Report

Experiment No : 08

Name of Group Members :(Group No: 04)

Shubham Shankar Sawant (541)

Sarvesh Baburao Borate (505)

Guruprasad Kashinath Dadas (508)

Indrajit Anil Gaikwad (514)

TITLE: MINI PROJECT (Rock-Paper-Scissor)

AIM: To create a Distributed Application for Interactive Multiplayer Games.

TITLE OF GAME:- Rock-Paper-Scissor Game

OBJECTIVE:

1. The objective of the Rock, Paper, Scissors multiplayer game is to outsmart your opponent by selecting a hand gesture that defeats theirs.

2. Players aim to predict and counter their opponents' moves using the hierarchy of gestures: rock beats scissors, scissors beats paper, and paper beats rock.

3. The ultimate objective is to win a series of rounds by consistently making strategic choices and anticipating the opponent's next move.

PROBLEM STATEMENT:-

The problem is to design and implement a multiplayer rock, paper, and scissor game that allows multiple players to compete against each other in real-time. The game should support simultaneous gameplay, handle the selection of moves, and determine the winner based on the game rules. Additionally, the system should be scalable, ensuring smooth gameplay experience for a large number of concurrent players.

TOOLS / ENVIRONMENT:

- S/W:
    1. Can be developed on any platform Windows /Linux.
    2. C/C++/Java
    3. Student can add tools used for implementation like netbeans, eclipse, VS Code,

etc

- H/W:

1. Any basic configuration loaded machine (e.g. P IV)

THEORY:

Distributed Application

A distributed application consists of one or more local or remote clients that communicate with one or more servers on several machines linked through a network. With this type of application, business operations can be conducted from any geographical location. For example, a corporation may distribute the following types of operations across a large region, or even across international boundaries:

- Forecasting sales
- Ordering supplies
- Manufacturing, shipping, and billing for goods
- Updating corporate databases

State of the art telecommunications and data networks are making distributed operations of this sort increasingly common. Applications developed to implement this type of strategy allow businesses to reduce costs and enhance their offerings of services to customers around the world.

Distributed Architecture

1. Client-Server Architecture

Before we get into exactly what different tier systems that are involved with client-server architecture, first we need to understand. What does it entail to create client server architecture for a Java application? The simplest way to have the design of a client server architecture, is to have framework that is segmented into different tasks or workloads that are provided by either a service or a request that are either the server or the service requester (client).

Hence, the name client-server architecture where you have the client who on one side is creating some request in which the server acts as the function of the requests and completes the task. In this scenario, the client is facilitating all communication with the server side therefore the client side does not share any reasons with other clients.

2. Broker Pattern Architecture

So, what is a broker pattern? Basically, a broker pattern application is a distributed system using decoupled components that interact with each other using remote service calls. What makes broker patterns unique is the broker component, which coordinates the communication of the decoupled components. When a client requests a task in an application, the broker receives that task or service and in turn redirects the task to the appropriate server to execute the task.

The main way achieve this design is to create what is called a CORBA. CORBA is a standard practice that is defined by the object management group (OMG) where they help facilitate the communications of technologies for these complex systems.

3. Service-Oriented Architecture

SOA is a design strategy in which applications use services that are available in the network. Each of those services can be formed together to create an application. SOA does have necessary requirements of the following:

- Standardized service contract
- Loose coupling
- Abstraction
- Reusability
- Autonomy
- Discoverability
- Compositability

The requirements above describe the components that make up a SOA. But there are also different roles defined by a SOA application. Those roles can range but in general most are three roles of the service provider, service consumer/requester and service broker.

The roles are pretty self-explanatory but the service provider acts as producer of the information to the service registry, the service broker helps to ensure that the information from the service provider is available to the service consumer. Basically, the responsibilities become a hybrid of both structures where the client-server now has a slightly decoupled service that acts as the broker.

Description of Game:

Rock, Paper, Scissors Multiplayer Game:

1. The Rock, Paper, Scissors multiplayer game is a classic hand game played between two players.

2. The game involves three possible choices: rock, represented by a closed fist, paper, represented by an open hand, and scissors, represented by two fingers extended.

