Bringing together EagleEyes users: Creating TCP Peer-to-Peer programs for two person interaction over the internet

Sharif Tai
Computer Science Thesis
April 25, 2003

Advisor: James Gips, Professor, John R. and Pamela C. Egan Chair, Computer Science Department

Table of Contents

Introduction

How could I get EagleEyes users to connect to each other and be able to do activities together? Previously when using EagleEyes, the user was constrained to games or programs that had to be run locally, with no interaction with other users. They could use internet games, but these were not designed with the limitations of the EagleEyes system in mind. Often, the action was confined in a very small area, or there were actions needed like click-and-drag, or the user could click on too many things that were not related to the program (such as ads). To improve on this process, I designed and created a series of games specifically for EagleEyes, called EagleEyes Connect. This would allow two EagleEyes users to connect to each other from anywhere they could get internet access, and play games with each other online. This is also a departure from other EagleEyes games that could possibly be played with another person in the same room, whereby the EagleEyes user makes their choice, the control is given to the other student, and they make a choice. Using this EagleEyes Connect system, both students can now use their own EagleEyes system to play against each other, and they do not have to be in the same country, let alone the same room. This makes it fair for both users because they are both using EagleEyes to make their choices. Also, it gives them the opportunity to compete against their peers for the first time.

The intended users of this system would be students at the campus school who are proficient at using EagleEyes, and are looking for new opportunities to use their system. Hopefully, as more and more people have access to the EagleEyes system, they will look to EagleEyes Connect as a way to interact with students their own age and ability. They will be able to play with users in other countries and build a community of their peers.

How did I create EagleEyes Connect?

To make this program, the users first create a TCP connection between the two computers. I chose TCP instead of UDP because I needed to ensure that the two computers were in sync – if one user launched a game and the other user did not get the message, then the system would not work. As for the network architecture, I chose to use a peer-to-peer setup in that the users connect directly to each other. One user does play the host in the sense that they have the other person connect to them, but there is no central server they both connect to. The programs, though, can easily be modified to run with a central server and satellite clients that connect to it. The reason for choosing peer-to-peer was all of the programs are made for two players, the lack of availability of a server, and the low number of people using the system, so it is unnecessary to dedicate a server for this purpose.

Problems Encountered

This approach, though, was initially more difficult for the users on the system to make the initial connection. The problem is because there is no central server, the connection must be created based on the IP address of the user's computer. Because of the dynamic nature of IP addresses, you cannot store the address of a person you connect to often because it could change each time they boot up their computer. Also, the IP address of the host must be sent to the client before the connection is made (via email, an instant message, or even over the phone). Some users were initially confused by this idea, but once they realize the idea of looking at the IP address like a phone number, they were able to connect with ease

Another initial problem was connecting through the firewall, but I was able to solve this (after consultation with the IT staff) by ensuring that the port used it over 1024. Unfortunately, due to an error, the port number 2000 was hard-coded into the program. This still allowed the users to connect to each other, but if they closed the connection, then tried to reconnect quickly, the port would be blocked. This is because the port can remain open for several minutes waiting for more data to be sent over. The user could specify a port number to connect on, but this lead to more confusion on the user's part. Once the reference to the specific port 2000 was removed, though, this problem was solved because then the operating system could choose any available port, thus ensuring the port it chose was always open, and freeing the user of ever having to manually enter a port number.

The program creates a TCP connection between the two users, which sends the multiplayer as a stream of bytes. This caused some problems as large amounts of data were sent and the receiving program could not keep up with the incoming stream. This problem was solved by placing a delimiter at the end of each stream sent (a new line character), then receiving the entire stream as a string, and then parsing the string based on the new line characters.

What programs constitute EagleEyes connect?

The main interface of the EagleEyes Connect system is a chat program chat allows the users to communicate with each other before they begin playing the games. The programs themselves are somewhat simple, but this is necessary because of the use with the EagleEyes system. I have created multiplayer Tic Tac Toe, Pong, Connect 4, Othello, and Aliens. The Aliens program was one of the first games created for EagleEyes, and it consists of the user trying to get the cursor over a randomly placed alien on the screen. In the two-player version, each user has a cursor on the screen (they can see the other player moving) and they are competing to see who can hit the alien first.

To simplify the process of distribution, I have also created an automatic update application. This allows the users to upgrade to the latest version without doing any work (it is totally transparent to the user). This upgrade system downloads a text file from the internet containing the latest versions numbers available. Using the contents of that file, it compares the latest version numbers with the ones currently on their system. If there is a later version available, the update program automatically downloads the new file and replaces the old file. Then, a short text file pops up explaining the new changes. This system allows the user to be kept constantly up to date without having to constantly download files and replace old ones themselves, as it is run automatically every two weeks without any intervention.

The system is also highly expandable. Additional programs can be written in any language, as long as it can run on the users' system. Additionally, the next system designer does not have to have any knowledge of how I created the system, as long as they follow simple guidelines for upgrading the system. How they choose to implement these new programs is entirely up to them. If a new program is made, all that has to be added is a button to the main ChatClient program to launch the new program, and then it can run on both systems. Then, the new files can easily be uploaded to the web, and each individual user will then grab these files when they do their next update. This allows the system to continue to expand and include new users and programs on a consistent basis.

