Connect Four

Overview

The topic of this project is the client-server implementation of the board game called Connect Four.

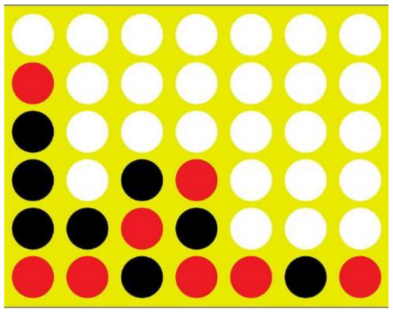

Figure 1: Example connect four board (Yellow circle - unlimited download. Cleanpng.com.)

Connect Four is a two-player game in which players take turns by entering a token of their chosen color into a board consisting of holes arranged in a 6x7 or 7x7 grid. Each player aims to connect four of their colored tokens on the board in a horizontal, vertical, or diagonal fashion, with the game ending when either of the players wins, or when there are no more empty holes left on the board.

In this project, we have created a Connect Four network-application that allows users to play Connect Four from the terminal of their respective computers. The two broad concepts of computer networks applied in our application are the Client-Server Paradigm and the Transmission Control Protocol (TCP) communications standard. Our application is based on a client-server model where the server remains on for the duration of the games and the clients communicate with each other indirectly through the server. Moreover, the application makes use of the TCP protocol in the transport layer, as the clients connect to a running server by creating their specific TCP sockets – specifying the IP address and port number of the server process – thereby getting bound to the server. The usage of TCP allows the application to prioritize reliability of information transfer between a client and server – which is necessary for the game to be evaluated correctly to determine the appropriate outcome.

Our application is defined across three classes. Each user (client) is defined in a client class that allows the player to make moves in the terminal of their computer. These clients connect to the server that is defined in the server class – which is tasked with pairing players for matches, validating the successful connection of the players, launching each game in its own thread, and keeping track of games from their start to their end. The functionality of a game – involving displaying the board to the players, asking them to make their moves and ascertaining the outcome of a game – are performed in the game class. The server can verify active players and games and handle multiple connections and games simultaneously.

An important characteristic of our implementation of Connect Four is the provision for a user to use a maximum of one hint throughout the game to determine the optimal move to play at a given stage. This feature is added because Connect Four is a solved game – meaning that at any given time, there exists an optimal sequence of moves that can be made to maximize chances of victory (Cahn, 2024). The provision of this hint is made possible using an API that communicates the state of a game to a minimax algorithm with alpha-beta pruning that processes the game, and returns moves with their corresponding minimax scores. The move with the highest minimax score is the optimal move at the stage – which is made on the user's behalf.

System Architecture

The Connect Four application has 2 main components, the server and clients. These are implemented in the C4Server.java and C4Client.java files respectively. The server's main function is to run the game logic and initiate communication between 2 clients. The client's main role is informing the server of what move they want to make during their turn.