3. Each player simultaneously selects one of the three options by making the corresponding hand gesture.

4. The outcome is determined by the following rules: rock beats scissors, scissors beat paper, and paper beats rock.

5. When both players choose the same option, it results in a tie, and the round is played again.

6. The game continues for multiple rounds until a player reaches a predetermined winning score.

7. A visual representation of the chosen hands and the result is displayed after each round.

8. The game can be played online or in person, allowing players to compete against friends or random opponents.

9. Strategies can be developed to anticipate opponents' choices based on patterns and tendencies.

10. The game offers a simple yet entertaining way to pass the time and engage in friendly competition with others.

IMPLEMENTATION:

Step 1: Set up the game

Create a list of possible moves (rock, paper, and scissors).

Define a function to get the player's move by taking user input and validating it against the list of possible moves.

Define a function to generate the computer's move randomly.

Step 2: Determine the winner

Define a function to compare the player's move and the computer's move and determine the winner based on the rock-paper-scissors rules.

Step 3: Play the game

In a loop, ask each player for their move and determine the winner for each round.

Keep track of the scores for both players.

Step 4: Display the results

After a certain number of rounds or when the players decide to stop, display the final scores and declare the overall winner.

Step 5: Repeat or end the game

Ask the players if they want to play again. If yes, return to step 1. If not, end the game.

Code : Rock, Paper and Scissors Game using Java RMI

File 1 : Interface

```
import java.rmi.Remote;

import java.rmi.RemoteException;

public interface RPSInterface extends Remote {

 int PLAYERS_NUM = 2;

 int ROCK = 1;

 int PAPER = 2;

 int SCISSORS = 3;


 int register() throws RemoteException;

 boolean play(int playerId, int move) throws RemoteException;

 boolean allPlayersReady() throws RemoteException;

 boolean isallPlayersPlayed() throws RemoteException;

 int getResult(int playerId) throws RemoteException;

}
```

File 2: Implementation

```
import java.rmi.RemoteException;

import java.rmi.server.UnicastRemoteObject;

public class RPSImpl extends UnicastRemoteObject implements RPSInterface {

 private int numPlayers;

 private boolean ready;

 private int[] moves;
 private boolean flag;
 public RPSImpl() throws RemoteException {
 super();
 numPlayers = 0;
 ready = false;
flag=false;
 moves = new int[2];
 }
 public int register() throws RemoteException {
 System.out.println("Player " + (numPlayers + 1) + " joined the game.");
 numPlayers++;
 if (numPlayers == 2) {
 ready = true;
 System.out.println("Both players are ready.");
 }
 return numPlayers;
 }
 public boolean allPlayersReady() throws RemoteException {
 System.out.println("Checking if both players are ready.");
 return ready;
 }
 public boolean play(int playerId, int move) throws RemoteException {
 if (moves[playerId - 1] == 0) {
 moves[playerId - 1] = move;
 System.out.println("Player " + playerId + " played " + move + ".");
 return true;
 } else {
 System.out.println("Player " + playerId + " already played.");
 return false;
 }
 }
public boolean isallPlayersPlayed() throws RemoteException {
if(moves[0]!=0 && moves[1]!=0){
flag=true;
}
 System.out.println("Checking if both players are played.");
 return flag;
 }
 public int getResult(int playerId) throws RemoteException {
 //int otherPlayerId = 3 - playerId;
 int result;
 if (moves[0]==moves[1]) {
 System.out.println("It's a tie!");
 return 0;
 }
else if((moves[0]==3 &&moves[1]==2)||(moves[0]==2 &&moves[1]==1)||(moves[0]==1
&&moves[1]==2)){
 System.out.println("Player " + 1 + " wins!");
 return 1;
 }
 else{
 System.out.println("Player " + 2 + " wins!");
 return 2;
}
 }
}
```

