

Abigail Davies - Assignment 2

by Abi Davies

Submission date: 14-May-2025 09:17PM (UTC+0100)

Submission ID: 259279216

File name: 32723_Abi_Davies_Abigail_Davies_-_Assignment_2_664977_471454300.docx (766.68K)

Word count: 933

Character count: 4629

Abigail Davies (2114472)

University of Wales Trinity St. David

Socket Programming and Concurrency Report (ACCA6028)

Assignment 2

Contents

Introduction 3

Prototype Design and Development 3

Logic of Tennis Game Implemented to Client / Server Design..... 8

Conclusion 9

References 10

Introduction

This report details the design and development stages of creating a tennis game that is communicated between a client and server. Initially, a prototype of the tennis game was developed in order to understand the logic required for the match rules. A basic chat application was then developed to ensure there was a connection between a client and a server that was successful. The last step was to combine the two prototypes to generate a networked tennis match game.

Prototype Design and Development

The first stage of designing the prototype of the tennis game was to have a clear understanding of score rules for a tennis game. For example, a player can score 0-4 but when both players are starting at 0 points, the score board should display LOVE [1]. Wilson blogs documented these rules in 2024 [1].

Figure 1 is a diagram created of the data flow of the basic tennis game.

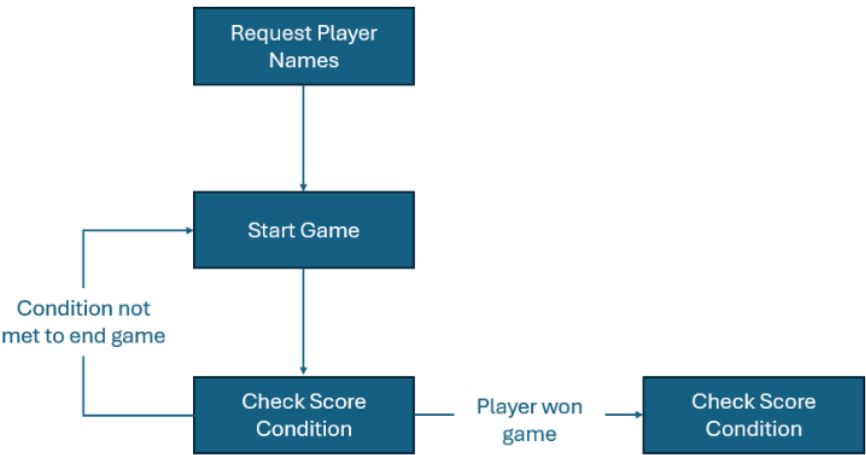


Figure 1

As shown in figure 1, the initial tennis match game requests both players names before starting the game by entering a loop. The game then stays in the loop until a condition is met where one of the players have won the match.

