);
        }
        catch // Silently handle any exceptions
        {
        }
        try
        {
            m_ClientSocket.Close();
        }
        catch // Silently handle any exceptions
        {
        }
        this.Close();
    }



    private void greenPictureBox_Click(object sender, EventArgs e)
    {
        chosenpictureBox.Image = greenPictureBox.Image;

        try
        {
            String szData = "Green"; //chosen colour to select
            chosencolour = szData; //sent to socket
            byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
            m_ClientSocket.Send(byData, SocketFlags.None);
        }
        catch // Silently handle any exceptions
        {
        }
    }

    private void redPictureBox_Click(object sender, EventArgs e)
    {
        chosenpictureBox.Image = redPictureBox.Image;
        try
        {
            String szData = "Red"; //chosen colour to select
            chosencolour = szData; //sent to socket

            byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
            m_ClientSocket.Send(byData, SocketFlags.None);
        }
        catch // Silently handle any exceptions
        {
        }
    }

    private void clearButton_Click(object sender, EventArgs e)
    {
        guessedpicturebox.Image = null;
        chosenpictureBox.Image = null;
        panelPlayer.BackColor = Color.LightGray;
        panelComputer.BackColor = Color.LightGray;
    }

    private void ExitButton_Click(object sender, EventArgs e)
    {
        Application.Exit();
    }

    private void CloseConnection_button_Click(object sender, EventArgs e)
    {
        try
```

```
            {
                String szData = "QuitConnection"; // Special code to signal 'close connection' to the server
                                                  // This ensures that the server is aware the Client wants to close the connection
                                                  // (TCP should otherwise automatically detect disconnection, but this approach
ensures a clean disconnect)
                byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
                m_ClientSocket.Send(byData, SocketFlags.None);
                m_ClientSocket.Shutdown(SocketShutdown.Both);
                m_ClientSocket.Close();
                CloseConnection_button.Enabled = false;
                CloseConnection_button.Text = "Connection Closed";
                Connect_button.Enabled = true;
                Connect_button.Text = "Connect";
            }
            catch // Silently handle any exceptions
            {
            }
        }

        private void Connect_button_Click(object sender, EventArgs e)
        {
            try
            {
                // Create the socket, for TCP use
                m_ClientSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
                m_ClientSocket.Blocking = true; // Socket operates in Blocking mode initially
            }
            catch // Handle any exceptions
            {
                Close_Socket_and_Exit();
            }
            try
            {
                // Get the IP address from the appropriate text box
                String szIPAddress = IP_Address_textBox.Text;
                System.Net.IPAddress DestinationIPAddress = System.Net.IPAddress.Parse(szIPAddress);

                // Get the Port number from the appropriate text box
                String szPort = SendPort_textBox.Text;
                int iPort = System.Convert.ToInt16(szPort, 10);

                // Combine Address and Port to create an Endpoint
                m_remoteEndPoint = new System.Net.IPEndPoint(DestinationIPAddress, iPort);

                m_ClientSocket.Connect(m_remoteEndPoint);
                m_ClientSocket.Blocking = false;     // Socket is now switched to Non-Blocking mode for send/ receive
activities
                Connect_button.Text = "Connected";
                Connect_button.Enabled = false;
                IP_Address_textBox.Enabled = false;
                SendPort_textBox.Enabled = false;
                CloseConnection_button.Enabled = true;
                greenPictureBox.Enabled = true;
                bluePictureBox.Enabled = true;
                redPictureBox.Enabled = true;
                chosenpictureBox.Enabled = true;
                guessedpicturebox.Enabled = true;
                CloseConnection_button.Text = "Close Connection";
                m_CommunicationActivity_Timer.Start();  // Start the timer to perform periodic checking for received
messages
```

```
        }
        catch // Catch all exceptions
        {   // If an exception occurs, display an error message
            Connect_button.Text = "(Connect attempt failed)\nRetry Connect";
        }
    }


    private void pinkpictureBox_Click(object sender, EventArgs e)
    {
        chosenpictureBox.Image = pinkpictureBox.Image;

        try
        {
            String szData = "Pink"; //chosen colour to select
            chosencolour = szData; //sent to socket

            byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
            m_ClientSocket.Send(byData, SocketFlags.None);
        }
        catch // Silently handle any exceptions
        {
        }
    }

    private void brownpictureBox_Click(object sender, EventArgs e)
    {
        chosenpictureBox.Image = brownpictureBox.Image;
        try
        {
            String szData = "Brown"; //chosen colour to select
            chosencolour = szData; //sent to socket

            byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
            m_ClientSocket.Send(byData, SocketFlags.None);
        }
        catch // Silently handle any exceptions
        {
        }
    }

    private void bluePictureBox_Click(object sender, EventArgs e)
    {

        chosenpictureBox.Image = bluePictureBox.Image;

        try
        {
            String szData = "Blue"; //chosen colour to select
            chosencolour = szData; //sent to socket

            byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
            m_ClientSocket.Send(byData, SocketFlags.None);
        }
        catch // Silently handle any exceptions
        {
        }
    }

    private void yellowpictureBox_Click(object sender, EventArgs e)
```

```
        {
            chosenpictureBox.Image = yellowpictureBox.Image;

            try
            {
                String szData = "Yellow"; //chosen colour to select
                chosencolour = szData; //sent to socket

                byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
                m_ClientSocket.Send(byData, SocketFlags.None);
            }
            catch // Silently handle any exceptions
            {
            }
        }

        private void button3_Click(object sender, EventArgs e)
        {
            try
            {
                String szData = textBox1.Text;
                listBox2.Items.Add(szData);
                byte[] byData = System.Text.Encoding.ASCII.GetBytes(szData);
                m_ClientSocket.Send(byData, SocketFlags.None);
            }
            catch // Silently handle any exceptions
            {
            }
            try
            {
                EndPoint RemoteEndPoint = (EndPoint)m_remoteEndPoint;
                byte[] ReceiveBuffer = new byte[1024];
                int iReceiveByteCount;
                iReceiveByteCount = m_ClientSocket.ReceiveFrom(ReceiveBuffer, ref RemoteEndPoint);
                string szReceivedMessage; // received answer from server
                if (0 < iReceiveByteCount)
                {   // Copy the number of bytes received, from the message buffer to the text control
                    szReceivedMessage = Encoding.ASCII.GetString(ReceiveBuffer, 0, iReceiveByteCount);
                    listBox2.Items.Add(szReceivedMessage);
                }
            }
            catch
            {
            }
        }
    }
}
```