File 3 : Server

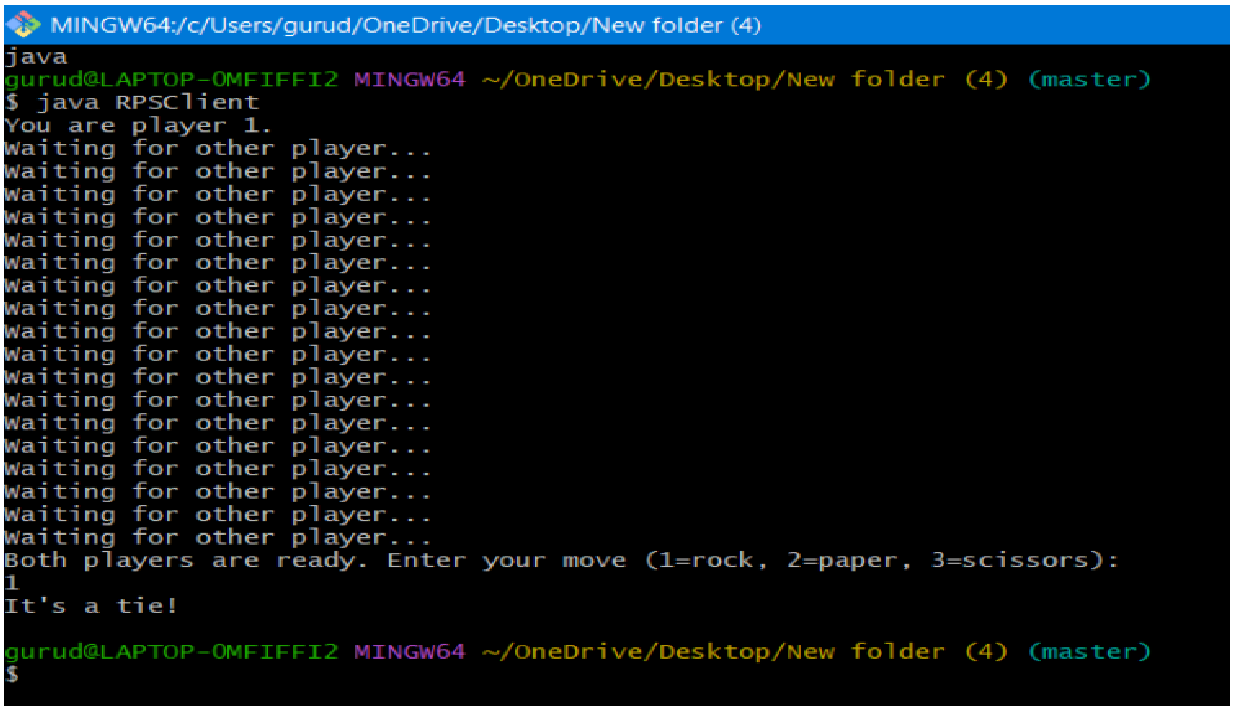
```
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
public class RPSServer {
 public static void main(String[] args) {
 try {
 LocateRegistry.createRegistry(1099);
 RPSImpl game = new RPSImpl();
 Naming.rebind("rmi://localhost/RPSGame", game);
 System.out.println("Server started.");
 } catch (Exception e) {
 System.err.println("Server exception: " + e.toString());
 e.printStackTrace();
 }
 }
}
```

File 4 : Client

```
import java.rmi.Naming;
import java.util.Scanner;
public class RPSClient {
 public static void main(String[] args) {
 try {
 RPSInterface game = (RPSInterface) Naming.lookup("rmi://localhost/RPSGame");
 int playerId = game.register();
 System.out.println("You are player " + playerId + ".");
 Scanner scanner = new Scanner(System.in);
 while (!game.allPlayersReady()) {
 System.out.println("Waiting for other player...");
 Thread.sleep(1000);
 }
 System.out.println("Both players are ready. Enter your move (1=rock, 2=paper,
3=scissors):");
```

