Rock Paper Scissors Game using Socket Programming

ABHINANDHU A
Dept of Computer Science Engineering (AI)
Amrita School of Engineering ,
Amrita Vishwa Vidyapeetham,
Amritapuri ,
Kollam,Kerala,India
amenu4aie20002@am.students.amrita.edu

MADDALA H S M KRISHNA KARTHIK
Dept of Computer Science Engineering (AI)
Amrita School of Engineering ,
Amrita Vishwa Vidyapeetham,
Amritapuri ,
Kollam,Kerala,India
amenu4aie20046@am.students.amrita.edu

***Abstract*—The complete overview of creating a client-server application implementing socket programming is covered in this research paper. By designing the classic rock paper scissors game, it demonstrates how to develop a multiplayer game with Python and socket programming. One of the best distributed computing techniques for enhancing system performance is socket programming. The main focus of the analysis was on socket-based connections between client-server application processes. Other amazing networking apps and games can be made using the same techniques that were used to create the network game. By developing a networking game or client-server application, the primary goal of this research project is to illustrate the ideas and principles of socket programming.**

***Index Terms*—Sockets,tkinter**

I. Introduction

Nowadays, client-server programming is no longer a novel idea in computer networks and distributed computing. In fact, there are numerous client-server apps on the internet right now that are always accessible. Every computer terminal or operation on the networks could be either a client or a server in a client-server approach. An application or computer terminal that enables users to access and examine its interfaces is referred to as a ”client.” Through the server, each client connects and communicates with the others. In a general sense, the client starts communication, and the server would be the one who listens patiently for the client to make a request.

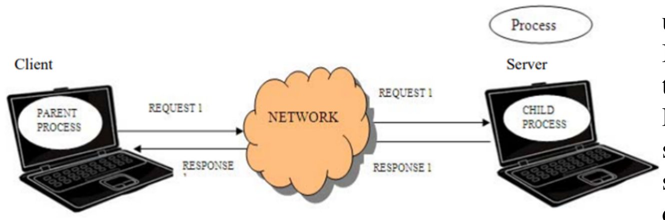


Fig. 1. Client-Server Communication

Socket Programming

A network socket serves as an endpoint for sending and receiving data within the network node of a computer network. The structure and characteristics of a socket are defined by an application programme interface (api) for the networking architecture. In a node-based application, sockets are only created during the course of a process. Since the TCP / ip is the setting in which the term ”network socket” is most frequently used, it is repeatedly alluded to as ”Internet socket.” In this case, a socket is externally identified to other hosts using its socket address, which would be composed of its tcp connection, Ipv4, and port number. Two network nodes can communicate with one another using socket programming. Whereas the other socket makes a connection with it, one socket (or node) waits on a specific port at an IP address. The listener socket is created by the server as the client establishes a connection to it. The majority of application-level protocols, including FTP, SMTP, and POP3, employ sockets to link the client and server and then exchange data.

An abstraction called a socket enables programmes to communicate with other devices. The transparency of socket programming is the key feature that motivates programmers to create a software. It simply means that a socket programme can interact with other socket programmes that are developed and developed in plenty of other programming languages like C or C++, regardless of whether the socket programme was designed and built in Java or Python. Two mechanisms with the same or different machines can interact with one another using sockets.

In socket programming, there are several different socket types. The top four are shown here.

Delivery in a structured setting is guaranteed by stream sockets. TCP is used by these sockets to transmit data. The sender receives an error indicator in the unlikely event that delivery is not feasible.

Datagram Sockets: Distribution in a structured setting isn’t guaranteed. Because you don’t need to have an active session like with Stream Sockets, they are connectionless. They make use of UDP.

Clients have access to the fundamental communication protocols through raw sockets, which enable socket

.

abstractions. Usually, these attachments are datagram-oriented.
With the exception that track boundaries are secured, sequence packed sockets are comparable to stream sockets. In a sense, this interface is provided as a part of the Network Services (NS) socket.

Rock-paper-scissors
The hand game rock-paper-scissors, which originated in China and is often played between two people, requires each player to simultaneously form one of three forms with an extended hand. These forms are called ”rock,” ”paper,” and ”scissors,” respectively. (a clench hand with the list finger and center finger amplified, shaping a V).

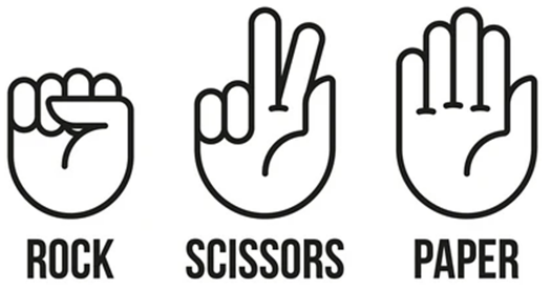


Fig. 2. Three hand shapes

Comparable to coin tossing, plucking straws, or throwing dice, rock paper scissors is frequently used as a fair choice-making method between two people in an effort to end a disagreement or reach a consensus among the group.
The simple game rules for rock, paper, and scissors are as follows:

- Rock wins against scissors.
- Scissors win against the paper.
- Paper wins against the rock.