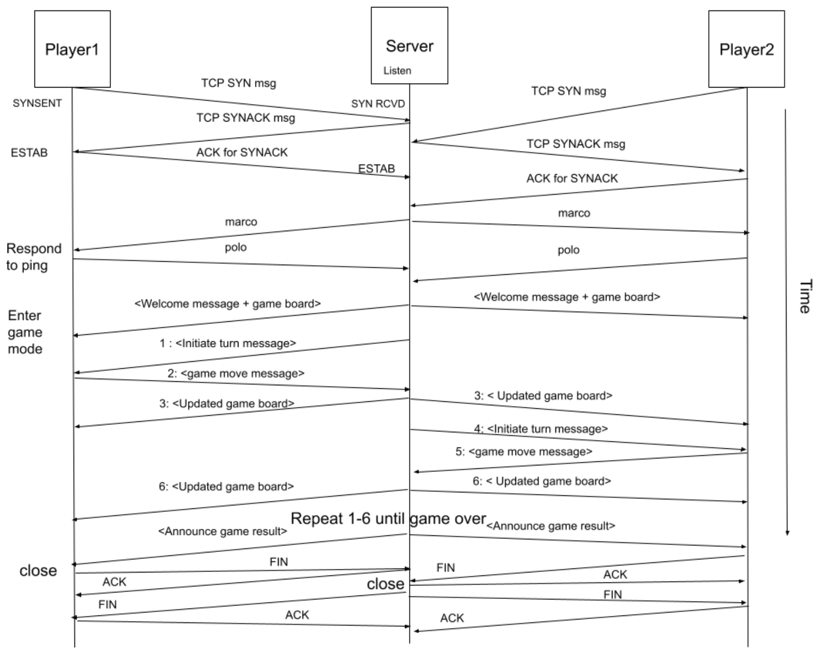

Figure 1: Timing diagram

The process that starts the exchange of messages between the clients and the server is the establishment of a TCP connection. The server creates a server class object with a server socket field at the localhost IP address and port number 12345 – and accepts connection requests made by the two clients from their respective client-side sockets.

Between the completion of the TCP handshake and the start of the game in its own thread, a pinging mechanism is used to ensure that a player is still active – as only then is a thread created for a game between two active clients (players). To do so, the server sends a message with the word "marco" to both clients. The clients receive the message and send the server a message with the word "polo" - indicating to the server that the player is active which means that the client is eligible to be matched with another active client for a game. In a first in first out order – two active clients are added to a C4Game object passed as a field to a thread object that runs the game.

During the game, the board is sent to both players and each player is asked to routinely make a move. The board is updated at the end of each move until an outcome is reached – with either player having won the game or there being a tie (with no empty spots left on the board). At this stage, a message indicating the outcome of the game is sent to both clients and the connection is ended. The entire process of communication between the clients and the server is illustrated in figure 2.

Our application runs in the Application Layer of the TCP/IP protocol stack diagram. The timing diagram focuses on application-to-application (message-to-message) communication without the details of the other network layers. In reality, every message on the timing diagram has to traverse all layers and each layer serves an important role in allowing communication. We can use the TCP/IP protocol stack diagram in figure 3 to illustrate how the network layers handle the ping mechanism.

We start at the application layer where the C4Server.java file is running. To conduct the ping mechanism, the server must send the text “marco” to the client and receive “polo”. As such, the application layer message “marco” is sent to the transport layer.

The transport layer is a part of the operating system of the machine and is running TCP. This layer adds a TCP header to the message including the source port (server port number) and destination port number (client port number). The data is now referred to as a TCP segment. The transport layer then sends this TCP segment to the network layer. Later, when the transport layer receives a TCP segment from the network layer, it will go through the process of demultiplexing in which it uses source IP address, source port number, destination IP address, and destination port number do determine to which socket to send the encapsulated message.

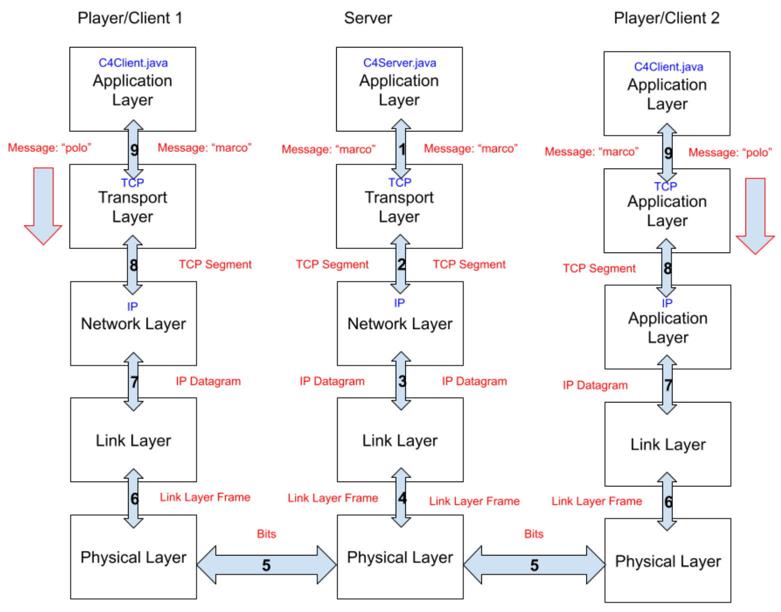

Figure 2: Stack diagram.