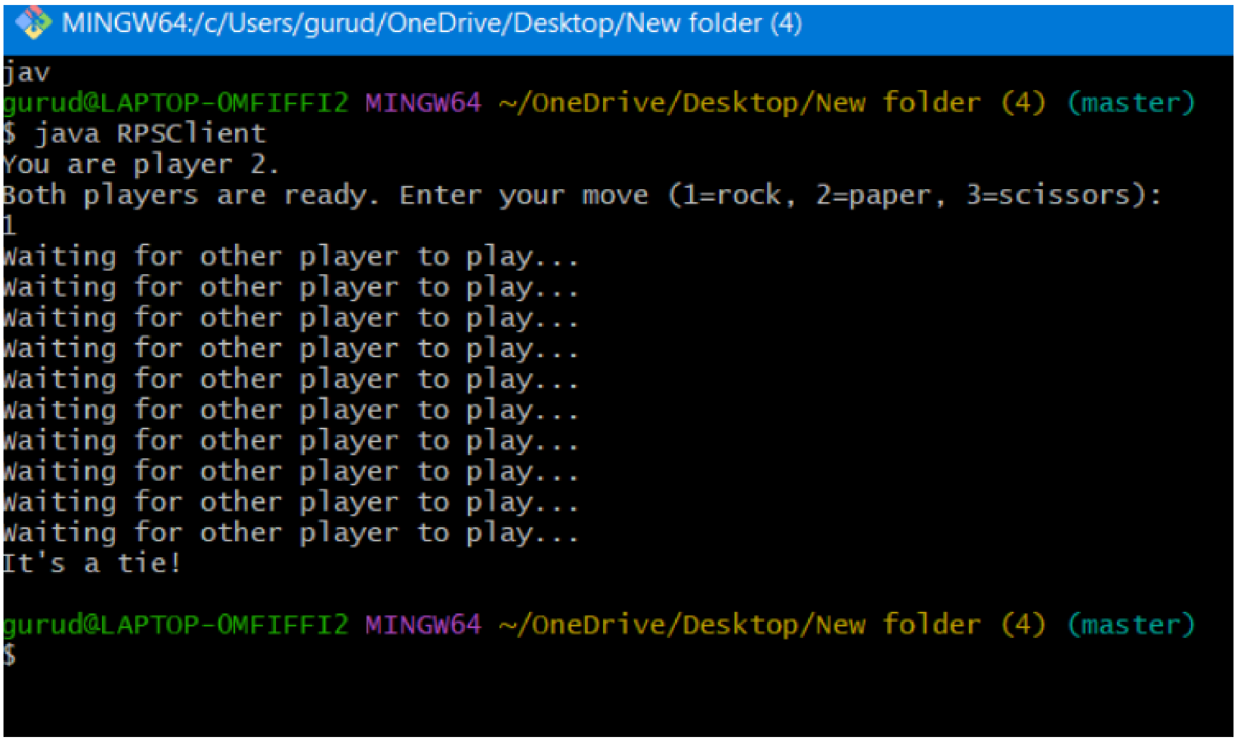
```
 int move = scanner.nextInt();
 boolean success = game.play(playerId, move);
 while (!success) {
 System.out.println("You already played. Enter your move again:");
 move = scanner.nextInt();
 success = game.play(playerId, move);
 }
while (!game.isallPlayersPlayed()) {
 System.out.println("Waiting for other player to play...");
 Thread.sleep(1000);
 }
 int result = game.getResult(playerId);
 if (result == 0) {
 System.out.println("It's a tie!");
 } else {
 System.out.println("Player " + result + " wins!");
 }
 } catch (Exception e) {
 System.err.println("Client exception: " + e.toString());
 e.printStackTrace();
 }
 }
}
```

Outcomes:



```
MINGW64:/c/Users/gurud/OneDrive/Desktop/New folder (4)
java
gurud@LAPTOP-OMFIFFI2 MINGW64 ~/OneDrive/Desktop/New folder (4) (master)
$ java RPSClient
You are player 1.
Waiting for other player...
Waiting for other player...
Waiting for other player...
Waiting for other player...
Waiting for other player...
Waiting for other player...
Waiting for other player...
Waiting for other player...
Waiting for other player...
Waiting for other player...
Waiting for other player...
Waiting for other player...
Waiting for other player...
Waiting for other player...
Waiting for other player...
Waiting for other player...
Waiting for other player...
Waiting for other player...
Both players are ready. Enter your move (1=rock, 2=paper, 3=scissors):
1
It's a tie!

gurud@LAPTOP-OMFIFFI2 MINGW64 ~/OneDrive/Desktop/New folder (4) (master)
$
```



```
MINGW64:/c/Users/gurud/OneDrive/Desktop/New folder (4)
jav
gurud@LAPTOP-OMFIFFI2 MINGW64 ~/OneDrive/Desktop/New folder (4) (master)
$ java RPSClient
You are player 2.
Both players are ready. Enter your move (1=rock, 2=paper, 3=scissors):
1
Waiting for other player to play...
Waiting for other player to play...
Waiting for other player to play...
Waiting for other player to play...
Waiting for other player to play...
Waiting for other player to play...
Waiting for other player to play...
Waiting for other player to play...
Waiting for other player to play...
Waiting for other player to play...
It's a tie!

gurud@LAPTOP-OMFIFFI2 MINGW64 ~/OneDrive/Desktop/New folder (4) (master)
$
```

CONLCUSION:

In conclusion, the multiplayer game of rock, paper, and scissors offers a simple yet engaging experience for players of all ages. Its strategic nature and element of chance make each round thrilling, fostering friendly competition and a sense of anticipation. Whether played casually or competitively, this timeless game continues to captivate and entertain players worldwide.