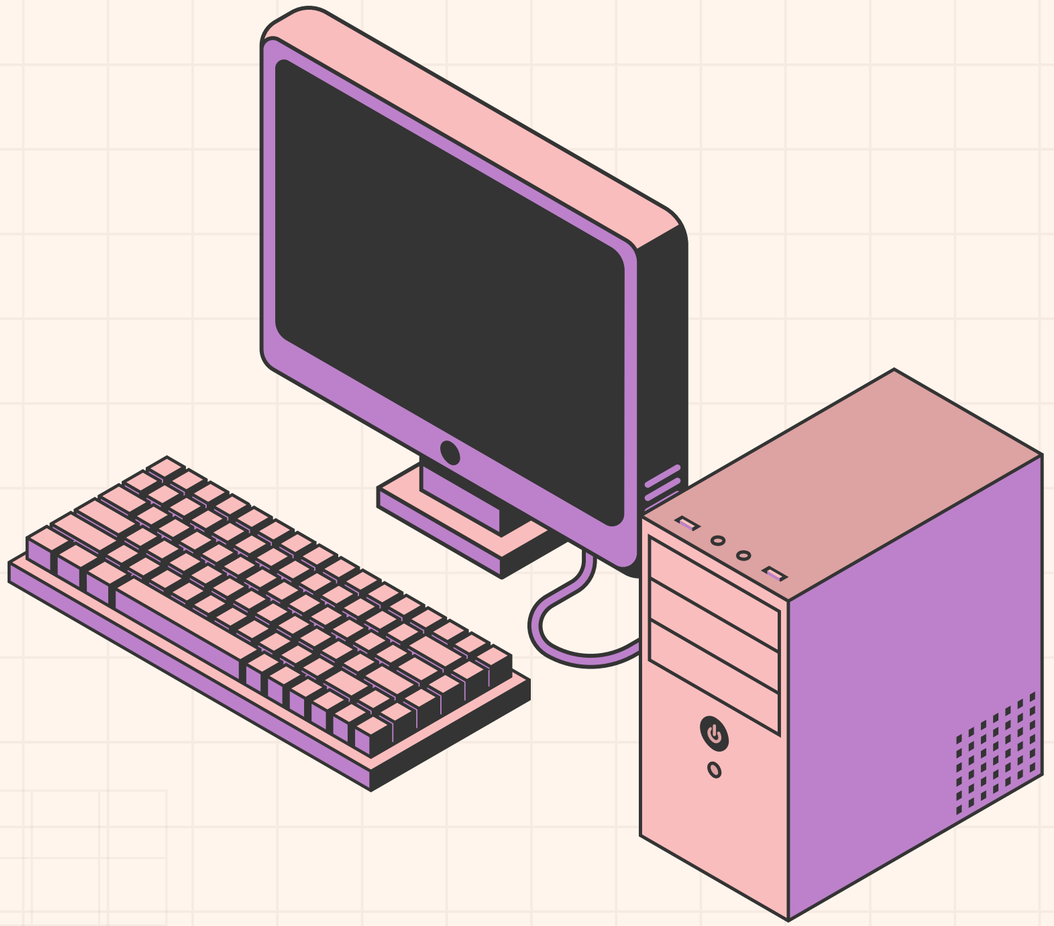
GRINNELL COLLEGE

MULTI-THREADED TIC-TAC-TOE

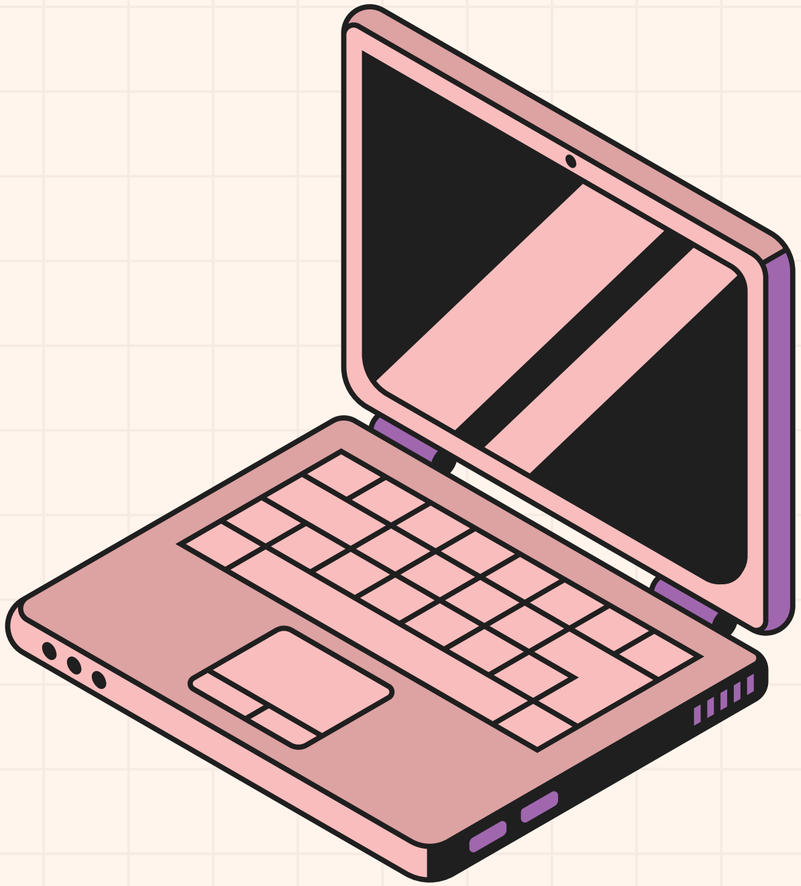
Jonathan Wang & Zakariye Abdilahi



GRINNELL COLLEGE

WHAT WE BUILT

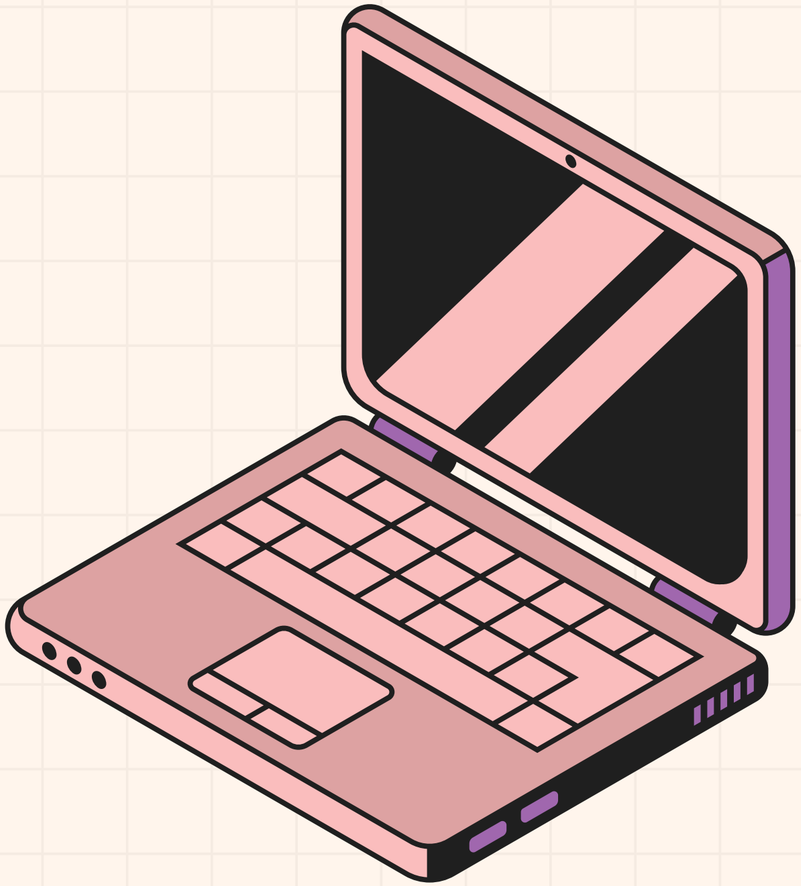
- A fully networked Tic-Tac-Toe game enabling real-time multiplayer interaction
- A setup that allows players to join remotely, enter their names, and play
- Concurrency support so multiple Tic-Tac-Toe matches can run at the same time
- Clear rules enforcement, move validation, and outcome detection for fair play
- A strong foundation that can be extended or adapted for future enhancements



