

# Net Game Development by P148551

*by* Andrew Ridgeway

---

|                |                                                                                       |                 |       |
|----------------|---------------------------------------------------------------------------------------|-----------------|-------|
| FILE           | 4150_ANDREW_RIDGEWAY_NET_GAME_DEVELOPMENT_BY_P148551_178155_2103797039.DOCX (280.16K) |                 |       |
| TIME SUBMITTED | 05-APR-2019 04:56PM (UTC+0100)                                                        | WORD COUNT      | 4027  |
| SUBMISSION ID  | 104225439                                                                             | CHARACTER COUNT | 24142 |



---

# Net Game Development

---



By Andrew Ridgeway  
P148551

L6 - Network Multiplayer Games (M6X01105)

## Table of Contents

|                     |    |
|---------------------|----|
| Introduction .....  | 2  |
| Design .....        | 2  |
| Techniques .....    | 18 |
| What is UDP .....   | 18 |
| What is TCP .....   | 18 |
| Testing Table ..... | 20 |
| References .....    | 21 |

## Table of Figures

|                                                                                |   |
|--------------------------------------------------------------------------------|---|
| Figure 1 - Here's a screenshot of the header of hangman. ....                  | 3 |
| Figure 2 - Here's a screenshot the setting of values for other functions. .... | 3 |
| Figure 3 - Here's a screenshot of the draw hangman. ....                       | 3 |
| Figure 4 -Here's a screenshot of the input words. ....                         | 3 |
| Figure 5 - Here's a screenshot of the symbol display.....                      | 4 |
| Figure 6 - Here's a screenshot of the correct word.....                        | 4 |
| Figure 7 - Here's a screenshot of entering a word.....                         | 4 |
| Figure 8 - Here's a screenshot for lives. ....                                 | 5 |

## Introduction

For this brief will be speaking about the design within the UDP and TCP server and client. The technical aspect of what type of things there is used within the server and some brief information about what is TCP and UDP. There will be a testing table that will be used to test the client and server connection to each other to see if there's any problems or no problems at all.

## Design

The design of this networking project is creating a hangman game that will be able to be run in the server and for the client to play the hangman, such as the service and client will use a UDP and a TCP. The TCP server will hold a menu that will allow it to send the information for the client to see to pick one of the following choices the TCP server will hold a menu that will allow it to send the information for the client to see to pick one of the following choices, such as a start game, a help menu and endgame, the hangman game itself will be stored on the UDP server and when the end-user picks to start the game to run hangman the UDP server will send information to the client to play Hangman.

Here are some screenshots of the hangman game code that is going to be used with the networking code:

```
#pragma once
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <vector>
#include <fstream>
#include <string>
#include <time.h>

using namespace std;

// this line of code is the set-up class for hangmen
class HangMan {
private:
    vector<string> vec;

public:
    HangMan() {}
    ~HangMan() {}

    // this line of code is used for the
    // drawing checking for letters and
    // usable symbols for the hangman game
    void theMessage(string, bool, bool);
    void draw(int);
    void letter(string, char, char);
    void usableSymbols(string);

    // this line of code is used for checking
    // word input in word and tries for the hammer game
    bool wordCheck(string, string);
    string randWord();
    int tries(string, string);
};
```

Figure 1 - Here's a screenshot of the header of hangman.

Here is a screenshot showing above of the hangman games header that will holds certain functions.

```
#include "pch.h"
#include "HangMan.h"

// This line of code is used to set values in other functions
void HangMan::theMessage(string message,
                          bool value1 = true,
                          bool value2 = true) {
    bool front = true;
    if (front)
        message = " " + message;
    else
        message = message + " ";
    front = !front;
    printf(message.c_str());
}
```

Figure 2 - Here's a screenshot the setting of values for other functions.

Here is a screenshot showing above of the message that is used to set values for the other functions for the hangman.

```
// this line of code is used to draw the hanged man
// and it wrote it every time the player loses a life while
// guesses the word wrong
void HangMan::draw(int theDraw = 0) {
    if (theDraw >= 9) { ... }
    else if (theDraw >= 8) { ... }
    else if (theDraw >= 7) { ... }
    else if (theDraw >= 6) { ... }
    else if (theDraw >= 5) { ... }
    else if (theDraw >= 4) { ... }
    else if (theDraw >= 3) { ... }
    else if (theDraw >= 2) { ... }
    else if (theDraw >= 1) { ... }
}
```

Figure 3 - Here's a screenshot of the draw hangman.