User Experience using EagleEyes Connect

To date, the system has been tested successfully with several EagleEyes users. In particular, I have used it with Matt Galligan, a 19 year old student with cerebral palsy who is long time user of the EagleEyes electrode system. He has been using EagleEyes Connect since its infancy, and has been instrumental in its development. He mainly uses the multiplayer system against some of his buddies, who use a regular mouse to control the second computer. The reason for him playing against non-EagleEyes users is due to the difficulty of having two EagleEyes users available at the same time. The benefit of this is that he gets to play against (and often beat) able-bodied students who he normally would not have been able to interact with via EagleEyes. The downside of this is that it is certainly easier for someone using the mouse to control their movements then someone on EagleEyes, so it does not provide an accurate account of how the system works solely using EagleEyes. To test the system with two EagleEyes users, we connected with the Hollybank School in England. We were able to successfully have two users in different countries connect and play against each other. The Hollybank School uses the system with some of their students as a reward for doing work during the day, with some of them using the electrodes, and others on the CameraMouse. They have several students who regularly use it, and each has their own favorite game to play. Andrew (9 years old) and Alex (11) have a joint session where they use the EagleEyes Connect system using CameraMouse. They have both been on the system for 15 months. Dave, a 40 year old car accident victim uses the CameraMouse as well to play against his teachers. Michael, a 19 year old student with muscular dystrophy, takes time off from writing a book and relaxes by playing multiplayer aliens or Connect 4. Sam, another 19 year old, has played Matt on several occasions, usually using his favorite game, Aliens. The Hollybank school is also looking to play against another local school, Turnshaws, which also uses the CameraMouse. The school likes to use the system against people over the internet, or even just two EagleEyes users in the same room.

Where to go from here

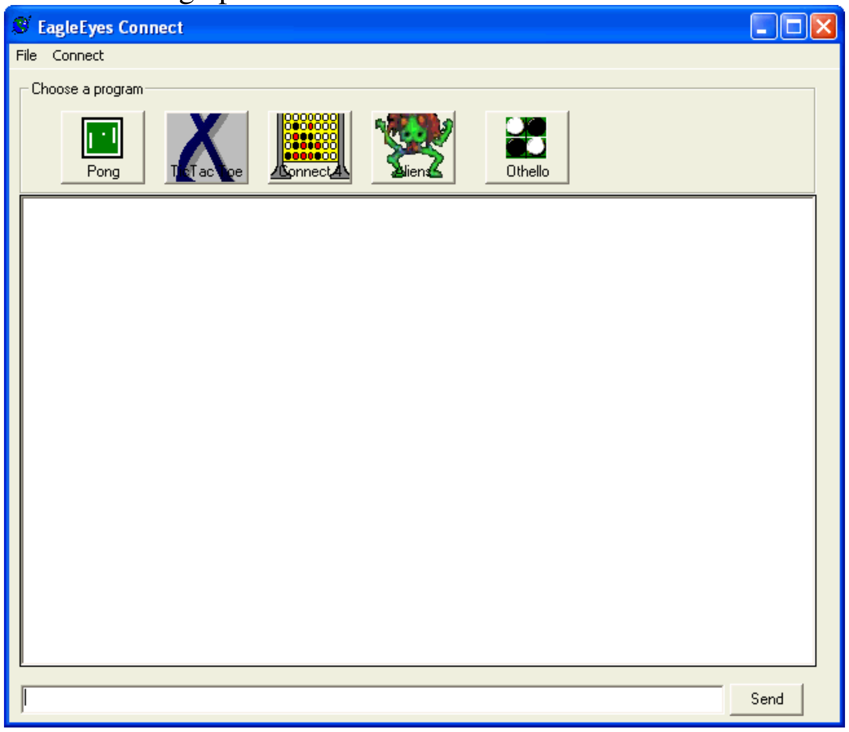
In the future, I hope that more students can continue to add on to the EagleEyes Connect system. This can be easily accomplished because one of the key tenets I kept in mind while designing it was its easy expandability. New programs can be written in any language without having to study my code in great depth. A program can be coded, and simply linked to the current applications with little effort, and then the update program will automatically grab the new files on the users systems. The update program makes it very simple to keep all users up to date. It is also an easy way to spread the programs; the user only has to have this one small file, run it once, and they will have all of the pertinent files right away. Also, it may be possible to expand the number of users to such a degree that a central server makes sense. This would allow the many users to easily check if there is someone online to play against without having to prearrange a meeting time. Hopefully, as the programs are rolled out to more EagleEyes users, the opportunity to connect to each other will grow, and more people will use the system.

EagleEyes Connect Multiplayer Instructions

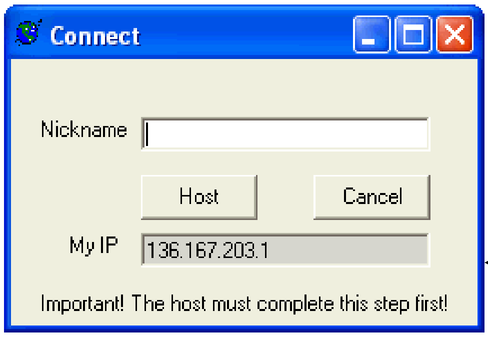
EagleEyes Connect allows two users of the EagleEyes system to connect, chat and play games against each other. Before you begin, you have to decide which user will be the host and which will connect. There is no difference between the two, except during the initial setup.

To begin, click on the ChatClient icon: ChatClient Boston College

This will bring up the main screen:



Now the host will have to select the "Connect" menu and select "Host." This will bring up the following screen:



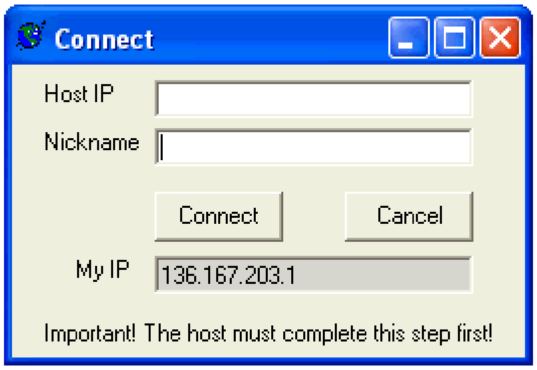
My IP 136.167.203.1

Important! The host must complete this step first!