GRINNELL COLLEGE

IMPLEMENTATION DETAILS

- The server uses a network "door" (a TCP/IP socket) to listen for and accept connections from player clients.
- Players provide their names and get paired by the server into separate game sessions, each managed by its own thread.
- The server enforces the rules of Tic-Tac-Toe: it validates moves, updates the board, and detects wins, draws, or quits.
- Each client runs an additional receiving thread to get real-time updates from the server while they enter their moves.
- The overall system allows multiple games to run in parallel, providing a smooth and interactive multiplayer experience.



GRINNELL COLLEGE

COURSE CONCEPTS

NETWORKING & DISTRIBUTED SYSTEMS

- Enable the server to handle multiple games simultaneously
- Use of sockets to establish a client-server model
- Each player inputs their move until the game ends

PARALLELISM WITH THREADS

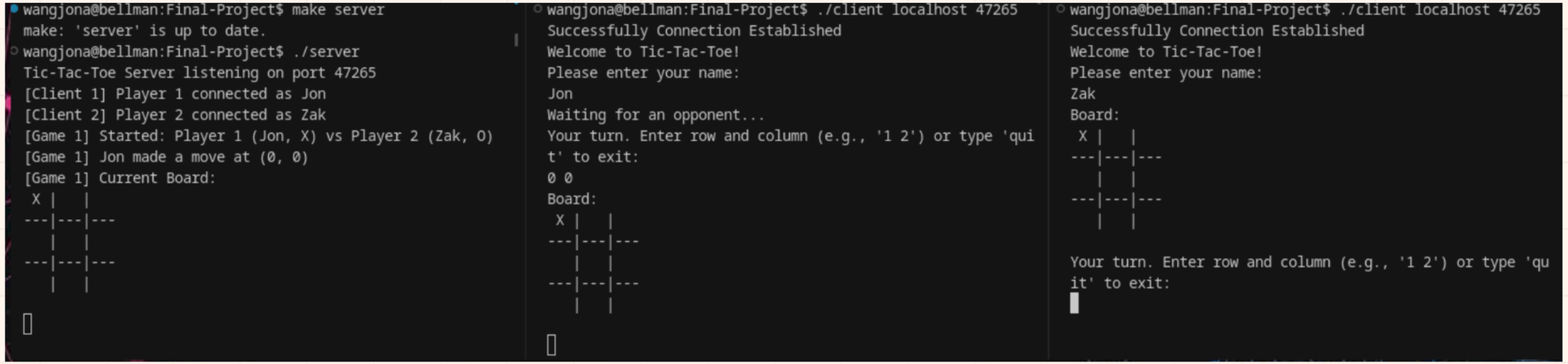
- Each game runs on their own dedicated thread
- Server generates a thread for each game, allowing independent plays in individualized sessions

FILES & FILE SYSTEMS

- Provide storage for game data, enabling save and loading game features using file streams & handling
- Store a game's current position title with each player's name
- Maintain player W/L stats