Here is a screenshot showing above of the hangman being drawn every time the end-user guesses the word from and the main reason why this line of code isn't showing fully because it takes over 90 lines of code just to do the draw hangman and it's not shown fully for that reason.

```
// This line of code is used to input a symbol
void HangMan::letter(string input, char from, char to) {
    string s;
    for (int i = from; i <= to; i++) {
        if (input.find(i) == string::npos) {
            s += i;
            s += " ";
        }
    }
    theMessage(s, false, false);
}
```

Figure 4 -Here's a screenshot of the input words.

Here is a screenshot showing above of this line of code that allows the end-user to input a letter for it to be used for the hangman game

```
// This line of code is used to display the symbols on the console window
void HangMan::usableSymbols(string taken) {
    printf("\navailable word\n");
    letter(taken, 'a', 'm');
    cout << endl;
    letter(taken, 'n', 'z');
    cout << endl;
    letter(taken, 'A', 'M');
    cout << endl;
    letter(taken, 'N', 'Z');
    cout << endl;
    letter(taken, '0', '9');
}
```

Figure 5 - Here's a screenshot of the symbol display.

Here is a screenshot showing above of this line of code allow the end-user to see each letter of the alphabet and numbers for the hangman game that will be displayed on the console screen for the client to see.

```
// this line of code is used to see if this word is correct
bool HangMan::wordCheck(string word, string guess) {
    string s;
    bool won = true;
    for (int i = 0; i < word.length(); i++) {
        if (guess.find(word[i]) == string::npos) {
            won = false;
            s += "_ ";
        }
        else
            s += word[i];
    }
    theMessage(s, false);
    return won;
}
```

Figure 6 - Here's a screenshot of the correct word.

Here is a screenshot showing above of this line of code is used for the end user to allow to check if they got the word correct or wrong.

```
// this line of code is used to allow the end user to input a word
string HangMan::randWord() {
    int count = 0;
    string sword;

    printf("Enter a word.\n");
    cin >> sword;

    count = sword.size();
    return sword;
}
```

Figure 7 - Here's a screenshot of entering a word.

Here is a screenshot showing above of this line of code will allow the end user to input a word for the hanged man game, for example the end-user could type in the word hello while other people will have to guess.

```
// This line of code is used to see if this word is incorrect
int HangMan::tries(string word, string guess) {
    int wrong = 0;
    for (int i = 0; i < guess.length(); i++)
        if (word.find(guess[i]) == string::npos)
            wrong++;
    return wrong;
}
```

Figure 8 - Here's a screenshot for lives.

Here is a screenshot showing above of this line of code is used for how many tries for an end-user so we'll have before they failed the game hangman, such as they will have 9 lives, if they fail 9 times their lives will hit 0 and lose the game.

here is the Source Code of the whole server and clients shown below.

|                                  |                                                                                                                                                                                                                                                                                                                |
|----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TCP and<br>UDP<br>server<br>main | <pre>#include "pch.h" #include "ClassTcpServer.h" #include "ClassUdpServer.h"  TCPServer tcp2; UDPServer udp2;  // this line of code is used to run TCP and UDP server int main(int argc, char* argv[]) {     tcp2.ClassTCP();     udp2.ClassUDP();      // Pause     while (!_kbhit());     return 0; }</pre> |
|----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Here is the main what runs the TCP and UDP server together this is used to make sure it sent the both servers together to the client to be able to get the information.