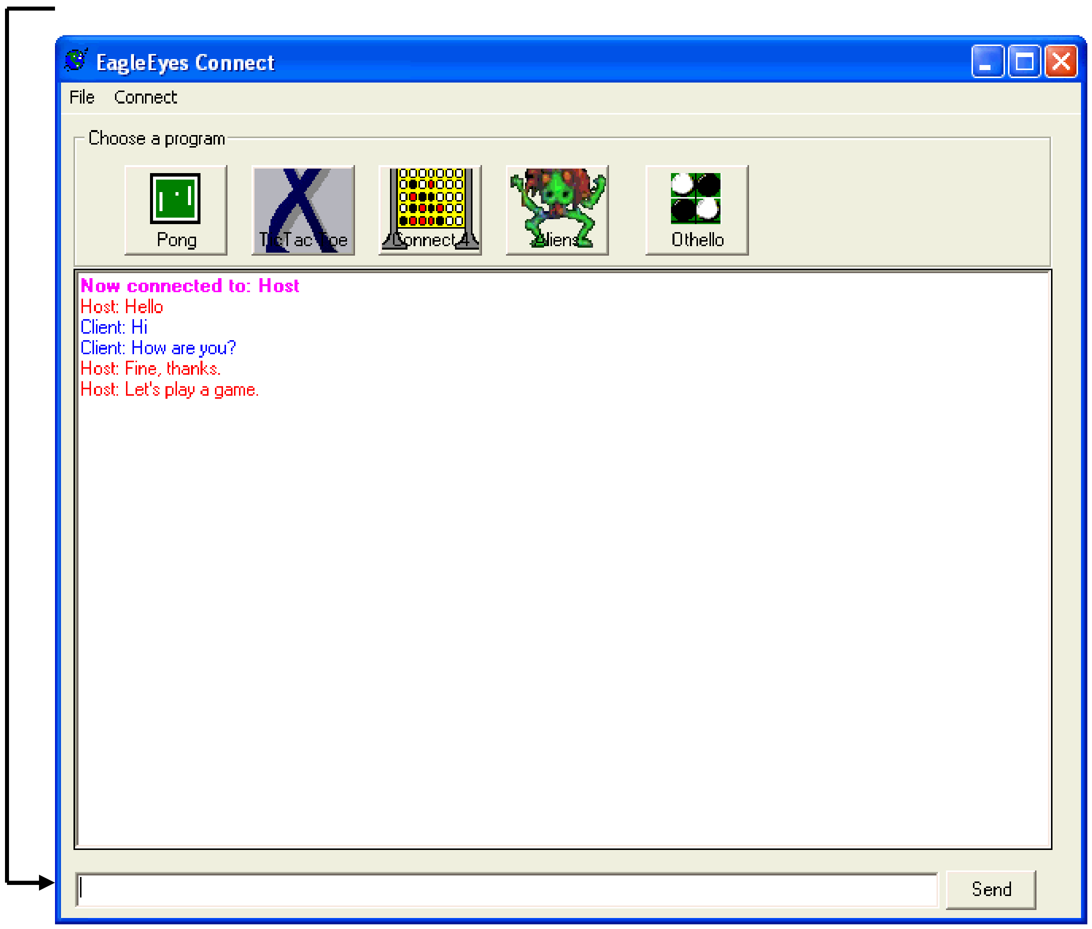
This IP address will be different for each user

The host must give their IP address to the other user. You can copy and paste this address into an email or instant message and send it to the other user. Once this is done, enter your nickname and click the "Host" button.

The client (the person who will be connecting to the host) will then click on the "Connect" menu and then "Connect." This brings up the following screen:



Cut and paste the IP address the host sent you in the previous step into the "Host IP" field and enter your nickname and click the "Connect" button. If everything is set up correctly, both users should see a message informing you that you are now connected. Once the connection is made, you can now start chatting or playing games. To talk to the other user, type in the box at the bottom and clicking "Send"