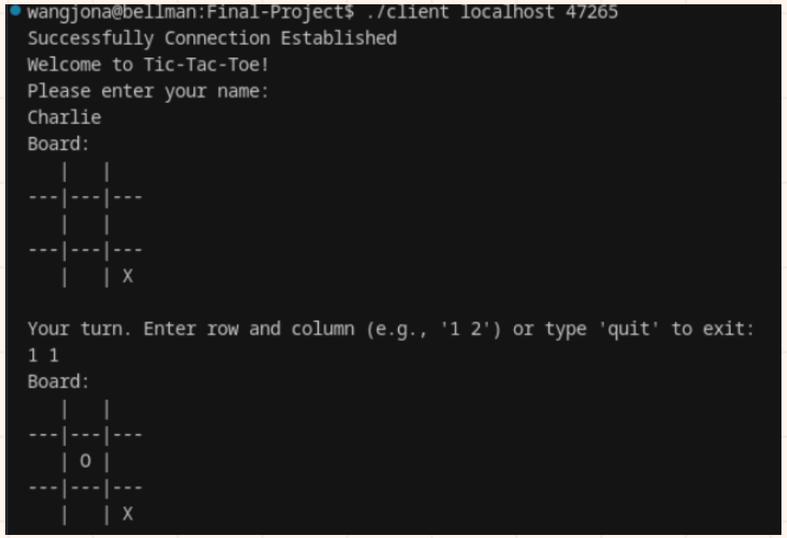
GRINNELL COLLEGE

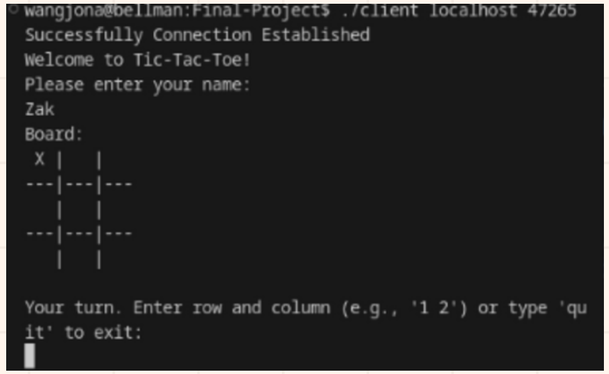
NETWORKING & DISTRIBUTED SYSTEMS

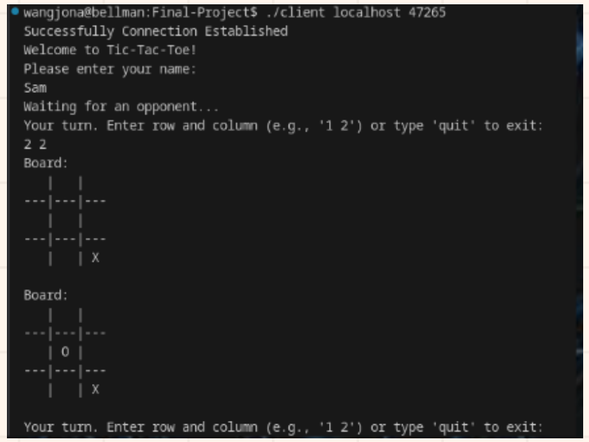


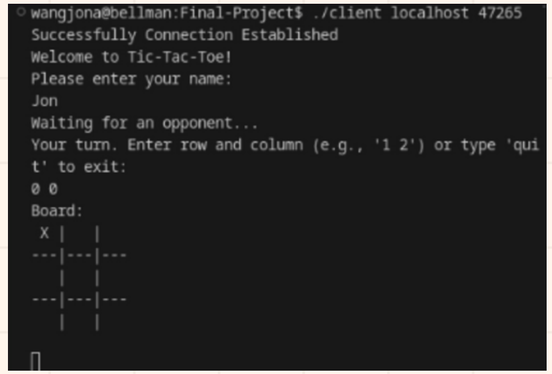
GRINNELL COLLEGE

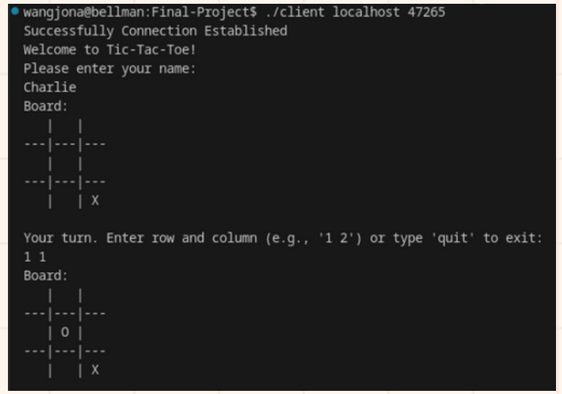
PARALLELISM WITH THREADS











GRINNELL COLLEGE

FILES & FILE SYSTEMS

```c
// Function to record the result in a file within the GameRecords directory
void convertToTxt(char winner, int gameNumber) {
    // Create the GameRecords directory if it doesn't exist
    if (mkdir("GameRecords", 0755) != 0 && errno != EEXIST) {
        perror("Couldn't find GameRecords directory");
        return;
    }

    // Construct the filename
    char filename[100];
    snprintf(filename, sizeof(filename), "GameRecords/Game%d.txt", gameNumber);

    FILE *fp = fopen(filename, "w");
    if (fp == NULL) {
        perror("Error opening file");
        return;
