**Ramrao Adik Institute of Technology**

**(Department of Computer Engineering)**

****

**Mini Project Report**

**On**

**The Rock-Paper-Scissors Game**

**Subject-: Network Programming Laboratory**

***Presented By***

|  |  |  |
| --- | --- | --- |
| Roll No | Batch-Sr. No | Name |
| **14CE1052** | **A1-11** | **Shrey Dhingra** |
| **14CE2019** | **A1-10** | **Shravani Deshmukh** |

***Under the guidance of***

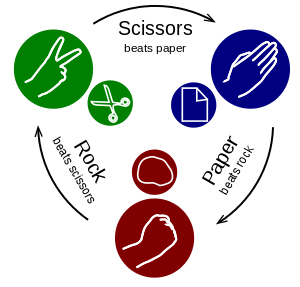
**Prof. Dhanashree Bhosale**

**Signature of Internal Examiner Signature of External Examiner**

**ABSTRACT**

**Rock–paper–scissors** or Scissor-Paper-Rock, is a [zero-sum](https://en.wikipedia.org/wiki/Zero-sum_game) [hand game](https://en.wikipedia.org/wiki/Hand_game) usually played between two people, in which each player simultaneously forms one of three shapes with an outstretched hand. These shapes are "rock" (a simple fist), "paper" (a flat hand), and "scissors" (a fist with the index and middle fingers together forming a V). The game has only two possible outcomes other than a tie: a player who decides to play rock will beat another player who has chosen scissors ("rock crushes scissors") but will lose to one who has played paper ("paper covers rock"); a play of paper will lose to a play of scissors ("scissors cut paper"). If both players choose the same shape, the game is tied and is usually immediately replayed to break the tie.

The game is often used as a choosing method in a way similar to [coin flipping](https://en.wikipedia.org/wiki/Coin_flipping), [drawing straws](https://en.wikipedia.org/wiki/Drawing_straws), or throwing [dice](https://en.wikipedia.org/wiki/Dice). Unlike truly [random](https://en.wikipedia.org/wiki/Randomness) selection methods, however, rock–paper–scissors can be played with a degree of skill by recognizing and exploiting non-random behavior in opponents.



The players usually count aloud to 3, or speak the name of the game (e.g. "Rock Paper Scissors!" ), each time either raising one hand in a fist and swinging it down on the count or holding it behind. They then "throw" by extending it towards their opponent.

**INDEX**

|  |  |  |
| --- | --- | --- |
| Sr no. | Chapter Name | Page No |
| 1 | Introduction | 1 |
| 2 | Implementation | 3 |
| 3 | Java Code | 4 |
| 4 | Screenshots | 10 |
| 5 | Conclusion | 13 |

**INTRODUCTION**

SOCKETS:

* Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else.
* To a programmer, a socket looks and behaves much like a low-level file descriptor. This is because commands such as read() and write() work with sockets in the same way they do with files and pipes.
* Sockets were first introduced in 2.1BSD and subsequently refined into their current form with 4.2BSD. The sockets feature is now available with most current UNIX system releases.

WHERE IS SOCKET USED?

* A Unix Socket is used in a client-server application framework. A server is a process that performs some functions on request from a client. Most of the application-level protocols like FTP, SMTP, and POP3 make use of sockets to establish connection between client and server and then for exchanging data.

SOCKET PROGRAMMING:

* Sockets provide the communication mechanism between two computers using TCP.
* A client program creates a socket on its end of the communication and attempts to connect that socket to a server.
* When the connection is made, the server creates a socket object on its end of the communication.

CLIENT-SERVER COMMUNICATION:

* Normally, a server has a socket that is bound to a specific port number.
* On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening.
* To make a connection request, the client tries to connect with the server on the servers machine and port.
* If everything goes well, the server accepts the connection.
* On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.

**IMPLEMENTATION**

TECHNOLOGIES USED:

* The “RPS Game” project is mainly implemented using JAVA with TCP/IP socket programming and the GUI is built using JFrames.
* The javax.swing package provides classes for java swing API such as JButton, JTextField, JFileChooser etc.
* Networking is supported through the java.net package.

SOCKET PROGRAMMING:

* Sockets provide the communication mechanism between two computers using TCP.
* A client program creates a socket on its end of the communication and attempts to connect that socket to a server.
* When the connection is made, the server creates a socket object on its end of the communication.
* The client and the server can now communicate by writing to and reading from the socket.
* The java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them.