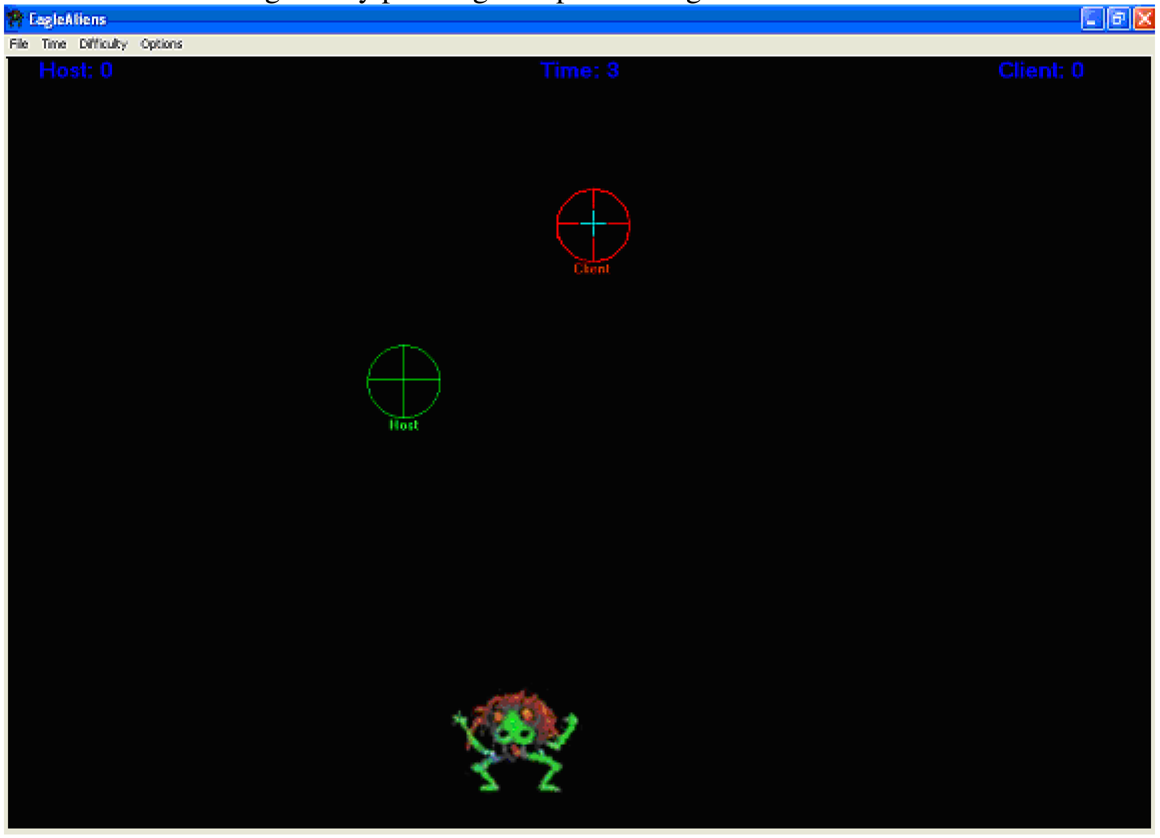


Now connected to: Host
Host: Hello
Client: Hi
Client: How are you?
Host: Fine, thanks.
Host: Let's play a game.

Either user can now click on one of the buttons to launch a game, which will bring the selected program up on each user's computer. The following section explains how to play the games. For all games (except Aliens) the person who goes first rotates.

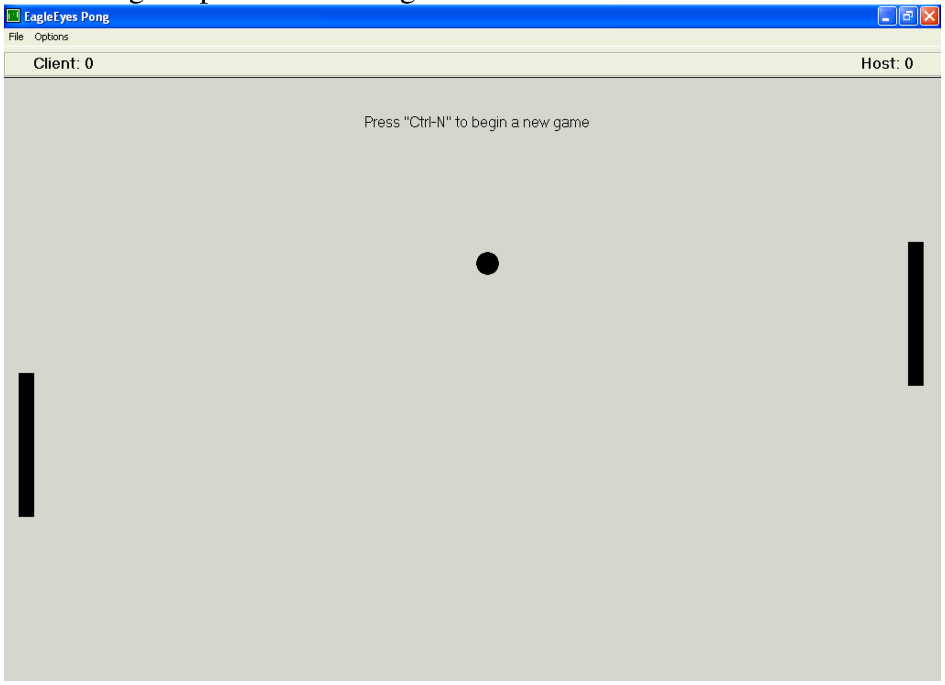
Aliens

Two player Aliens is very similar to single player aliens, except that there will be two crosshairs on the screen (you will see a label telling you which cursor is yours). Also, 20 aliens will be displayed, instead of only 10. When multiplayer aliens is launched, you will see the same startup screen. When either side presses the space bar, the game will start, just as in a single player game. You will always be the red crosshair on your screen. Once the game is started, the alien will pop up in a random position, and both people have to try and hit the alien. Whoever gets there first will get a point. After 20 aliens have been shown, you will see a summary of the scores for both people. Either user can start a new game by pressing the space bar again.



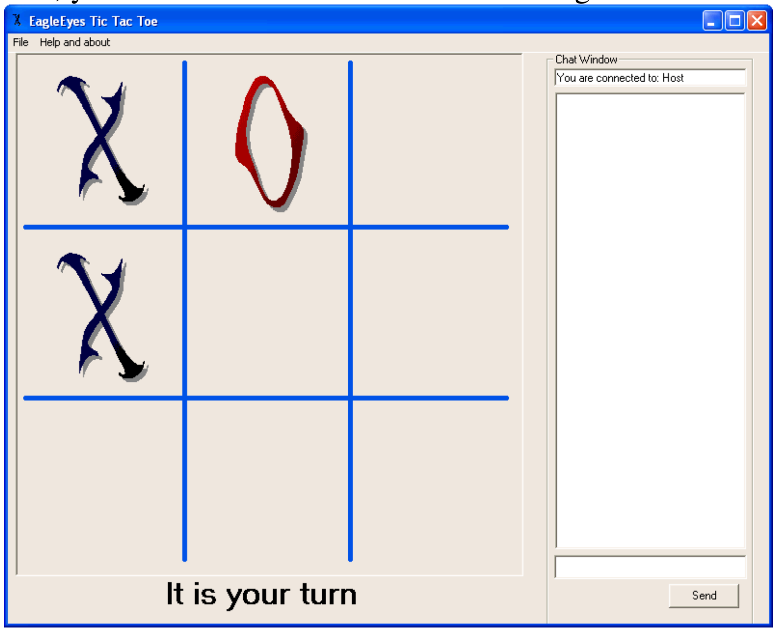
Pong

Pong is the classic game where you use your paddle and try to deflect the bouncing ball away from your side. If it gets past you, the other person gets a point. The game is usually played up to 5 points. Press “Ctrl-N” to start a new game. You will always be controlling the paddle on the right.