    }

    fprintf(fp, "Game %d between %s and %s\n", gameNumber, player1Name, player2Name);

    if (winner == PLAYER1 || winner == PLAYER2) {
        fprintf(fp, "Winner: %c (%s)\n", winner, winner == PLAYER1 ? player1Name : player2Name);
    } else {
        fprintf(fp, "It's a tie!\n");
    }

    fprintf(fp, "Final Board State:\n");
    for (int i = 0; i < 3; i++) {
        fprintf(fp, " %c | %c | %c \n", board[i][0], board[i][1], board[i][2]);
        if (i < 2) fprintf(fp, "---|---|---\n");
    }

    fclose(fp);
}
```

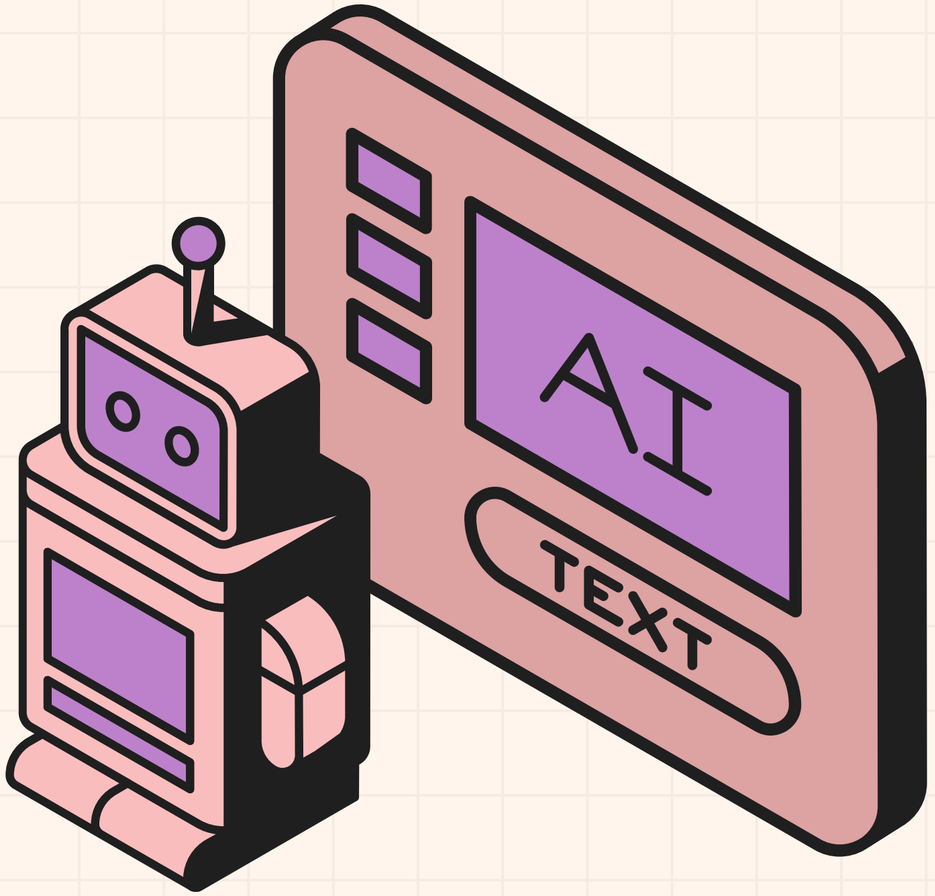
Game-Records.txt

```
Game between Jon and Zak
Winner: T
Board State:
 X | O | X
---|---|---
 X | O | X
---|---|---
 O | X | O

Game between Jon and Zak
Winner: Jon
Board State:
 X | O | O
---|---|---
 X |   |
---|---|---
 X |   |

Game between Sam and Charlie
Winner: Sam
Board State:
 X | X | X
---|---|---
 O | O |
---|---|---
   |   |
```