|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| UDP<br>server<br>source | <pre>#include "pch.h" #include "ClassUdpServer.h" #pragma warning(disable : 4996) #include "HangMan.h"  MESSAGE_HANGMAN msg; HangMan hangMan1; UDPServer udps;  // this line of code that is used for multithreading DWORD WINAPI ThreadProc(LPVOID lpParameter) {     DWORD param = (DWORD)lpParameter; // Recover      int count;     // this is used for a countdown     for (count = 10; count &gt;= 0; --count) {         printf("\nCount down timer has %d\n", count);         Sleep(1000);         system("cls");     }     return 0; }</pre> |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

```

// this line of code is used to package the hangmen for the server
unsigned short PackHangman(MESSAGE_HANGMAN* msgs, unsigned char* buff) {
    unsigned short msglen;

    PackShort(buff, msgs->code);
    PackBytes(buff, msgs->hangman, MESSAGE_LENGTH);

    msglen = sizeof(unsigned short) + sizeof(msgs->hangman);
    return msglen;
}

// this line of code is used to run the hangman game
void UDPServer::mainmenu() {
    char msg[128];
    //char* choice;
    string guess1, theGuess;
    char x;
    int trie = 0;
    bool win = false;
    //cout << "game start!\n";

    messageSend(msg);
    char* choice = messageRecv();
    strcmp(choice, ""), theGuess = hangMan1.randWord();

    // this will do the do loop to run the hangman game
    do {
        system("cls");
        strcmp(choice, ""), hangMan1.usableSymbols(guess1);

        strcmp(choice, "\nGuess the word\n");
        strcmp(choice, ""), win = hangMan1.wordCheck(theGuess,
guess1);

        if (win)
            break;

        // allow the end-user to enter a word and shows an arrow
for the user
        strcmp(choice, "\n->");
        cin >> x;

        if (guess1.find(x) == string::npos)
            guess1 += x;
        trie = hangMan1.tries(theGuess, guess1);

    } while (trie < 9);

    // this is used to allow the end user to see if they won or lose
the game
    for (index = 0; index < MAXTHREADS; index++) {
        threadhdl[index] = CreateThread(NULL, 0, ThreadProc,
(LPVOID)param, 0, &threadid);
        if (win)
            printf("\nYOU WON!!!\n");
        else
            printf("\nYOU SUCC!!\n");
    }

    for (index = 0; index < 30; index++) {
        printf("[%x] cmtb=%i\r\n", GetCurrentThreadId(), index);
    }
}

```

```

        Sleep(100);
    }
    terminate_app = 1;
}

// this line of code is used to send the message to the client
void UDPServer::messageSend(char* msgs) {
    iResult = send(UdpSocket, msgs, (int)strlen(msgs) + 1, 0);
    shutdown(UdpSocket, SD_SEND);
    closesocket(UdpSocket);
}

// this line of code is used to receive the message from the client
char* UDPServer::messageRecv() {
    iResult = recvfrom(UdpSocket, recvbuf,
        recvbuflen, 0, (SOCKADDR *) &sender_addr,
        &iSenderAddrSize);

    // this is used to receive the bytes from the client
    if (iResult > 0) {
        printf("Bytes received from client: %d\n",
iSenderAddrSize);
        closesocket(UdpSocket);
        WSACleanup();
    }

    return recvbuf;
}

// this line of code is used to call all the other functions and send it
to the client
void UDPServer::ClassUDP() {
    // Do this only once at startup
    iResult = WSASStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != 0) {
        printf("WSAStartup failed: %d\n", iResult);
    }

    // this is getting the message.h details
    msg.code = MESSAGE_CODE_HANGMAN;
    msg.hangman;
    strcpy((char*)msg.hangman, "Bob");

    messageRecv();
    printf("Message being sent: code:%i\n usr:%s\n", msg.code,
msg.hangman);

    // this is used to avoid any undesired effects of optimizing
compilers
    // and to clear a block of memory
    ZeroMemory(&hints, sizeof(hints));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_DGRAM;
    hints.ai_protocol = IPPROTO_UDP;

    // this line of code is used for the server
    int iRcvdBytes = 0;

    UdpSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    rx_addr.sin_family = AF_INET;
    rx_addr.sin_port = htons((u_short)atoi(DEFAULT_PORT));
    rx_addr.sin_addr.s_addr = htonl(INADDR_ANY);

```

```

        iResult = bind(UdpSocket, (SOCKADDR *)&rx_addr, sizeof(rx_addr));

        iResult = recvfrom(UdpSocket, recvbuf,
                           recvbuflen, 0, (SOCKADDR*)&sender_addr,
                           &iSenderAddrSize);

        printf("Go Go byter ranger recieve: %d\nMsg=%s\n", iResult,
               recvbuf);

        char *sendbuf = recvbuf;
        // Number of byter received
        int sendbuflen = iResult;

        iResult = sendto(UdpSocket, sendbuf,
                         sendbuflen, 0, (SOCKADDR*)&sender_addr,
                         iSenderAddrSize);
        while (!_kbhit()) {
            printf("Google byter send: %d\n", iResult);
            messageRecv();
            mainmenu();
        }

        closesocket(UdpSocket); // Cleanup
        WSACleanup(); // Do this only once at the end
    }
}

```

Here is the whole code of the UDP server that was sent and received the message to the client to allow the client to gain information of choosing to start the hangman game or clicking on the help menu to see what help the player can get.

