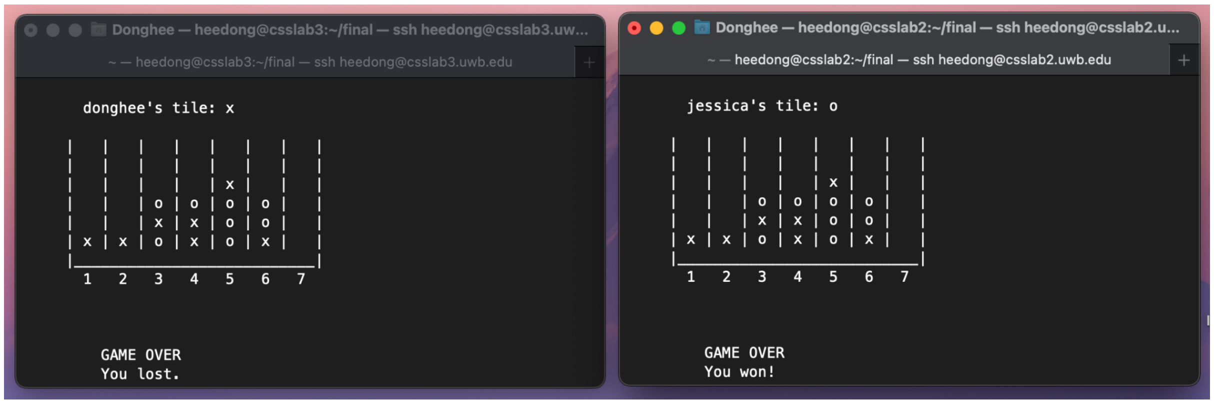
Online Connect 4

Jessica Nguyen, Donghee Lee
CSS 432 Final Project



Donghee — heedong@csslab3:~/final — ssh heedong@csslab3.uw...

~ — heedong@csslab3:~/final — ssh heedong@csslab3.uwb.edu

```
donghee's tile: x

|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   | x |   |   |
|   |   | o | o | o | o |   |
|   |   | x | x | o | o |   |
| x | x | o | x | o | x |   |
|___________________________|
  1   2   3   4   5   6   7


   GAME OVER
   You lost.
```

Donghee — heedong@csslab2:~/final — ssh heedong@csslab2.u...

~ — heedong@csslab2:~/final — ssh heedong@csslab2.uwb.edu

```
jessica's tile: o

|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   | x |   |   |
|   |   | o | o | o | o |   |
|   |   | x | x | o | o |   |
| x | x | o | x | o | x |   |
|___________________________|
  1   2   3   4   5   6   7


   GAME OVER
   You won!
```

Online Connect 4: Client-Server Turn-based Terminal game

Online Connect 4 has a client-server architecture where clients communicate with each other through a server. This terminal game includes 2 players where each player takes turns until the game is over.
The project is available on GitHub at: https://github.com/heedong0612/OnlineGame-Connect4

Run Instructions

Online Connect-4 needs one server and two clients to start the game. Run the following commands on the terminal to host the server and join as clients from two different computers.

To compile and host server:

```
g++ server.cpp NetworkAPI.cpp -o server
./server
```

To compile and connect to the server:

```
g++ client.cpp NetworkAPI.cpp -o client
./client {hostname}
```

How to play

Server

1. The server must be set up and running before running clients join
2. Host the server by running the ./server executable
3. The messages between the clients will be displayed as the standard output