This was implemented and tested against a number of scenarios to ensure that the logic was behaving as expected. A couple of the tests were put in place to confirm that LOVE was returned as the starting score of 0-0, DEUCE was returned when both player 1 and player 2 were drawing with 4 points, ADVANTAGE for a player who had more than 3 points and was the next player to score, and finally, GAME was returned for the winner's score. Other tests such as ensuring that the results displayed to the console, as well as providing information on which player had won the game were also performed. Figure 2 is a screenshot of the logic developed.

```

1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 /*Function that executes a switch statement to return the naming convention of a point.
7 https://www.wischools.com/cpp/cpp\_switch.asp */
8 string getScore(int point) {
9     switch (point) {
10         case 0:
11             return "0";
12         case 1:
13             return "15";
14         case 2:
15             return "30";
16         case 3:
17             return "40";
18         default:
19             return "40";
20     }
21 }
22
23 int main() {
24     // Declaring and initialising variables required for tennis game.
25     string player1;
26     string player2;
27     int p1Points = 0;
28     int p2Points = 0;
29     bool matchComplete = false;
30     string player;
31
32     // Information displayed to console to request each players name.
33     cout << "Please enter player 1's name?" << endl;
34     cin >> player1;
35     cout << "Please enter player 2's name?" << endl;
36     cin >> player2;
37
38     // Print to console to confirm game has started.
39     cout << player1 << " against " << player2 << endl;
40     cout << "Starting score: LOVE" << endl;
41
42     /*While loop used to continue game, e.g. while match complete is not true
43     https://www.geeksforgeeks.org/cpp-while-loop/*/
44     while (!matchComplete) {
45         // User prompted to enter which player scored a point.
46         cout << "Enter 1 if " << player1 << " has scored a point." << endl;
47         cout << "Enter 2 if " << player2 << " has scored a point." << endl;
48         cin >> player;
49
50         //Point system update: if user enters 1 (player1), p1points increase by 1, vice versa.
51         if (player == "1") {
52             p1Points++;
53         }
54         else if (player == "2") {
55             p2Points++;
56         }
57         // validation to prevent user from entering an invalid option.
58         else {
59             cout << "You have entered an invalid option." << endl;
60             cout << "Please try again." << endl;
61             continue;
62         }
63
64         /*Score board logic:
65         if both player points are less than 3, print score. */
66         if (p1Points < 3 && p2Points < 3) {
67             cout << "Score: " << getScore(p1Points) << " - " << getScore(p2Points) << endl;
68         }
69         // if both player points are greater than or equal to 3, and equal to eachother, print DEUCE.
70         else if (p1Points >= 3 && p2Points >= 3 && p1Points == p2Points) {
71             cout << "Score: DEUCE" << endl;
72         }
73         // if bother players points are greater than 3 but 1 player has 1 more point, print advantage to the desired player.
74         else {
75             if (p1Points > 3 && p1Points == p2Points + 1) {
76                 cout << "Score: ADVANTAGE to " << player1 << endl;
77             }
78             else if (p2Points > 3 && p2Points == p1Points + 1) {
79                 cout << "Score: ADVANTAGE to " << player2 << endl;
80             }
81             else {
82                 // checking if either player has to have a minimum of 4 points and 2 points ahead of oponent to win the game.
83                 if (p1Points >= 4 && p1Points >= p2Points + 2) {
84                     cout << "Score: GAME - " << getScore(p2Points) << endl;
85                     cout << player1 << " won!" << endl;
86                     // updated boolean to true to exit while loop once player has won the game.
87                     matchComplete = true;
88                 }
89                 else if (p2Points >= 4 && p2Points >= p1Points + 2) {
90                     cout << "Score: " << getScore(p1Points) << " - GAME" << endl;
91                     cout << player2 << " won!" << endl;
92                     // updated boolean to true to exit while loop once player has won the game.
93                     matchComplete = true;
94                 }
95             }
96         }
97     }
98 }

```

Figure 2

Client / Server Design

Once the game was functioning as expected, the next stage was to develop a client and server that had a successful connection to each other. For this step, a basic chat application was generated to test that simple messages could be sent and received from both the client and server.

The design of this step is shown in figure 3.