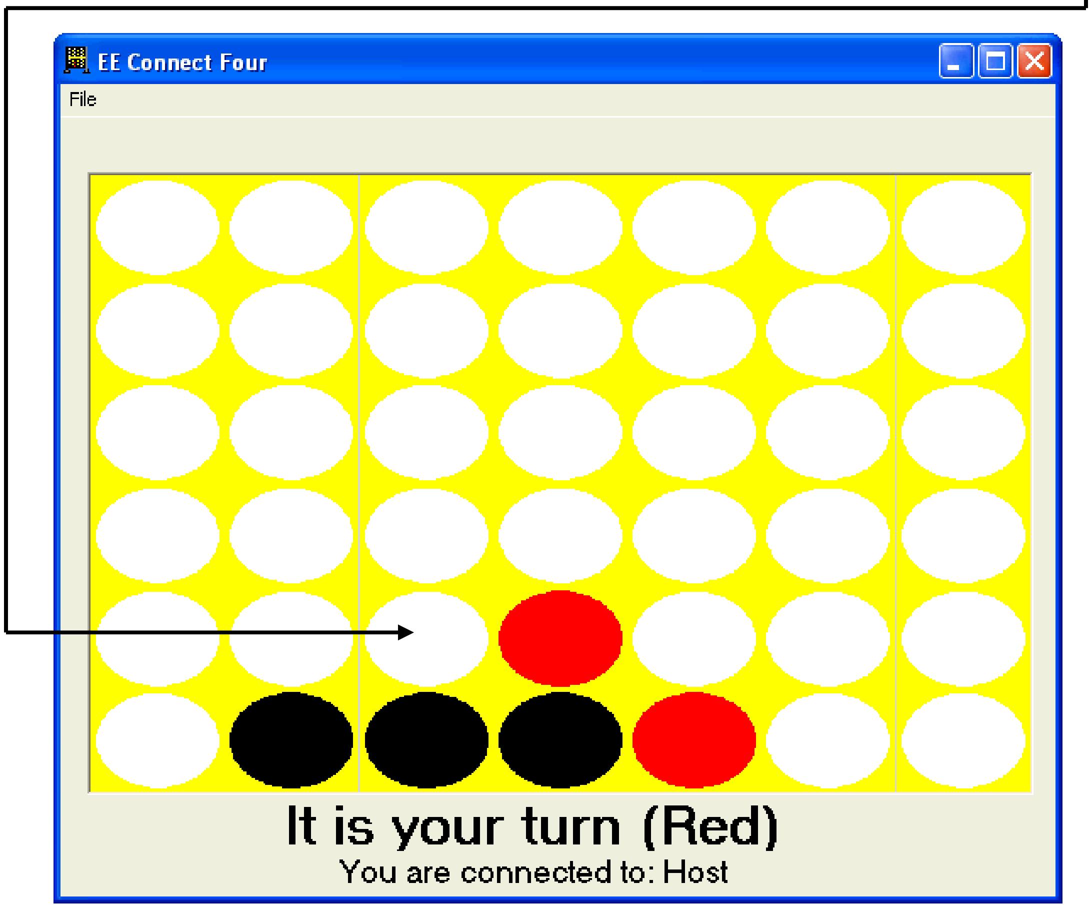
Tic Tac Toe

This is a basic game of Tic Tac Toe. You can be either X or O, and if you get three in a row, you win. You can use the box on the right hand side to chat with the other player.



Connect 4

In Connect 4, you try to get 4 pieces of your color in a row. To play, you click on any column, and the piece will fall to the lowest point in that column. Where you click vertically does not matter in this game. For instance, if the Red player clicked in the third column from the left, their piece would fall into the place above the black piece.

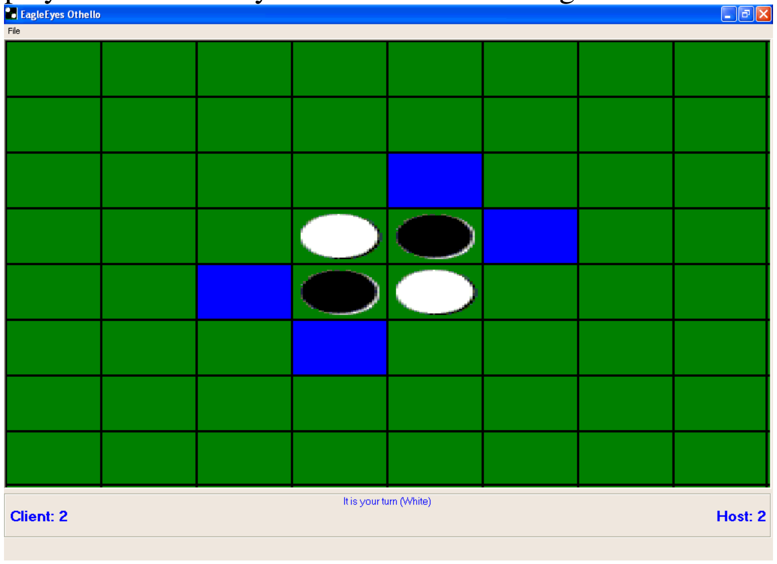


It is your turn (Red)