Client

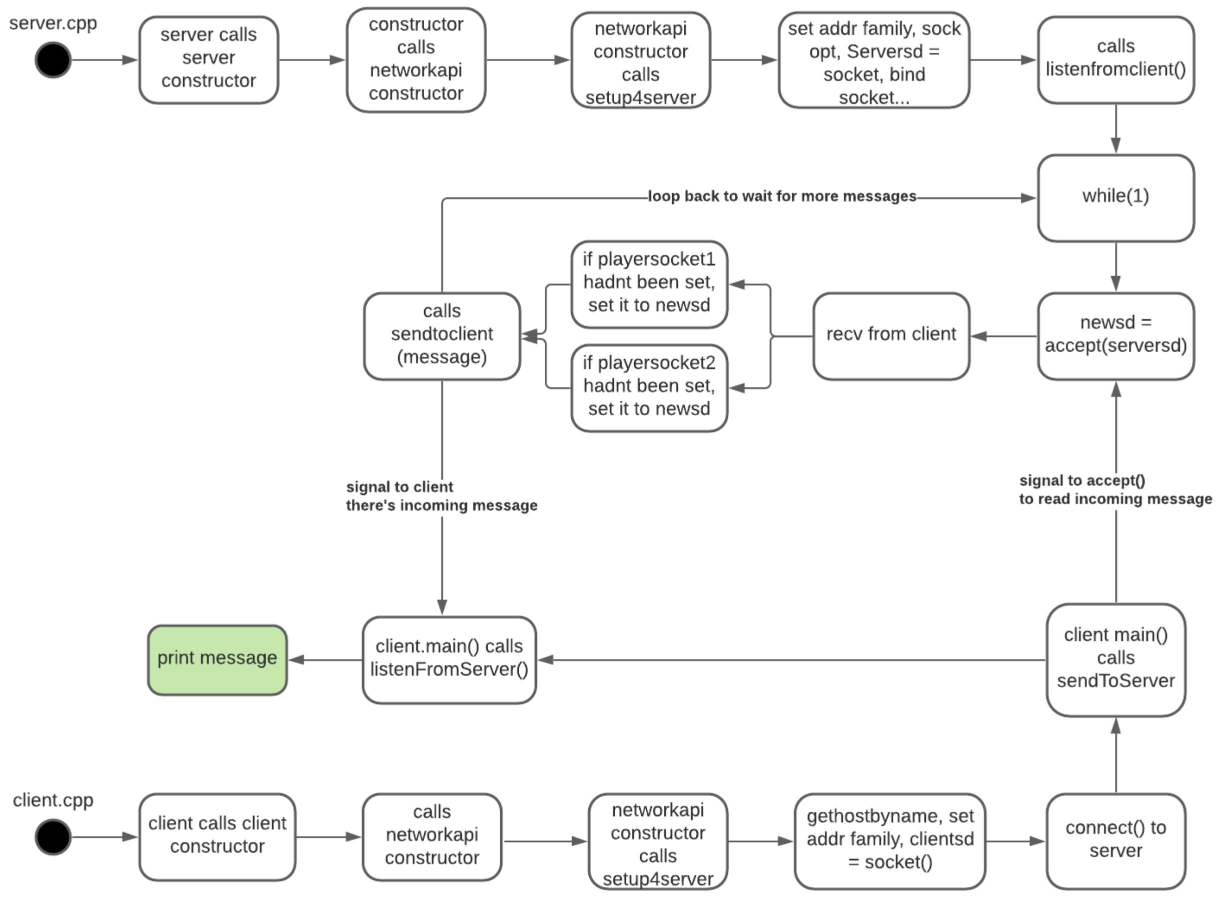
1. Join a game as a client by running the ./client executable with the server name as a command line argument (ex: “csslab1.uwb.edu”)
2. The program prompts the client for a username
3. If you are the first player to join, a waiting message will be displayed and your program will stay idle until the second player joins
4. When there are two players connected, the game will start
5. The clients can input their moves taking turns, and quit any time during the game by typing in ‘q’
6. If there is a winner or a quitter, the program will announce the winner and end game

Gameplay

The game follows the rules of Connect 4 (a.k.a Drop the tile).