|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| UDP<br>server<br>header | <pre> #pragma once #include "pch.h" #pragma warning(disable : 4996) #include &lt;iostream&gt; #include &lt;ws2tcpip.h&gt; #include &lt;stdlib.h&gt; #include &lt;stdio.h&gt; #include &lt;conio.h&gt; #include &lt;time.h&gt; #include "Messages.h" #include "PackUnpack.h" using namespace std;  #define DEFAULT_PORT "27015" #define DEFAULT_BUFLen 512 #define MAXTHREADS 1  class UDPServer { private:     WSADATA wsaData;     const char *sendbuf = "this is a text";     char recvbuf[DEFAULT_BUFLen];     int iResult;     int recvbuflen = DEFAULT_BUFLen;     SOCKET UdpSocket = INVALID_SOCKET;      sockaddr_in rx_addr;     sockaddr_in sender_addr; </pre> |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

```

        int iSenderAddrSize = sizeof(sender_addr);

        struct addrinfo *addressList = NULL;
        struct addrinfo hints;
        int iRecvdBytes = 0;

        HANDLE threadhdl[MAXTHREADS];
        DWORD threadid;
        int index = 0;
        DWORD param = 2; // Some value to pass to thread

        int terminate_app = 0;

public:
    UDPServer() {}
    ~UDPServer() {}

    void mainmenu();
    void messageSend(char*);
    char* messageRecv();
    void ClassUDP();
};

```

Here is the header that holds the functions for the UDP server to allow the UDP Source Code to call these functions.

|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TCP<br>server<br>source | <pre> #include "pch.h" #include "HangMan.h" #include "ClassTcpServer.h" #include "ClassUdpServer.h" #pragma warning(disable : 4996)  UDPServer udp3; HangMan hangMan;  // this line of code is used to display the help menu to the client void TCPServer::help() {     char msg[128];     strcpy(msg, "\n\n{Ahahah, you really think I will help you?\n\n 1 - Play Hangman.\n 2 - Help.\n 3 - Exit.\n Enter your choice and press return: ");      messageSend(msg);     messageRecv(); }  // this line of code is used to allow the client to // pick three choices such as start game, help menu and endgame void TCPServer::hang_man() {     srand(clock());     bool theGame = true;     char msg[128];      bool exitLoop = false;     while (theGame != false) {         // this line of code is used to run the choice menu         do {             strcpy(msg, "\n 1 - Play Hangman.\n 2 - Help.\n 3 - Exit.\n Enter your choice and press return: "); </pre> |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

```

        messageSend(msg);
        char* choice = messageRecv();

        if (strcmp(choice, "1") == 0)
            exitLoop = true;
        if (strcmp(choice, "2") == 0)
            exitLoop = true;
        if (strcmp(choice, "3") == 0)
            exitLoop = true;

        if (strcmp(choice, "1") == 0) {
            // this is used to call from the udp to
            // allow the client to play Hangman
            udp3.mainmenu();
        }
        else if (strcmp(choice, "2") == 0) {
            // this is used to call the help menu
            help();
        }
        else if (strcmp(choice, "3") == 0) {
            strcpy(msg, "\nEnd of Program.\n");
            theGame = false;
        }
        else {
            cout << "Not a Valid Choice. \n";
            cout << "Choose again.\n";
        }
    } while (!exitLoop);

}

system("pause");
getchar();
}

// this line of code is used to send the message to the server
void TCPServer::messageSend(char* msg) {
    iResult = send(clientSocket, msg, (int)strlen(msg) + 1, 0);
    shutdown(clientSocket, SD_SEND);
    closesocket(clientSocket);
}

// this line of code is used to receive the message from the server
char* TCPServer::messageRecv() {
    // Block waiting for somebody to talk to me!!
    clientSocket = accept(listenSocket, NULL, NULL);
    iResult = recv(clientSocket, recvbuf, recvbuflen, 0);
    if (iResult > 0) {
        printf("Bytes received from client: %d msg: %s\n", iResult,
recvbuf);
        iRecvBytes = iResult;
    }
    else if (iResult == 0) {
        printf("Client closing connection\n");
    }
    else {
        printf("recv failed: %d\n", WSAGetLastError());
        WSACleanup();
    }

    return recvbuf;
}

