# 22i-1111 // Ibraheem Farrukh

# Report on Ludo Game using TCP Sockets

## Introduction

This project implements a Ludo game where four player processes connect to a central server using TCP sockets. The server orchestrates the game logic, ensuring synchronization and maintaining the state of the game. The client programs represent the players, which communicate with the server to perform actions during the game.

## Technical Architecture

The project uses a client-server model. The server (`server.cpp`) listens for connections from four players, manages the game state, and coordinates interactions. The clients (`player.cpp`) connect to the server, send game moves, and receive updates.

Key components of the architecture:

- \*\*Server:\*\* Handles player connections, game logic, and maintains the game state.  
- \*\*Clients:\*\* Represent individual players, send moves to the server, and display game progress.  
- \*\*Communication Protocol:\*\* Uses TCP sockets for reliable communication.

## Code Analysis

### Server Code

The server is implemented in `server.cpp`. It performs the following tasks:  
- Creates a listening socket to accept player connections.  
- Uses shared memory to maintain the state of the game.  
- Handles game logic, such as turn management, dice rolls, and win conditions.  
- Communicates with clients to send updates and receive player moves.

Snippet of the server code:

#include <netinet/in.h>  
#include <unistd.h>  
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
#include <sys/wait.h>  
#include <sys/shm.h>  
#include <SFML/Graphics.hpp>  
#include <SFML/Main.hpp>  
#include <SFML/Window.hpp>  
using namespace sf;  
#include <iostream>  
using namespace std;  
  
int\* client\_sockets;  
float block\_size = 1224/15;  
float redroute[2][57]={  
 {  
 (block\_size\*1)+10,(block\_size\*2)+10,(block\_size\*3)+10,(block\_size\*4)+10,(block\_size\*5)+10,  
  
 (block\_size\*6)+10,(block

### Client Code

The client program, implemented in `player.cpp`, allows players to:  
- Connect to the server using a socket.  
- Send their moves to the server and receive game updates.  
- Display game progress (potentially using a graphical interface).

Snippet of the client code:

#include <netinet/in.h>  
#include <unistd.h>  
#include <iostream>  
#include <stdio.h>  
#include <string.h>  
#include <cstdlib>  
using namespace std;  
  
int main() {  
  
 srand(time(NULL));  
  
 int sock = socket(AF\_INET, SOCK\_STREAM, 0);  
 if (sock < 0) {  
 perror("Socket creation failed");  
 return 1;  
 }  
  
 struct sockaddr\_in server\_address;  
 server\_address.sin\_family = AF\_INET;  
 server\_address.sin\_port = htons(8000);  
 server\_address.sin\_addr.s\_addr = INADDR\_ANY;  
  
 if (co

## Game Flow

1. The server starts and waits for four players to connect.  
2. Players connect to the server and take turns based on the server's instructions.  
3. The server manages the game state and ensures rules are followed.  
4. Players interact with the game by sending moves and receiving updates.  
5. The game ends when one player wins, and the server notifies all clients.

## Challenges and Future Improvements

Challenges faced during development include:  
- Synchronizing player actions and managing turn order.  
- Handling disconnections and maintaining a consistent game state.  
Future improvements could include:  
- Adding a graphical user interface for players.  
- Implementing more robust error handling.  
- Supporting more players or extending the game rules.

## Conclusion

This project demonstrates the use of TCP sockets to build a networked multiplayer game. It highlights key aspects of client-server communication and game logic implementation, providing a solid foundation for future enhancements.