Dinesh Pamu

B2- 30

**ASSIGNMENT NO: 8**

**Title:**

Distributed Application for Interactive Multiplayer Games

**Problem Statement:**

To design and develop a distributed application for interactive multiplayer games. The application should allow multiple users to connect and interact with each other in a game environment. The system should be able to handle multiple concurrent users and maintain consistency across all clients.

**Tools/Environment:**

The following tools and technologies will be used in this project:

Programming language: Java

IDE: Eclipse

Networking: Java Sockets

**Theory:**

There are mainly two possible network architectures: peer-to-peer and client-server. In the peer-to-peer architecture, data is exchanged between any pair of connected players while in the client-server architecture, data is only exchanged between players and the server.

There are mainly three components in game networking:

* Transport protocol: how to transport the data between clients and the server?
* Application protocol: what to send from clients to the server and from the server to clients and in which format?
* Application logic: how to use the exchanged data to update clients and the server?

To build a distributed application for multiplayer games, we need to consider the following concepts:

Client-Server architecture: The system will have a central server that will handle client requests and send back responses.

Synchronization: The server must maintain consistency across all clients by synchronizing game state data.

Communication: The server must handle communication between clients, sending messages and data as necessary.

Game mechanics: The game mechanics will be designed to support multiplayer interactions, such as real-time updates and player interactions.

**Implementation:**

The implementation of the distributed application for multiplayer games will involve the following steps:

Design the game mechanics and user interface using JavaFX.

Implement the server using Java Sockets, which will handle client requests and maintain game state data.

Implement the client using Java Sockets, which will connect to the server, send requests, and receive responses.

Implement synchronization between the server and clients to ensure consistency of game state data.

Implement communication between clients to allow real-time interactions.

Test the system thoroughly to ensure that it is working as expected.

**Conclusion:**

The development of a distributed application for interactive multiplayer games involves a number of challenges, including synchronization, communication, and game mechanics. By following the steps outlined above, it is possible to design and develop a system that can handle multiple concurrent users and maintain consistency across all clients. This practical assignment will provide students with valuable experience in designing and implementing distributed applications.

**Code:**

**Server-Side Code:**

**import** java.io.\*;

**import** java.net.\*;

**public** **class** GameServer {

**public** **static** **void** main(String[] args) **throws** Exception {

ServerSocket serverSocket = **new** ServerSocket(9027);

System.***out***.println("Server started on port 9027");

**int** numClients = 2; // connect 2 clients

**for** (**int** i = 1; i <= numClients; i++) {

System.***out***.println("Waiting for client " + i + " to connect...");

// Wait for a client to connect

Socket socket = serverSocket.accept();

System.***out***.println("Client " + i + " connected.");

// Start a new thread to handle the client

Thread clientThread = **new** Thread(**new** ClientHandler(socket));

clientThread.start();

}

// Close the server socket

serverSocket.close();

}

}

**class** ClientHandler **implements** Runnable {

**private** Socket socket;

**private** BufferedReader input;

**private** PrintWriter output;

**private** **int** score;

**public** ClientHandler(Socket socket) **throws** Exception {

**this**.socket = socket;

input = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));

output = **new** PrintWriter(socket.getOutputStream(), **true**);

score = 0;

}

**public** **void** run() {

**try** {

**int** numQuestions = 3; // ask 3 questions

**for** (**int** i = 1; i <= numQuestions; i++) {

// Ask a question

String question = "Question " + i + ": Who is the almighty to whom everyone should bow?";

output.println(question);

// Prompt for answer

output.println("Enter your answer:");

// Set a timeout of 10 seconds

socket.setSoTimeout(10000);

// Read the client's answer

String answer = input.readLine();

**if** (answer != **null** && answer.equalsIgnoreCase("tejas kadam")) {

// Correct answer, increase the client's score

score++;

output.println("Correct! You get 1 point.");

} **else** {

output.println("Wrong answer.");

}

}

// Send the client's final score

output.println("Game over. Your score: " + score);

} **catch** (SocketTimeoutException e) {

output.println("Timeout. You did not answer in time.");

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

**try** {

socket.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

}

**Client-side Code:**

**import** java.io.BufferedReader;

**import** java.io.InputStreamReader;

**import** java.io.PrintWriter;

**import** java.net.Socket;

**public** **class** GameClient {

**public** **static** **void** main(String[] args) **throws** Exception {

Socket socket = **new** Socket("localhost", 9027);

BufferedReader input = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));

PrintWriter output = **new** PrintWriter(socket.getOutputStream(), **true**);

**int** numQuestions = 3; // ask 3 questions

**for** (**int** i = 1; i <= numQuestions; i++) {

// Read the question from the server

String question = input.readLine();

System.***out***.println(question);

// Read the prompt for answer

String prompt = input.readLine();

System.***out***.print(prompt);

// Set a timeout of 10 seconds

socket.setSoTimeout(10000);

// Read the client's answer

BufferedReader userInput = **new** BufferedReader(**new** InputStreamReader(System.***in***));

String answer = userInput.readLine();

// Send the answer to the server

output.println(answer);

// Read the server's response

String response = input.readLine();

System.***out***.println(response);

}

// Read the final score from the server

String finalScore = input.readLine();

System.***out***.println(finalScore);

// Close the socket

socket.close();

}

}

**Output:**