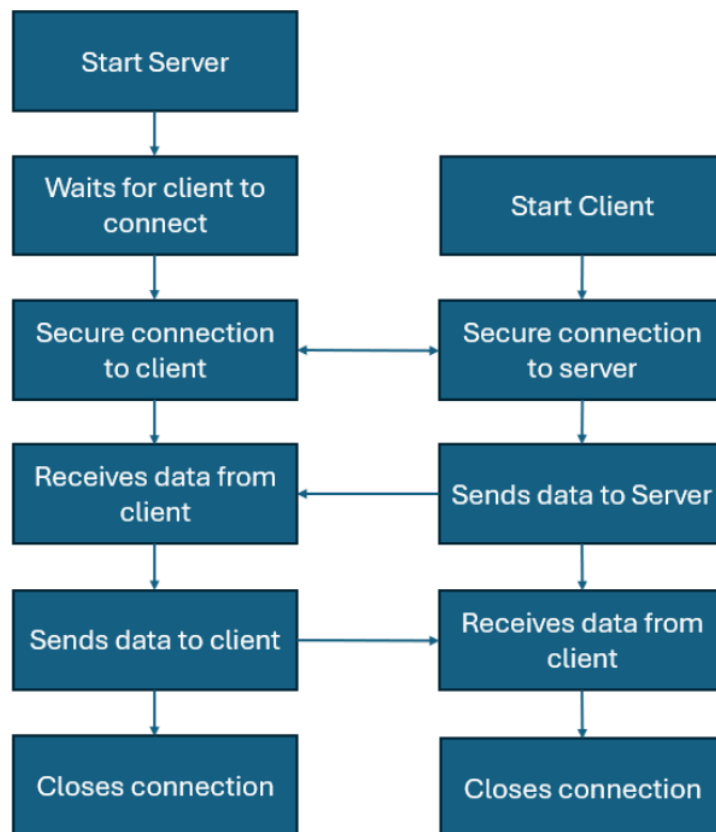
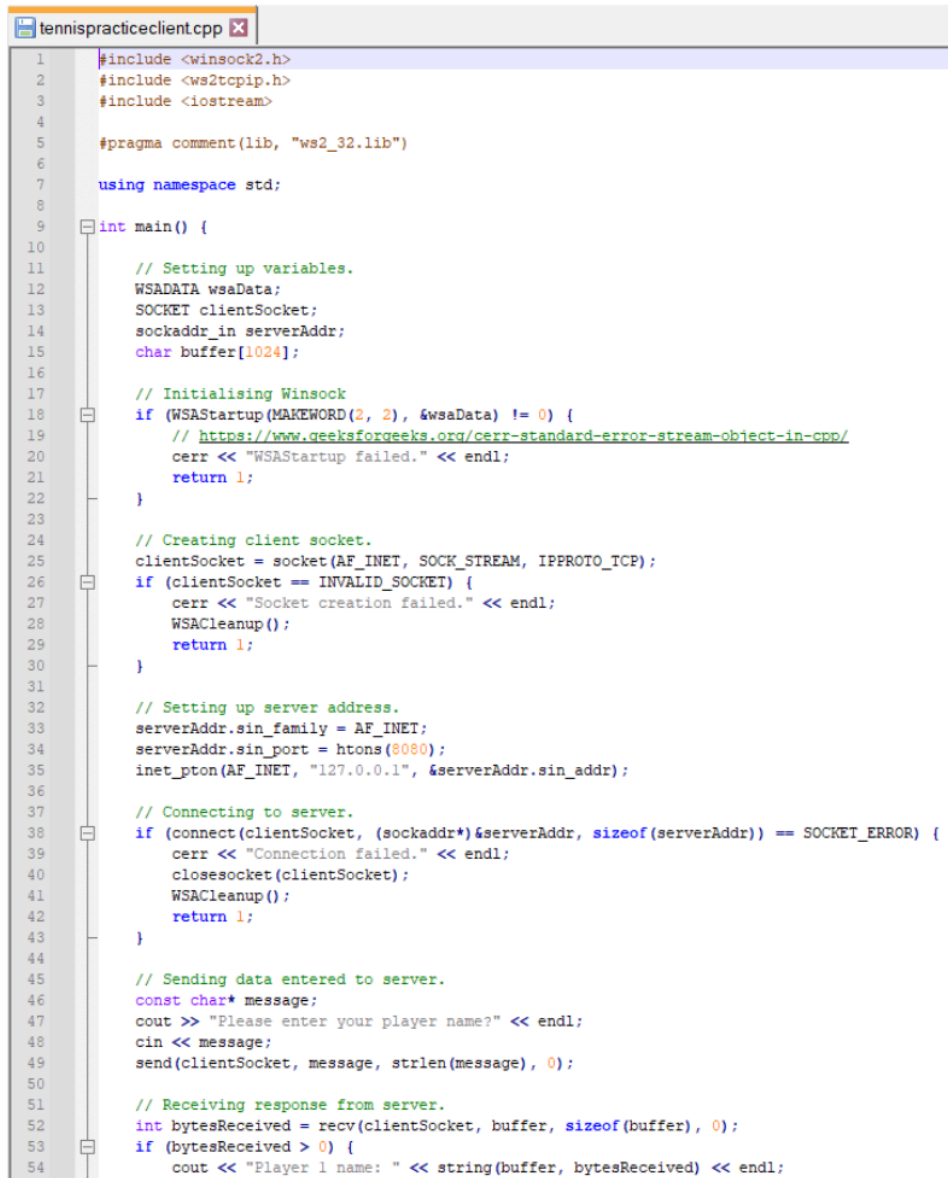