**JAVA CODE FOR SERVER SIDE:**

import java.io.\*;

import java.net.\*;

import java.util.Scanner;

public class Server {

private static Integer port = 1337;

private static Double versionNumber = 1.0;

private static String welcomeMsg = "--- Welcome to Paper Scissors Stone Server V. " + versionNumber + " --- \n";

private static boolean validPort(Integer x) {

return x >= 1 && x <= 65535 ? true : false;

}

private static int getPort() {

Integer input;

Scanner sc = new Scanner(System.in);

do {

System.out.print("Please select a port by entering an integer value between 1 and 65535 or\n");

System.out.print("insert \"0\" in order to continue with the default setting (" + Server.port + "): ");

input = sc.nextInt();

} while (input != 0 && !Server.validPort(input));

sc.close();

return input == 0 ? Server.port : input;

}

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

String resClient\_1 = "";

String resClient\_2 = "";

String inputClient\_1;

String inputClient\_2;

// Print welcome msg

System.out.println(Server.welcomeMsg);

// Set port

Server.port = Server.getPort();

// Create new server socket & dump out a status msg

ServerSocket welcomeSocket = new ServerSocket(Server.port);

System.out.println("\nOk, we're up and running on port " + welcomeSocket.getLocalPort() + " ...");

while (!welcomeSocket.isClosed()) {

// Player one

Socket client\_1 = welcomeSocket.accept();

if (client\_1.isConnected()) {

System.out.println("\nPlayer one (" + (client\_1.getLocalAddress().toString()).substring(1) + ":"

+ client\_1.getLocalPort() + ") has joined ... waiting for player two ...");

}

ObjectOutputStream outClient\_1 = new ObjectOutputStream(client\_1.getOutputStream());

ObjectInputStream inClient\_1 = new ObjectInputStream(client\_1.getInputStream());

// Player two

Socket client\_2 = welcomeSocket.accept();

if (client\_2.isConnected()) {

System.out.println("Player two (" + (client\_2.getLocalAddress().toString()).substring(1) + ":"

+ client\_1.getLocalPort() + ") has joined ... lets start ...");

}

ObjectOutputStream outClient\_2 = new ObjectOutputStream(client\_2.getOutputStream());

ObjectInputStream inClient\_2 = new ObjectInputStream(client\_2.getInputStream());

// Get client inputs

inputClient\_1 = (String)inClient\_1.readObject();

inputClient\_1 = inputClient\_1.toUpperCase();

inputClient\_2 = (String)inClient\_2.readObject();

inputClient\_2 = inputClient\_2.toUpperCase();

if (inputClient\_1.equals(inputClient\_2)) {

resClient\_1 = "Draw";

resClient\_2 = "Draw";

System.out.println("It's a draw.");

}

else if (inputClient\_1.equals("R") && inputClient\_2.equals("S")) {

resClient\_1 = "You win";

resClient\_2 = "You lose";

System.out.println("Player one wins.");

}

else if (inputClient\_1.equals("S") && inputClient\_2.equals("R")) {

resClient\_1 = "You lose";

resClient\_2 = "You win";

System.out.println("Player two wins.");

}

else if (inputClient\_1.equals("R") && inputClient\_2.equals("P")) {

resClient\_1 = "You lose";

resClient\_2 = "You win";

System.out.println("Player two wins.");

}

else if (inputClient\_1.equals("P") && inputClient\_2.equals("R")) {

resClient\_1 = "You win";

resClient\_2 = "You lose";

System.out.println("Player one wins.");

}

else if (inputClient\_1.equals("S") && inputClient\_2.equals("P")) {

resClient\_1 = "You win";

resClient\_2 = "You lose";

System.out.println("Player one wins.");

}

else if (inputClient\_1.equals("P") && inputClient\_2.equals("S")) {

resClient\_1 = "You lose";

resClient\_2 = "You win";

System.out.println("Player two wins.");

}

else

{

resClient\_1 = "Wrong input";

resClient\_2 = "Wrong input";

}

// Send responses in uppercase and close sockets

outClient\_1.writeObject(resClient\_1.toUpperCase());

outClient\_2.writeObject(resClient\_2.toUpperCase());

inputClient\_1 = (String)inClient\_1.readObject();

inputClient\_2 = (String)inClient\_2.readObject();

System.out.println(resClient\_1.toUpperCase());

if(inputClient\_1.equals("N")||inputClient\_2.equals("N")) {

outClient\_1.writeObject("EXIT");

outClient\_2.writeObject("EXIT");

}

else {

outClient\_1.writeObject("DONT EXIT");

outClient\_2.writeObject("DONT EXIT");

}

client\_1.close();

client\_2.close();

}

}

}

**JAVA CODE FOR CLIENT SIDE:**

import java.io.\*;

import java.net.\*;

class Client {

private static String host = "localhost";

private static Integer port = 1337;

private static Double versionNumber = 1.0;

private static String msgWelcome = "--- Welcome to Paper Scissors Stone V. "

+ versionNumber + " --- \n";

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

String input = "";

String response;

BufferedReader inFromUser = new BufferedReader(new InputStreamReader(

System.in));

System.out.println("Enter the server ip address:");

Client.host=inFromUser.readLine();

System.out.println("Enter the port number:");

Client.port=Integer.parseInt(inFromUser.readLine());

System.out.println(Client.msgWelcome);

while(true) {

Socket clientSocket = new Socket(Client.host, Client.port);

ObjectOutputStream outToServer = new ObjectOutputStream(

clientSocket.getOutputStream());

ObjectInputStream inFromServer = new ObjectInputStream(clientSocket.getInputStream());

if (input.equals("-rules")) {

System.out.println(Client.msgRules);

}

// Prompt user for select rock, paper or scissors ...