GRINNELL COLLEGE

INITIAL/CURRENT CHALLENGES

- Implementing Game Logic Inside the Server vs. a Separate File:
  - Keeping all game logic in one place is easier but makes the server code more crowded.
  - Splitting the logic into separate files would be cleaner and easier to maintain, but takes more work upfront.
- Change of plan for files & file systems (Initially tried to include more features)
  - Ongoing attempts at implementing the save and load features
- Server getting messy

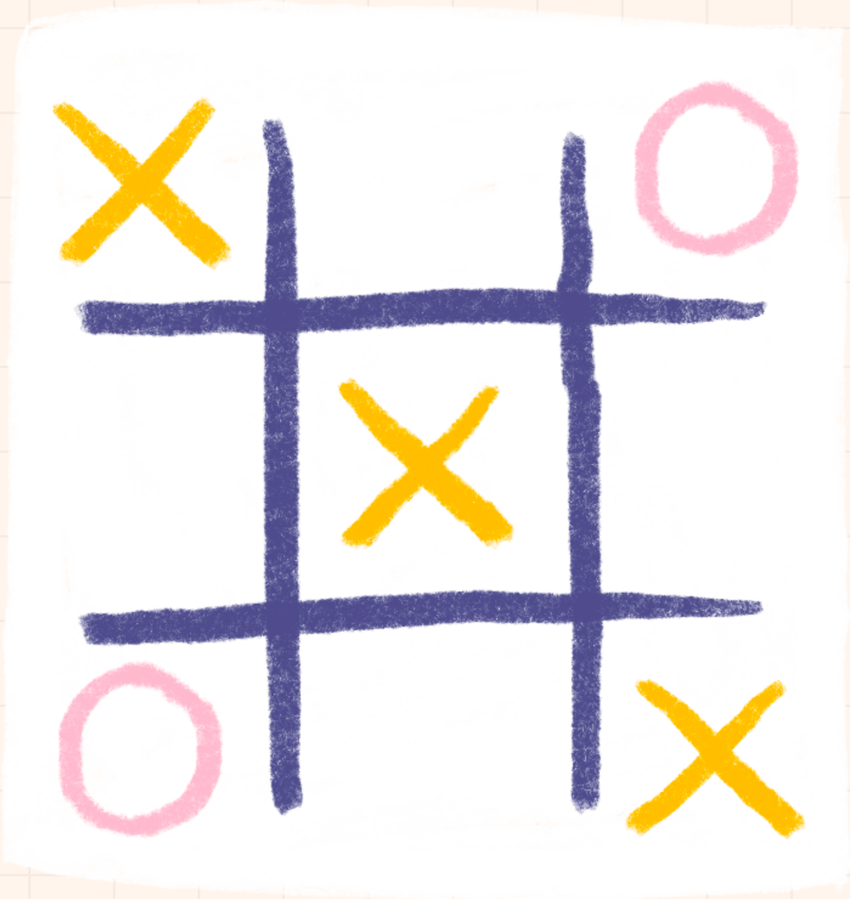


GRINNELL COLLEGE

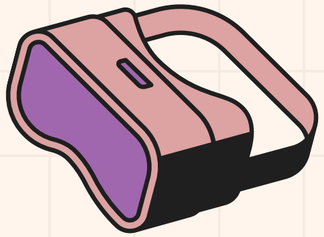
LIVE DEMO

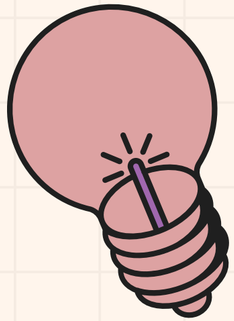
message.c: Handles message communication between server and clients.
socket.h: Helper functions to open server sockets & client connections
server.c: Main logic for tic tac toe, manages game sessions using threads
client.c: Client- side logic that handles player inputs (names and moves) and updating the board

- We found the starter code from the networking excercise to be very beneficial for this



GRINNELL COLLEGE





QUESTIONS