Figure 3

Figure 3 shows the simplified data flow of the chat server generated. The server is started first and waits for a client to be available to connect. The client server is then started and the server should be listening to allow the connection to be successful. The Client is prompted to enter their name and sent to the server, followed by the server waiting to receive the message and respond with their data. Once the above has been received successfully from the client, the connections are closed on both systems.

Figure 4-7 is a screenshot of the code being executed. This was also part of testing the application by ensuring that the error handling messages were not displayed. Although, the design process was initially to ensure a secure connection was successful, the testing highlighted that a feature needed to be implemented to be able to exit from the application as it was stuck in a continuous loop.



```
1 #include <winsock2.h>
2 #include <ws2tcpip.h>
3 #include <iostream>
4
5 #pragma comment(lib, "ws2_32.lib")
6
7 using namespace std;
8
9 int main() {
10
11     // Setting up variables.
12     WSADATA wsaData;
13     SOCKET clientSocket;
14     sockaddr_in serverAddr;
15     char buffer[1024];
16
17     // Initialising Winsock
18     if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
19         // https://www.geeksforgeeks.org/cerr-standard-error-stream-object-in-cpp/
20         cerr << "WSAStartup failed." << endl;
21         return 1;
22     }
23
24     // Creating client socket.
25     clientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
26     if (clientSocket == INVALID_SOCKET) {
27         cerr << "Socket creation failed." << endl;
28         WSACleanup();
29         return 1;
30     }
31
32     // Setting up server address.
33     serverAddr.sin_family = AF_INET;
34     serverAddr.sin_port = htons(8080);
35     inet_pton(AF_INET, "127.0.0.1", &serverAddr.sin_addr);
36
37     // Connecting to server.
38     if (connect(clientSocket, (sockaddr*)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR) {
39         cerr << "Connection failed." << endl;
40         closesocket(clientSocket);
41         WSACleanup();
42         return 1;
43     }
44
45     // Sending data entered to server.
46     const char* message;
47     cout >> "Please enter your player name?" << endl;
48     cin << message;
49     send(clientSocket, message, strlen(message), 0);
50
51     // Receiving response from server.
52     int bytesReceived = recv(clientSocket, buffer, sizeof(buffer), 0);
53     if (bytesReceived > 0) {
54         cout << "Player 1 name: " << string(buffer, bytesReceived) << endl;
```

Figure 4

```

tennispracticeclient.cpp
52     int bytesReceived = recv(clientSocket, buffer, sizeof(buffer), 0);
53     if (bytesReceived > 0) {
54         cout << "Player 1 name: " << string(buffer, bytesReceived) << endl;
55     }
56
57     // Closing socket for client, following clean up as winsock function is complete.
58     closesocket(clientSocket);
59     WSACleanup();
60
61     return 0;
62 }
63