System.out.println("Start the game by selecting (R)ock (P)aper, (S)cissors");

input = inFromUser.readLine();

// Transmit input to the server and provide some feedback for the user

outToServer.writeObject(input);

System.out.println("\nYour input ("+ input+ ") was successfully transmitted to the server. Now wait for the result ...");

// Catch respones

response = (String)inFromServer.readObject();

// Display respones

System.out.println("Response from server: " + response);

// Close socket

System.out.println("Do you wish to continue playing? (y/n) ");

String ch = inFromUser.readLine();

outToServer.writeObject(ch.toUpperCase());

response = (String)inFromServer.readObject();

if(response.equals("EXIT")) {

System.out.println("GAME OVER");

break;

}

if(ch.equals("N")||ch.equals("n")){

outToServer.close();

inFromServer.close();

clientSocket.close();

break;

}

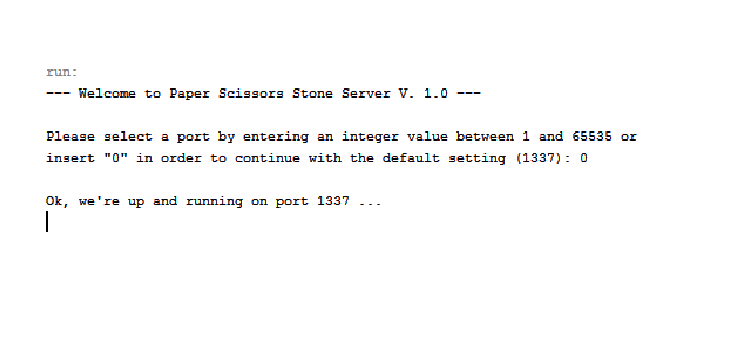
}

}

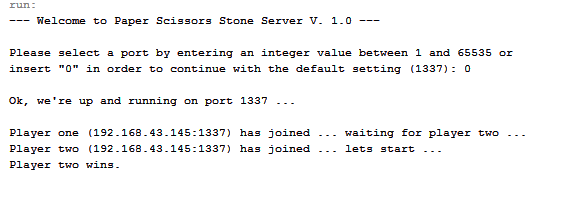
}

**SCREENSHOTS**

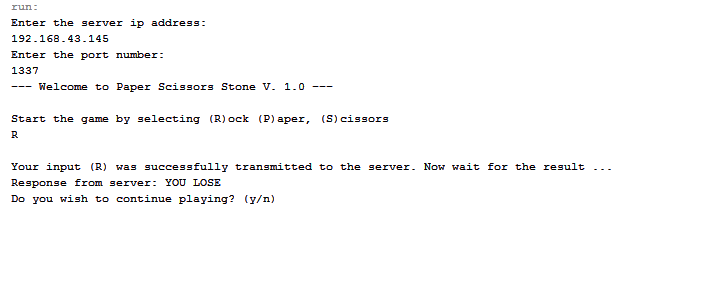
**SERVER SIDE (BEFORE PLAYING):**



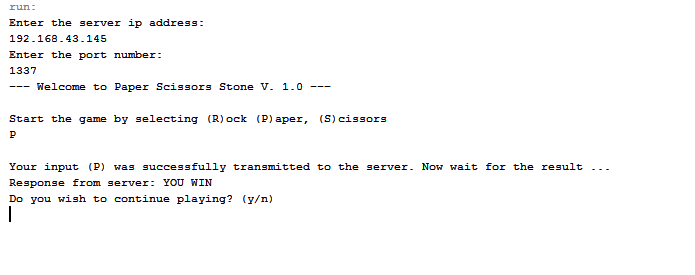
**SERVER SIDE (AFTER PLAYING):**

****

**CLIENT #1 SIDE:**

****

**CLIENT #2 SIDE:**

****

**CONCLUSION**

Thus, THE ROCK-PAPER-SCISSOR game application was developed using Socket Programming in Java programming language.

A **socket** is one of the most fundamental technologies of computer network programming. Sockets allow network software applications to communicate using standard mechanisms built into network hardware and operating systems.

Sockets are used to communicate between two programs. It is generally used to implement TCP/IP or UDP protocol. It generally follows the client server model. We can implement single client single server or multiple clients single server model.

Java API provides Socket and ServerSocket classes as part of java.net package to implement this functionality.

Rock Paper Scissor is a simple two player game. It has purely entertainment based applications. It is simple to play and has a very intuitive GUI.