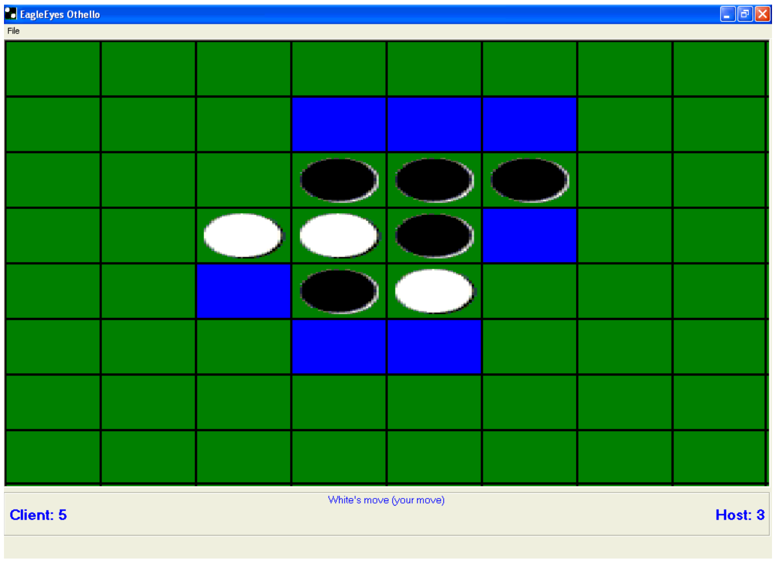
You are connected to: Host

Othello

This is a game where you try to take over the board with your color (you can be white or black). You do this by clicking on a square that sandwich any number of the other player's color with yours. Here is how the game starts out:



The player who turn it is will see blue squares indicating valid moves, while the other player will not see any blue squares until it is their turn. Because this is white's turn, they have four options available. By clicking on any of those squares, all the black pieces in the middle will turn white, and the score will change to reflect the number of both colors remaining on the board. The score during the game is not important, though, as it only matters who has more pieces at the end of the game. Here is how a sample game might look after several moves:



There may be a situation where a player cannot make a move. Then the turn goes back to the first player. Therefore, a player may have the chance to go two or more times in a row, depending on how the game is played.

Coding conventions used in EagleEyes Connect

I used the same coding ideas throughout the programs to make them similar and easier to understand. This also makes it easy for new programs to be added. For instance, there needs to be certain global variables which are usually declared in a module. The module looks like this:

```
Global Connected As Boolean, GameInProgress As Boolean
Global AmTheHost As Boolean, MyName As String, TheirName As String
Global TheirScore as Integer, MyScore as Integer
Global RecvBuffer As String 'stores the incoming winsock data
Global MultiGame As Boolean
```

Regardless of whether or not the game is being played over the network, I use these variables to keep track of things like the score and names. For instance, if someone is playing against the computer, then TheirName might be set to “Computer.” This makes it easy to write the code once, and not have to modify it when you want to add multiplayer capability. Also, in the (General) (Declarations) section, I define a Boolean variable DebugState. This is useful when debugging because it allows you to add message boxes that you wouldn’t want a normal user seeing, but then you do not have to hunt down every MsgBox statement. Instead, you can simply add a line like this: If DebugState = True then MsgBox(“Some message”). In the Form_Load sub, you can set DebugState = True when debugging, then simply make it false when compiling the program for release, and the message boxes will not pop up.

Compiling EagleEyes Connect

All of the EagleEyes Connect programs are witten in Visual Basic 6.0 (SP 4). These have not been tested or compiled on any later version such as .Net studio. I do not know if these will work if tested on .Net, or if they can be upgraded using the upgrade wizard, but I recommend maintaining these programs using Visual Basic 6.0. As for adding other programs, the new programs can be written in any language, be it .Net, Java, C++ or any thing else as long as it will run on the client’s computer. As you will see below, this only launches external programs (they do not even have to be ones you have written – if you find a useful program, you can link to it with the ChatClient program), so those programs do not have to be similar to these EagleEyes Connect programs. As for compiling the EagleEyes connect programs, you must maintain some procedure for updating the version numbers, as the update program will use these. You will see when you compile one of the programs, when you click on Options, you can change the version numbers. I tend to not change the Major and Minor numbers very often, but I change the Revision number more frequently. Also, if I am just testing a program and making several changes, I will change the fourth number with each compile. This number is not used for anything other then keeping track of how many times I actually compiled it, and you can ignore it. Thus, you will see a number like 1.2.3.18 where 1 is the major version number, 2 is the minor version number, 3 is the revision, and 18 is the build number. I also include the date in the comments, but this too is only for bookkeeping use, not the update program.

The program must also save its current version so the update program can easily find it. To do so, in the Form_Load sub, insert the following line:

```
SaveSetting "EE Connect", "Versions", "App Name", App.Major & "." & App.Minor & "." & App.Revision
'write the current version
```

This adds a line to the registry with the current Major, Minor, and Revision version numbers.

Where to put the applications

All of the applications must be in the same directory – the ChatClient program will attempt to launch the programs by simply taking its path, and appending the name of the program. Also, this structure is important for uploading the files to the web, as you will see later. If they are not in the same folder, then the process gets more complicated. The structure of the folders is: {root folder}\EE Connect\Update\Files.

The executable files must be kept in the “Files” folder, while the “Versions.txt” and “Changes.txt” (see the Update Program section for an explanation of these files) files must be kept in the “Update” folder. The source code for the applications can be stored anywhere, as long as the executables are placed in the proper folder, and the two text files are maintained as well.