```

Figure 5

```

tennispracticeserver.cpp
1     #include <winsock2.h>
2     #include <ws2tcpip.h>
3     #include <iostream>
4
5     #pragma comment(lib, "ws2_32.lib")
6
7     using namespace std;
8
9     int main() {
10
11         // Setting up variables.
12         WSADATA wsaData;
13         SOCKET serverSocket, clientSocket;
14         sockaddr_in serverAddr, clientAddr;
15         int clientAddrSize = sizeof(clientAddr);
16         char buffer[50];
17
18         // Initialising Winsock
19         if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
20             // https://www.geeksforgeeks.org/cerr-standard-error-stream-object-in-cpp/
21             cerr << "WSAStartup failed." << endl;
22             return 1;
23         }
24
25         // Creating server socket.
26         serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
27         if (serverSocket == INVALID_SOCKET) {
28             cerr << "Socket creation failed." << endl;
29             WSACleanup();
30             return 1;
31         }
32
33         // Setting up server address.
34         serverAddr.sin_family = AF_INET;
35         serverAddr.sin_addr.s_addr = INADDR_ANY;
36         serverAddr.sin_port = htons(8080);
37
38         if (bind(serverSocket, (sockaddr*)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR) {
39             cerr << "Bind failed." << endl;
40             closesocket(serverSocket);
41             WSACleanup();
42             return 1;
43         }
44
45         // Listening for any incoming connections.
46         cout << "Waiting for second player to connect..." << endl;
47         if (listen(serverSocket, 1) == SOCKET_ERROR) {
48             cerr << "Listen failed." << endl;
49             closesocket(serverSocket);
50             WSACleanup();
51             return 1;
52         }
53

```

Figure 6

```
tennispracticeserver.cpp
49     closesocket(serverSocket);
50     WSACleanup();
51     return 1;
52 }
53
54 // Found client connection and ensuring it can be accepted.
55 clientSocket = accept(serverSocket, (sockaddr*)&clientAddr, &clientAddrSize);
56 if (clientSocket == INVALID_SOCKET) {
57     cerr << "Accept failed." << endl;
58     closesocket(serverSocket);
59     WSACleanup();
60     return 1;
61 }
62
63 // Informational message to console to show client successfully connected.
64 cout << "Client connected" << endl;
65
66 // Receiving data from client.
67 int bytesReceived = recv(clientSocket, buffer, sizeof(buffer), 0);
68 if (bytesReceived > 0) {
69     cout << "Player 2 name: " << string(buffer, bytesReceived) << endl;
70 }
71
72 // Sending a reply to the client.
73 const char* response;
74 cout >> "Please enter your player name?" << endl;
75 cin << response;
76 send(clientSocket, response, strlen(response), 0);
77
78 // Closing socket for client and server, following clean up as winsock function complete.
79 closesocket(clientSocket);
80 closesocket(serverSocket);
81 WSACleanup();
82
83 return 0;
84 }
85
```

Figure 7

Logic of Tennis Game Implemented to Client / Server Design

The final stage was to combine both prototypes and develop a networked tennis game where the server controlled the score board and returned it the console for both the client and server to view. The client was one player, and the server was another player.

The .cpp files are commented throughout so that the reader can have a clear understanding of what each section of code aims to do. It proved difficult in some areas to combine both prototypes and therefore, chat gpt was utilised to analyse the code. In addition, some testing resulted in memory warnings that would prevent the match from completing in full. Chat gpt provided an understanding whether the code would work successfully if more memory was available.

The match initially prompts both players to send their player names to their opponent. This feature was added for a personal touch. It also prevents confusion between players that view the score board. For example, the informational messages will state the players name rather than player 1 / player 2, etc.

For further development, this project could be improved by introducing an option menu to the players. This will allow the players to quit mid match if required without throwing errors to their opponent.

Conclusion

To conclude, the program performs as anticipated and is commented throughout for readability. The testing actioned was successful and created an understanding that the programs on the client and server were executing as intended. Creating a prototype of the non-networked tennis match, followed by creating a simple chat server, assisted in the creation of the final product. Reducing the code into smaller tasks allowed me to recognise the requirements of each area before merging them together. However, it could be improved by implementing more classes. Introducing more multiple classes to store the existing variables, logic and connectivity code will allow it to be easier to classify the sections of code.

References

- [1] Wilson, "How to Keep Score in Tennis", *Wilson*. Jan. 18, 2024. [Online]. Available: <https://www.wilson.com/en-us/blog/tennis/how-tos/how-keep-score-tennis> [Accessed on May. 10, 2025].

Abigail Davies - Assignment 2

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

A reasonable, if limited effort made - not quite wholly realised, but nonetheless a good application of the principles.

52/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10