The network layer includes the machine on which the program is running as well as all routers and switches in the larger internet. The main protocol at the network layer is IP. Like TCP, IP also has a header which it uses to encapsulate the TCP segment. The TCP header includes, most importantly, the source (server) and destination (client) IP addresses. The addition of the IP header creates an IP datagram. The two main functions of the network layer are forwarding and routing. Forwarding is moving packets from a router’s input link to appropriate router output link while routing is determining the route to take from source to destination. The combination of the two along with the information in the IP header is essential to move the packet from source to destination.

The link layer resides between routers to routers and routers to hosts. The link layer facilitates the exchange of a link layer frame between nodes in the network. A frame is the term for an IP datagram encapsulated within a link layer header and trailer which includes information such as the source and destination MAC addresses. “Source” and “destination” in this case being the nodes involved in the data transfer.

The physical layer consists of the physical transportation of bits as voltages or wave frequencies for example. The bits are transported via some medium such as an ethernet cable or air for example.

The message sent by the server then travels the stack in reverse with each layer removing its header and ultimately the transport layer delivers the message to the application layer. Then the player responds with “polo” and thus travels through the stack to the server.

System Implementation

The programming language we used for developing this application was Java. During the development stage, the application was routinely tested by compiling and running on machines running different operating systems such as MacOS and Windows. Therefore, our application can run on all platforms that support a java compiler or interpreter such as a Java Virtual Machine (JVM).

The only API used in our application was for the single hint feature per client in each game. This API – called Connect Four AI -is a free and open-source API developed by Kevin Albertson (kevinAlbs, Connect4/back-end/info.html at master · kevinalbs/connect4). The API endpoint that is used for making a hint is http://kevinalbs.com/connect4/back-end/index.php/getMoves, that takes two parameters: the state of the board as a string player making the move - either 1 or 2. The returned JSON object consists of the possible moves and their minimax scores parsed as a string and the move with the highest minimax score is made on the players' behalf.

To execute this application, it is necessary for the device to have a functioning Java Virtual Machine (JVM), as it makes use of the javac command which compiles the files to run on the JVM.

To execute the application, navigate into the directory containing the downloaded files via a command line terminal, and compile both the server and the client files via their respective commands: javac C4Server.java, javac C4Client.java. Then, execute the server file using the command: java C4Server. This command allows the server to start running on the localhost machine at port number 12345. Lastly, execute the client file in two different terminals using the command: java C4Client localhost 12345. Note that additional users can join since the server can handle concurrent games.

We are currently assuming that the server and all client are running on the same machine. There have been varying levels of success running the server and clients on a separate machine. For the best results, we suggest running both the server and clients on the same machine.

Conclusion

Our Connect Four network client-server-based application allows multiple pairs of users to play Connect Four. Each game takes place in its own thread, and another thread is used by the server to keep track of games that are occurring in real time and those that have ended. The server can check for active players and match them for games. A TCP/IP communications standard ensures reliability of information transfer, leading to accurate game evaluation. Currently, the standout feature in our application is the usage of a Connect Four AI API that can be used to provide a hint per player in each game if the player does not know what the optimal move to play at a given stage is.

Moving forward, there are a variety of extra features that can be added to enhance functionality. The most important feature being authentication – allowing users to create accounts for the game with distinct usernames and passwords. Additionally, features that could give our application a more modern look involve providing a timer for players to make moves, as well as allowing the players to communicate with each other in real time using emoticons – like the “emote” feature in popular games such as Clash Royale.

References

1. Cahn, L. (2024, April 13). *How to win Connect 4 every time, according to the computer scientist who solved it*. Reader's Digest. https://www.rd.com/article/how-to-win-connect-4/
2. kevinAlbs. (n.d.). *Connect4/back-end/info.html at master · kevinalbs/connect4*. GitHub. https://github.com/kevinAlbs/Connect4/blob/master/back-end/info.html
3. *Yellow circle* - *unlimited download. Cleanpng.com.* cleanpng.com. (n.d.). https://www.cleanpng.com/png-connect-four-board-game-clip-art-connect-four-clip-604155/