Adding on to EagleEyes Connect – ChatClient steps

There are several steps that need to be completed to add another program to the EagleEyes connect program. First, add a button to the ChatClient program. Simply copy one of the other buttons (such as Aliens or Pong) because the buttons are in an array, and they need to work together. Then change the title and picture to correspond to the new program. In the code for the ChatClient program, in the sub StartProgram, you will see the following code:

```
Sub StartProgram(Index As Integer)
Dim Arguments As String, IP
Dim x As Long, LocalAppPath As String

If AmTheHost = True Then
   IP = Winsock.LocalIP
Else
   IP = Winsock.RemoteHost
End If
Arguments = IP & "," & EEProgramPort & "," & AmTheHost & "," & MyName & "," & TheirName & ","

Select Case Index
Case 0
   LocalAppPath = App.Path & "\Pong.exe"
Case 1
   LocalAppPath = App.Path & "\Tic Tac Toe.exe"
Case 2
   LocalAppPath = App.Path & "\Connect 4.exe"
Case 3
   LocalAppPath = App.Path & "\Aliens.exe"
Case 4
   LocalAppPath = App.Path & "\Othello.exe"
End Select

LocalAppPath = LocalAppPath & " " & Arguments
x = Shell(LocalAppPath, vbNormalFocus)
End Sub
```

As you can see, the programs are each launched with the same set of arguments. Therefore, all you need to do is add another case for the new program, and set the LocalAppPath variable to point to your new program. When the button is clicked, it will put together the arguments to be passed to the program, then launch it on each computer, using the Shell command. As for the arguments, each program that is launched gets the same arguments in the same order. You will see later how to parse this information. The first argument is the IP (this really only matters for the client to connect to, not the host). The next is the port number to communicate on. The third argument is a Boolean variable stating if you are the host or not, while the last two are your nickname, and the nickname of the person you are connected to.

Adding on to EagleEyes Connect – new program steps

Once you have put in the functionality into the ChatClient, you can now begin to add the network ability to your program. There are several steps to this. First, you must parse the command line arguments that were passed to the program via the ChatClient. You can simply copy the code below, which will handle splitting up the arguments into their proper elements. In the Form_Load sub, add the line: Call ProcessCommandLine(Command). This passes the command line arguments to the ProcessCommandLine sub, which is shown here:

```
Sub ProcessCommandLine(Args As String)
Dim CurArg As String, NextComma As Long, Port As Integer

NextComma = InStr(1, Args, ",")  'finds arguments seperated by spaces
If (NextComma = 0) Then
    MultiGame = False
    MyName = "Right"
    TheirName = "Left"
    Exit Sub 'there is nothing left in the buffer
Else
    'there are command arguments, so this means they are connecting via EE Connect
    'the arguments are:  IP to connect to, port to connect on, a boolean if they are
    'the host, your name, their name.  Each argument is seperated by a space, so
    'first parse the arguments, then start the multiplayer game
    MultiGame = True
    NextComma = InStr(1, Args, ",", vbTextCompare) - 1 'find the next comma
    CurArg = Left(Args, NextComma) 'the buffer up the delimeter
    Winsock.RemoteHost = CurArg
    If DebugState Then MsgBox "Winsock.remotehost=" & CurArg

    Args = Mid(Args, NextComma + 2, Len(Args))
    NextComma = InStr(1, Args, ",", vbTextCompare) - 1 'find the next comma
    CurArg = Left(Args, NextComma) 'the buffer up the delimeter
    Port = Val(CurArg)
    If DebugState Then MsgBox "Port=" & CurArg

    Args = Mid(Args, NextComma + 2, Len(Args))
    NextComma = InStr(1, Args, ",", vbTextCompare) - 1 'find the next comma
    CurArg = Left(Args, NextComma) 'the buffer up the delimeter
    If CurArg = True Then 'this is the host computer
        Winsock.LocalPort = Port
        Winsock.Listen
        AmTheHost = True
        If DebugState Then MsgBox "Winsock.Listen(hosting)"
    Else 'this is the connecting computer
        Winsock.RemotePort = Port
        Winsock.Connect
        AmTheHost = False
        If DebugState Then MsgBox "Winsock.Connect(connecting)"
    End If
```

```
Args = Mid(Args, NextComma + 2, Len(Args))
   NextComma = InStr(1, Args, ",", vbTextCompare) - 1 'find the next comma
   CurArg = Left(Args, NextComma) 'the buffer up the delimeter
   MyName = CurArg
   If DebugState Then MsgBox "MyName=" & MyName

   Args = Mid(Args, NextComma + 2, Len(Args))
   NextComma = InStr(1, Args, ",", vbTextCompare) - 1 'find the next comma
   CurArg = Left(Args, NextComma) 'the buffer up the delimeter
   TheirName = CurArg
   If DebugState Then MsgBox "TheirName=" & TheirName

   If DebugState Then FormPong.Caption = FormPong.Caption & MyName

   lblLeft.Caption = TheirName & ": 0"
   lblRight.Caption = MyName & ": 0"

   NumOfPlayers = 2
End If
End Sub
```

The majority of this code will remain the same for each program that is added, as all it is doing is creating a connection between the host and client programs and setting variables letting each program know what they are.

Parsing the Winsock Data

The following code is standard for all programs (assuming the winsock control is names Winsock, or else change the sub names accordingly):

```
Private Sub Winsock_Close()
If Connected = True Then Winsock.Close
lblInfo = TheirName & " has closed the program"
Connected = False
RecvBuffer = ""
End Sub
```

Private Sub Winsock_Close()

Closes the connection upon the exit of the other user

```
Private Sub Winsock_Connect()
RecvBuffer = ""
Connected = True
End Sub
```

Private Sub Winsock_Connect()

Clears the buffer that collects the received data, informs the program you are connected

```
Private Sub Winsock_ConnectionRequest(ByVal requestID As Long)
'this is the host computer
Dim lpKeyName As String, FileData As String, x As Integer

'this is the host computer
Winsock.Close
Winsock.Accept requestID
'cmdSend.Enabled = True
Connected = True
End Sub
```

Private Sub Winsock_ConnectionRequest (ByVal requestID As Long)

Makes the connection between the two programs

```
Private Sub Winsock_Error(ByVal Number As Integer,
 Description As String, ByVal Scode As Long,
ByVal Source As String, ByVal HelpFile As String,
ByVal HelpContext As Long, CancelDisplay As Boolean)

MsgBox "Error " & Number & " " & Description,
vbExclamation, "App Name"
Call Winsock_Close
End Sub
```

Private Sub Winsock_Error(Arguments)

Informs the user of the error and closes the connection

The above code simply makes the connection between the two programs (which will happen as soon as the ProcessCommandLine sub is called, as that does the Winsock.Listen or Winsock.Connect calls that initiate this code). The following code is how to deal with received data. It is broken into two parts because one sub receives the data, while another parses and acts on it. The data has to be parsed because of the nature of the connection. It is a TCP connection, which means the network data is not sent as discrete packets, but rather a stream of bytes. This bring up a very important point:

When sending data, you must append the constant vbNullChar to each line. If not, then if there is a large amount of data being sent, the program will not know how to break up the stream and will end up dropping some of the data.