```

```

}

// this line of code is used to call all the other functions and send it
to the client
void TCPServer::ClassTCP() {
    WSASStartup(MAKEWORD(2, 2), &wsaData);

    // this is used to avoid any undesired effects of optimizing
    compilers
    // and to clear a block of memory
    ZeroMemory(&hints, sizeof(hints));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_protocol = IPPROTO_TCP;
    hints.ai_flags = AI_PASSIVE; // Passive opens
    getaddrinfo(NULL, DEFAULT_PORT, &hints, &addressList);

    // Create a socket for connecting to the client
    listenSocket = socket(addressList->ai_family,
        addressList->ai_socktype,
        addressList->ai_protocol);

    BOOL optionValue = TRUE;
    int optionLength = sizeof(BOOL);
    setsockopt(listenSocket, SOL_SOCKET, SO_REUSEADDR,
        (char*)&optionValue, optionLength);

    // Set scket to listen on the port number provided (DEFAULT_PORT)
    bind(listenSocket, addressList->ai_addr, (int)addressList->
    ai_addrlen);
    freeaddrinfo(addressList);
    listen(listenSocket, SOMAXCONN);

    // this is used to scent the information to the client
    while (!_kbhit()) {
        printf("Listening for incoming connections\n");

        messageRecv();
        hang_man();
    }

    closesocket(listenSocket);
    WSACleanup();
}

```

Here is the whole code of the TCP server that was sent and received the message to the client to allow the client to gain information of choosing to start the hangman game or clicking on the help menu to see what help the player can get.

|                         |                                                                                                                                                                                                                                      |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TCP<br>server<br>header | <pre> #pragma once #include "pch.h" #pragma warning(disable : 4996) #include &lt;iostream&gt; #include &lt;ws2tcpip.h&gt; #include &lt;stdlib.h&gt; #include &lt;stdio.h&gt; #include &lt;conio.h&gt; #include &lt;time.h&gt; </pre> |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

```

using namespace std;

#define DEFAULT_PORT "27015"
#define DEFAULT_BUFLen 512
#define _CRT_SECURE_NO_WARNINGS 1

class TCPServer {
private:
    WSADATA wsaData;
    char recvbuf[DEFAULT_BUFLen];
    int iResult;
    int recvbuflen = DEFAULT_BUFLen;

    struct addrinfo *addressList = NULL;
    struct addrinfo *ptr = NULL;
    struct addrinfo hints;

    SOCKET listenSocket = INVALID_SOCKET;
    SOCKET clientSocket = INVALID_SOCKET;
    int iRecvBytes = 0;

public:
    TCPServer() {}
    ~TCPServer() {}

    void help(void);
    void hang_man(void);
    void messageSend(char*);
    char* messageRecv();
    void ClassTCP();
};

```

Here is the header that holds the functions for the TCP server to allow the TCP Source Code to call these functions.

|                             |                                                                                                                                                                                                                                                                                                                                                                                  |
|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Message<br>server<br>header | <pre> #pragma once #define MESSAGE_LENGTH 1000  typedef struct {     unsigned short code; //Type of message     unsigned short length; //Length of message }MESSAGE_PREAMBLE;  #define PREAMBLE_LENGTH 4 //4 bytes (2 x unsigned short)  //Code values #define MESSAGE_CODE_HANGMAN 3  typedef struct {     int code;     char hangman[MESSAGE_LENGTH]; }MESSAGE_HANGMAN; </pre> |
|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Here is the hangman message that is used to send information to the client from the server.

|                                     |                                                                                                                                                                                                                                                                                                                |
|-------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TCP<br>and<br>UDP<br>client<br>main | <pre> #include "pch.h" #include "ClassTcpClint.h" #include "ClassUdpClint.h"  TCPClient tcp2; UDPClient udp2;  // this line of code is used to run TCP and UDP client int main(int argc, char* argv[]) {     tcp2.ClassTCP();     udp2.ClassUDP();      // Pause     while (!_kbhit());     return 0; } </pre> |
|-------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Here is the main what runs the TCP and UDP client together this is used to make sure it sent the both servers together to the server to be able to get the information.