1. Players are given a shared board of 6x7 grid
2. Players are assigned either an **X** or an **O** tile to play with
3. On their turns, players are prompted to type in the column number to drop the tile on
4. Players take turn placing one tile at a time
5. A player who places a tile that makes 4 consecutive tiles of the same type in a vertical, horizontal, or diagonal direction wins
6. The game is over when there is one winner or one of the players quits the game

Game Flowchart


server.cpp
server calls server constructor
constructor calls networkapi constructor
networkapi constructor calls setup4server
set addr family, sock opt, Serversd = socket, bind socket...
calls listenfromclient()
while(1)
loop back to wait for more messages
newsd = accept(serversd)
recv from client
if playersocket1 hadnt been set, set it to newsd
if playersocket2 hadnt been set, set it to newsd
calls sendtoclient (message)
signal to client there's incoming message
signal to accept() to read incoming message
print message
client.main() calls listenFromServer()
client main() calls sendToServer
client.cpp
client calls client constructor
calls networkapi constructor
networkapi constructor calls setup4server
gethostbyname, set addr family, clientsd = socket()
connect() to server

Network documentation

Interface NetworkAPI

The Network API consists of a set of tools that the server and client use to communicate with each other. Network API allows the clients to connect to the server and the server to accept their connections. Network API also allows the clients to send and receive messages from the server and the server to send and receive messages from the client. The Network API provides a communication bridge that allows the server to decide to accept a player, start the game, manage the game flow, and disconnect the clients. It also allows the client program to alert the server about the decisions that the players make during the game such as setting up a username, making a move, or quitting the game. The Network API plays a critical role in making the game online and interactive.

The following part notes the functionalities that the Network API provides to the server and clients.

Function	**Description**
NetworkAPI()	Sets the port number(12345) and all sockets to null
bool setup4Client (char hostname[])	Connects the client to the server indicated by the hostname using TCP protocol and the preset port number
bool setup4Server()	Accepts the client connection through the preset port number and remembers each client socket
bool sendToServer (const char message[])	Allows the client to send a message to the server using send()
bool sendToClient (const char message[])	Allows the server to send a message to the client using send()
string listenFromServer()	Returns the message from the server to a client
string listenFromClient()	Returns the message from a client to the server

Client-Server documentation

The client and server program consist of a set of functions that determines and executes actions in-game according to the message passed by the Network API. The following notes the implementation of the client and server.

Client

The client manages the delivery of the user input to the server and back. It communicates with the server and shares the current game status with the user via terminal standard outputs.

Function	**Description**
Client()	Connects to the server and Initializes the game
void registerUser (string username)	Prompts the user for a username and sends to the server
bool chooseMove (char c)	Prompts the user for the next move and check for its validity. The valid move can be either digits between 1 and 7, or a character ‘q’ for quit.
bool isGameOver()	Returns whether the game is over
bool isMyTurn()	Returns whether it is the user’s turn
bool listenForServer()	Reads a message from the server and make proper decisions
void drawBoard()	Draws a 6x7 grid on the terminal displaying the tiles from both players
void endGame()	Terminates the program on the client side

Server

The server accepts messages from both clients and makes decisions if the messages are valid and allowed. The server initiates the game, accepts valid moves, broadcasts decisions, records the board, determines a winner, and terminates the connections.

Function	**Description**
Server()	Initializes the game
void acceptUser()	Accepts two clients
void startGame()	Broadcasts the start of the game to all clients with a message indicating the first player to move
void acceptMove()	Accepts a decision from the player and broadcasts to all clients
bool isGameOver()	Checks and returns whether the game is over
void endGame()	Broadcasts the end of the game and the winner to all clients
string player_names[2]	Returns the user names of the players

Work Distribution

Task	**Assigned To**
How to play presentation	Both
Network API Implementation	Jessica
Client Implementation	Donghee
Server Implementation	Donghee
Merge Implementation	Both
Network protocol document	Jessica

How we coordinated the development of the project

- Discussed and brainstorm for project ideas in week 7
- Discussed with the professor about server architecture and network API in week 8
- Decided on the function headers on the server, client, and network API codes in week 9
- Distributed work and implemented the functions in week 9 and 10
- Discussed, debugged, and merged our codes in week 10
- Demoed the program to the class in week 10
- Made further changes to improve the program in week 10

Tools that we used to develop the project

- We used *Github* for version control and used c++ as the programming language
- We worked on three different computers in *UWB Linux machines*
- We collaborated using *zoom, discord, and visual studio code*