Therefore, if you want to send something, the code should look like this (check if there is a connection first, or else it will fail when not playing over the network):

```
If Connected = True then Winsock.SendData("ID" & data & vbNullChar)
```

The ID part that is sent is how the program decides what the data actually means. This is a 4 letter long code that will be used to call specific sub routines or functions. These codes are up to you, as is what you do upon receiving one of them, as you will see in the following code. The following sub routine is still a Winsock sub, and it simply receives the data in a stream, breaks it up on the null character, and sends it along to the sub ProcessWinsockData to be further processed. This code should not have to change for any program.

```
Private Sub Winsock_DataArrival(ByVal bytesTotal As Long)
Dim SentText As String, NextSpace As Long

Winsock.GetData SentText

RecvBuffer = RecvBuffer & SentText
Do
   NextSpace = InStr(1, RecvBuffer, vbNullChar)
   If (NextSpace = 0) Then Exit Sub 'there is nothing left in the buffer
   SentText = Left(RecvBuffer, NextSpace - 1) 'the buffer up the delimeter
   RecvBuffer = Mid(RecvBuffer, NextSpace + 1)
   ProcessWinsockData (SentText)
Loop
End Sub
```

Processing the Winsock Data

Once the data is split into the individual lines that the other user sent, you can begin to process them. The following code is from the Pong program, but this can be changed and adapted to suit the new program. Obviously, there will be a lot of individuality in each program, but the process remains the same.

```
Sub ProcessWinsockData(SentData As String)
Dim WhatType As String, ActualText, x As Integer

WhatType = Mid(SentData, 1, 4)
ActualText = Right(SentData, Len(SentData) - 4)
Select Case WhatType
   Case "Chat"
      MsgBox ActualText
   Case "Move"  'where to move their paddle
      LPad.Top = Val(ActualText) * FormPong.ScaleHeight 'multiply by percentage
   Case "BTop"  'where the top of the ball is
      Ball.Top = Val(ActualText) * FormPong.ScaleHeight
   Case "BLef" 'where the left of the ball is
      Ball.Left = (1 - Val(ActualText)) * FormPong.ScaleWidth
   Case "Scor" 'update the scores
      TheirScore = Val(Mid(ActualText, 1, 1)) 'their score
      MyScore = Val(Mid(ActualText, 2, 1)) 'my score
      lblLeft.Caption = TheirName & ": " & TheirScore
      lblRight.Caption = MyName & ": " & MyScore
   Case "Wins" 'someone won
      lblWin = ActualText & " is the winner!!!"
      lblWin.Visible = True
      'MsgBox ActualText & " wins!"
   Case "NewG"
      lblInfo = TheirName & " has started a new game"
      If AmTheHost Then Call NewBall
End Select
End Sub
```

As you can see, the incoming data is split into its identifier, and the actual data. Using a Select Case statement, you choose what type of data it is, then act accordingly. For instance, in this program, when the identifier is “Move”, it updates the position of their paddle. When the identifier is “Scor”, it updates the score. Add or subtract identifiers and actions as is needed.

After completing these steps, you should be ready to add another program to the EagleEyes Connect collection. That brings us to the next important (and final) step – uploading the programs onto the web so other users can easily download them.

EagleEyes Connect Update program

The update program will compare the version that is on the users computer (via the number written to the registry using the code given above) with the most recent available version available on the web, and if needed, download the newer file. Therefore, you must maintain a file of the current version numbers. This is done in the file called “Versions.txt” in the Update folder. If you make an update to a program and change the version number, simply make the change in this file as well. If you add another program, simply add a new entry to the Programs section, put its version number into the Versions section and save the file. Here is an example of the file:

```
[Program Names]
Program1=ChatClient
Program2=Tic Tac Toe
Program3=Pong
Program4=Connect 4
Program5=Aliens
Program6=Othello
Program7=The name of your new program

[Versions]
ChatClient=1.0.5
Tic Tac Toe=1.2.5
Pong=1.1.1
Connect 4=1.0.2
Aliens=1.2.0
Othello=1.2.0
The name of your new program=1.0.0
```

Then, in the file named “Changes.txt” make a note of any changes made, new programs, or instructions. This file will be visible to the user who is upgrading their programs, so give them any information you feel is necessary. After saving these two files and compiling the new versions of all necessary programs, you can upload them to the web. I have created a script for this task. Simply run the “Upload EE Connect” file, and this will pop up a screen asking for a password (please obtain this password from Professor Gips), and will then proceed to upload the Versions and Changes files, as well as the program files. It does this by reading a script file called ftpEEApps.txt, listed here:

```
cd Web/EEConnect/
put Versions.txt
put Changes.txt
cd Files
lcd Files
put Aliens.exe
put Othello.exe
put ChatClient.exe
put Pong.exe
put "Connect 4.exe"
put "EE Update.exe"
put "Tic Tac Toe.exe"
put "Your Program Name.exe"
quit
```

If you add a new program, simply add the filename to the script so it will be uploaded to the web.