|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| UDP<br>client<br>source | <pre> #include "pch.h" #include "ClassUdpClint.h" #pragma warning(disable : 4996) #include "Messages.h" #include "PackUnpack.h"  MESSAGE_HANGMAN msg;  // this line of code is used to package the hangmen for the client unsigned short PackHangman(MESSAGE_HANGMAN* msg, unsigned char* buff) {     unsigned short msglen;     //unsigned char *startptr = buff;      PackShort(buff, msg-&gt;code);     PackBytes(buff, msg-&gt;hangman, MESSAGE_LENGTH);      msglen = sizeof(unsigned short) + sizeof(msg-&gt;hangman);     return msglen; }  // this line of code is used to send the message to the server void UDPClient::messageSend(char* msgs) {     struct addrinfo *addressList = NULL;     iResult = getaddrinfo("127.0.0.1", DEFAULT_PORT, &amp;hints, &amp;addressList);     //     if (iResult != 0) {         printf("getaddrinfo failed: %d\n", iResult);         WSACleanup();         //return 1;     }      // Create a socket for connecting to the server     UdpSocket = socket(addressList-&gt;ai_family,         addressList-&gt;ai_socktype,         addressList-&gt;ai_protocol); } </pre> |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

```

        iResult = sendto(UdpSocket, sendbuf,
            (int)strlen(sendbuf) + 1, 0,
            addressList->ai_addr, addressList->ai_addrlen);

        printf("Hello Byter sent: %Id\n", iResult);

        // Avoid memory leak
        freeaddrinfo(addressList);
    }

    // this line of code is used to receive the message from the server
    char* UDPClient::messageRecv() {
        iResult = recv(UdpSocket, recvbuf, recvbuflen, 0);
        if (iResult > 0) {
            printf("Bytes received: %d\n%s\n", iResult, recvbuf);
        }
        else if (iResult == 0) {
            printf("Connection closed by server\n");
        }
        else {
            printf("recv failed: %d\n", WSAGetLastError());
        }

        return recvbuf;
    }

    // this line of code is used to call all the other functions and send it
    to the server
    void UDPClient::ClassUDP() {
        unsigned char sendbu[1000];
        int i;
        unsigned short buflen;

        msg.code = MESSAGE_CODE_HANGMAN;
        msg.hangman;
        strcpy((char*)msg.hangman, "Bob");

        //
        messageRecv();
        printf("Message being sent: code:%i\n usr:%s\n", msg.code,
            msg.hangman);

        // this is used to avoid any undesired effects of optimizing
        compilers
        // and to clear a block of memory
        ZeroMemory(&hints, sizeof(hints));
        hints.ai_family = AF_UNSPEC;
        hints.ai_socktype = SOCK_DGRAM;
        hints.ai_protocol = IPPROTO_UDP;

        // Block waiting for server
        iResult = recvfrom(UdpSocket, recvbuf, recvbuflen, 0, NULL, NULL);
        printf("Move Byter received: %d Msg=%s\n", iResult, recvbuf);
        while (!kbhit()) {
            strcpy(recvbuf, messageRecv());

            printf("%s", recvbuf);
            char buffer[64];

            scanf("%s", buffer);

```

|  |                                                                                                                                                |
|--|------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <pre>         messageSend(buffer);     }      closesocket(UdpSocket);    // Cleanup     WSACleanup(); // Do this only once at the end } </pre> |
|--|------------------------------------------------------------------------------------------------------------------------------------------------|

here is the whole code of the udp there is used to receive the message from the server to allow to gain access of the menu within the server for the client to choose if they want to go into help menu or start game.

|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| UDP<br>client<br>header | <pre> #pragma once #include "pch.h" #pragma warning(disable : 4996) #include &lt;iostream&gt; #include &lt;ws2tcpip.h&gt; #include &lt;stdlib.h&gt; #include &lt;stdio.h&gt; #include &lt;conio.h&gt;  using namespace std;  #define DEFAULT_PORT "27015" #define DEFAULT_BUFLen 512  class UDPClient { private:     WSADATA wsaData;     const char *sendbuf = "this is a text";     char recvbuf[DEFAULT_BUFLen];     int iResult;     int recvbuflen = DEFAULT_BUFLen;     SOCKET UdpSocket = INVALID_SOCKET;      sockaddr_in rx_addr;     sockaddr_in sender_addr;     int iSenderAddrSize = sizeof(sender_addr);      struct addrinfo hints;     int iRecvBytes = 0;  public:     UDPClient() {}     ~UDPClient() {}      void messageSend(char*);     char* messageRecv();     void ClassUDP(); }; </pre> |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Here is the header that holds the functions for the UDP client to allow the UDP Source Code to call these functions.

|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TCP<br>client<br>source | <pre> #include "pch.h" #include "ClassTcpClint.h" #pragma warning(disable : 4996)  #define _CRT_SECURE_NO_WARNINGS 1  // this line of code is used to send the message to the server void TCPClint::button() {     char msg[128];      strcpy(msg, " ");     messageSend(msg); }  // this line of code is used to receive the message from the client void TCPClint::messageSend(char* msg) {     // this is used to avoid any undesired effects of optimizing     // compilers     // and to clear a block of memory     ZeroMemory(&amp;hints, sizeof(hints));     hints.ai_family = AF_UNSPEC;     hints.ai_socktype = SOCK_STREAM;     hints.ai_protocol = IPPROTO_TCP;      getaddrinfo(serverName, DEFAULT_PORT, &amp;hints, &amp;addressList);     connectSocket = socket(addressList-&gt;ai_family,         addressList-&gt;ai_socktype,         addressList-&gt;ai_protocol);      // Active open     connect(connectSocket, addressList-&gt;ai_addr, (int)addressList- &gt;ai_addrlen);     freeaddrinfo(addressList);      iResult = send(connectSocket, msg, (int)strlen(msg) + 1, 0);     printf("Bytes sent: %d Message: %s\n", iResult, msg);      shutdown(connectSocket, SD_SEND); }  // this line of code is used to receive the message from the client char* TCPClint::messageRecv() {     // Should be in a loop     // Block waiting to receive response from server     iResult = recv(connectSocket, recvbuf, recvbuflen, 0);     if (iResult &gt; 0) {         // Iresult is the number of bytes received         // Assumes NULL char is included in recvbuf         // To terminate the string         // Bytes received is %d and msg is %s         printf("Bytes received: %d\n%s\n", iResult, recvbuf);     }     else if (iResult == 0) {         printf("Connection closed by server\n");     }     else {         printf("recv failed: %d\n", WSAGetLastError());     }      return recvbuf; } </pre> |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

```

// // this line of code is used to call all the other functions and send
it to the server
void TCPClient::ClassTCP() {
    char msg[128];
    serverName = "127.0.0.1";
    strcpy(msg, "Hello");

    //
    WSASStartup(MAKEWORD(2, 2), &wsaData);
    bool start = true;
    char sendbuf[1000];

    sprintf(sendbuf, "\n%s %i", "this is a test\n", clock());

    // this line of code is used to call the function button
    if (start) {
        button();
        start = false;
    }

    // this is used to connect to a server
    while (!_kbhit()) {
        strcpy(recvbuf, messageRecv());

        printf("%s", recvbuf);
        char buffer[64];

        scanf("%s", buffer);
        messageSend(buffer);
    }

    WSACleanup();
}

```

Here is the whole code for the client to gain information from the server to allow them to information to play the hangman game and send information back to the server.

|                         |                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TCP<br>client<br>header | <pre> #pragma once #include "pch.h" #pragma warning(disable : 4996) #include &lt;iostream&gt; #include &lt;ws2tcpip.h&gt; #include &lt;stdlib.h&gt; #include &lt;stdio.h&gt; #include &lt;conio.h&gt; #include &lt;time.h&gt;  using namespace std;  #define DEFAULT_PORT "27015" #define DEFAULT_BUFLen 512 #define _CRT_SECURE_NO_WARNINGS 1  class TCPClient { private:     WSADATA wsaData; </pre> |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

```

char recvbuf[DEFAULT_BUFLen];
int iResult;
int recvbuflen = DEFAULT_BUFLen;

struct addrinfo *addressList = NULL;
struct addrinfo *ptr = NULL;
struct addrinfo hints;

const char* serverName;

SOCKET connectSocket = INVALID_SOCKET;

public:
    TCPClnt() {}
    ~TCPClnt() {}

    void button();
    void messageSend(char*);
    char* messageRecv();
    void ClassTCP();
};

```

Here is the header that holds the functions for the TCP client to allow the TCP Source Code to call these functions.

## Techniques

### What is UDP

UDP also known as user datagram protocol is an alternative that is used as an alternative communication protocol compare to transmission control protocol also known as TCP and this is mostly used for low-latency and loss-tolerating connections, this means it's used for connection between two applications that is used on the internet[1][2]. UDP has the capability to send process-to-process Communications this means it allowed to people to communicate through the internet to each other[2][3]. This protocol is most ideal for applications such as gaming voice and video communication and the main reason for that is to make sure it has to be critical for example if you're playing a shooter game you have to make sure every bullet hit your target without any latency loss[2].

### What is TCP

TCP also known as transmission control protocol and it's mostly works with the IP address or also known as internet protocol to allow any system that can connect to the internet to allow them to send packets of data to other devices, this means TCP would be mostly used to send information to another device for the other user to

have the information[4][5]. This protocol is a connection-oriented protocol type this means TCP would maintain its connection between two application programs to each other and both would keep connecting to each other until both have finished exchanging messages to each other[4][5].

For developing this network there was various techniques that was used to create the UDP and the TCP, such as the multithreading when the end user has win or lose at the hangman game there will be a timer and the multithreading will kick in within this timer for 10 seconds for the end-user to wait till they can do something.

```
// this line of code that is used for multithreading
DWORD WINAPI ThreadProc(LPVOID lpParameter) {
    DWORD param = (DWORD)lpParameter; // Recover

    int count;
    // this is used for a countdown
    for (count = 10; count >= 0; --count) {
        printf("\nCount down timer has %d\n", count);
        Sleep(1000);
        system("cls");
    }
    return 0;
}
```

Here is the multithreading code that forces the end user to wait 10 seconds to they can do something.

There is a message that has to be built to allow information to be able to be sent to the client to allow them to gain information for the game and various other stuff such as the menu.

```
// this is getting the message.h details
msg.code = MESSAGE_CODE_HANGMAN;
msg.hangman;
strcpy((char*)msg.hangman, "Bob");
```

here is the code that is used to get the message details.

## Testing Table

Here is a testing table will test the server and client connections together, the game to see if there's any problems and if there are any problems or no problems at all they will be included in the test in table.

| No. | Test                                                                               | Expected Results                                                                       | Actual Results                                                                                | Pass/Fail |
|-----|------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|-----------|
| 1   | testing connection of TCP and UDP to client and server                             | expected both connections will connect to travel without fail                          | both connections have connected to each other perfectly well                                  | pass      |
| 2   | testing the mainmenu within the TCP server for the hangman have sent to the client | client receives the information                                                        | client receives the information                                                               | pass      |
| 3   | testing to see the game running on that the UDP server has been sent to the client | the client has received the hangman game                                               | the client hasn't received the hangman game, but it has received bytes of information instead | fail      |
| 4   | testing if multithreading works while the end-user has won or loss                 | the end user has to wait 10 seconds for multithreading to finish                       | the end-user has waited 10 seconds until multithreading has finished                          | pass      |
| 5   | testing the server and client connection multiple times to see there is no crashes | to see no crashes at all                                                               | there were no crashes at all                                                                  | pass      |
| 6   | keeping a server and client connected for long periods of time                     | seeing if the server and client would crash while staying on for a long period of time | there were no crashes at all                                                                  | pass      |

## References

- [1] S. Varadarajan, "User User Datagram Datagram Protocol (UDP) Protocol (UDP)."
- [2] M. Rouse, "What is UDP (User Datagram Protocol)? - Definition from Whatls.com," *Search networking*, 2018. [Online]. Available: <https://searchnetworking.techtarget.com/definition/UDP-User-Datagram-Protocol>. [Accessed: 05-Apr-2019].
- [3] C. Fu and E. Canada, "TCP/UDP Basics."
- [4] Margaret Rouse, "What is TCP (Transmission Control Protocol)? - Definition from Whatls.com," *SearchNetworking*, 2014. [Online]. Available: <https://searchnetworking.techtarget.com/definition/TCP>. [Accessed: 05-Apr-2019].
- [5] A. Rodriguez, J. Gatrell, J. Karas, and R. Peschke, "TCP/IP Tutorial and Technical Overview Understand networking fundamentals of the TCP/IP protocol suite Contains advanced concepts such as QoS and security Includes the latest TCP/IP protocols."

# Net Game Development by P148551

---

## GRADEMARK REPORT

---

### FINAL GRADE

60/100

### GENERAL COMMENTS

#### Instructor

UDP Server (30) 18

TCP Server (30) 18

Process Sync (20) 11

Work Independently (20) 13

Report provides quite a good discussion of the code you produced.

Code is clean and network code is well structured. However, the use of interaction between client and server is very limited. The hangman game is played entirely within the server and client is used to start the game. There was some evidence of threading.

---

PAGE 1

---

PAGE 2

---

PAGE 3



#### Comment 1

UDP

---

PAGE 4

---

PAGE 5

---

PAGE 6

---

PAGE 7

---

PAGE 8

---

PAGE 9

---

PAGE 10

---

PAGE 11

---

PAGE 12

---

PAGE 13

---

PAGE 14

---

PAGE 15

---

PAGE 16

---

PAGE 17

---

PAGE 18

---

PAGE 19

---

PAGE 20

---

PAGE 21

---

PAGE 22

---