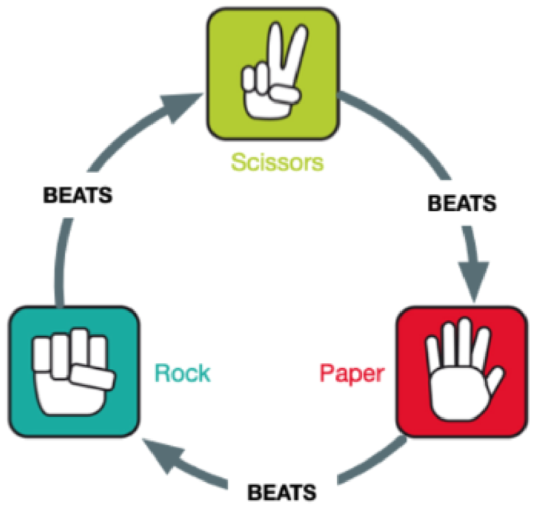
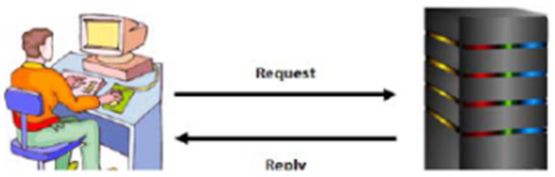


Fig. 3. Game Rule

II. Literature Review

The ideas and tenets of socket programming in a distributed computing environment serve as the foundation for this network gaming application.

A. *Client Server Communication*



Based on its request-reply protocol, the above diagram shows how the client-server concept is used. In this approach, the client directly requests a service from the server, who then performs the requested service and immediately responds to the client with the requested data or error code. Depending on what the client requests, the server provides many facilities to the client [4].

B. *Sockets and Socket Programming*

Socket programming can be used to create client-server applications in distributed computing environments, which are made up of client and server programmes. Client and server programmes operating in a distributed environment can communicate with one another using socket [8]. It has an efficient two-computer communication system.
Socket classes are used in Java programming to describe interactions seen between client and the server [6]. Socket and Server Socket are two classes that Java offers for this use. These sockets link two programmes and carry out the two programme sides (client and server).
The user can utilise the client server programme to launch the client application and formulate a query. The client uses a socket to establish a connection with the specified server and submit a query to it (client side). Once the server has received the client’s request, it evaluates the query and immediately delivers the client the answer.

III. Methodology

Rock-paper-scissors is a multi-user online network game that is created using socket programming with Python. It enables two players to collaborate while playing together by connecting to the server. The rock, paper, or scissors choice can be made by all participants at once. After a predetermined number of rounds, the game is over (6 rounds in our game). After the countdown, the winner is announced and a new game session begins. Tkinter, a Python GUI package, is used for user interfaces.
There are primarily two phases in the rock, paper, scissors network game. They are listed below.

A. *Game Client Application*

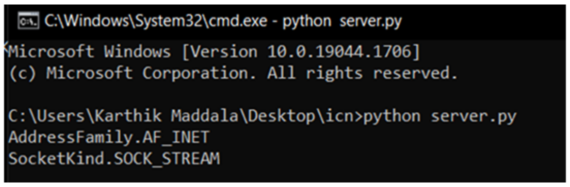
Make a function that initially applies the rock, paper, scissors game rule. A player can play against some other connected player by connecting to the game server through the game client application.
For two connected clients, this application includes a gaming client window (assuming the server is already running). The server returns with a greeting when a player links to it. once the players are both linked. The opponent's name is sent to the game client, which then displays it on the game window. On the gaming client window, the Game Log area is also visible.
We are aware of the opponent once both clients are connected. Game play may now begin. The server sending a clock pulse to the clients is one way for things to get going. When the customer receives the start signal, the game activity can begin. However, it is preferable to conduct features of the game on the client side in order to limit network communications and increase our game performance and efficiency. By doing this, we may prevent message duplication and network congestion, hence lowering network traffic. Therefore, we begin the game session whenever we are certain that our opponent is prepared, or when we receive their name from the server. removing the requirement for server contact to launch the game.
To limit when users can participate in this networking game, we employ a countdown timer and the ability to enable or disable the option buttons. The chosen option is sent to the server, which then sends the opponent's option to the client. The games client window shows this information. It's vital to remember that until we are certain that both players who made their decisions, we do not keep sending player choices. Therefore, the server holds off on sending the information until it has received both options.
Finding the winner of the latest game round is the next stage. One approach is that the server selects the winner and informs the clients of the outcome. This would undoubtedly function, but not quickly or effectively. But our guiding concept is to handle as much logic on the consumer side as feasible in order to minimise server/client connections. Therefore, the outcome of the latest game round will be decided by the client side.
Now, a connect () method is called on the consumer side when a player hits the Connect button.
The connect function runs the connect to server() function, which really establishes the connection to the server, and is straightforward. It makes sure the player inputs a name. The game client programme will then have a main function that receives, decodes, or otherwise reads signals from the main server and modifies the game consumer UI as necessary. Make a method that deals with relaying players' decisions to the server as well.

B. *Game Server Application*

The application running on the game server is in charge of managing connections from and message delivery to client computers. A gaming server window is present in this application. By pressing the start button, the server is launched, and then clients can join and begin streaming. The names of the connected clients are listed in the window. We only allow two connected clients since we only want two people to play our fun game. To manage communications (sending as well as receives signals) with the devices connected, we begin a new thread here.
Create a function now that is in charge of managing client message sending and receiving. The player name will then be sent to the main server when a device connects thanks to a main function we'll build into a game server programme. It then sends the connected client a welcome message. The client for the game can interpret the message and adjust the UI accordingly. We obtain the option and connection (socket) details for each connected client. We then send player1's decision to player2 as well as vice versa after making sure that both players who made their decisions. In this case, the server is not required to be aware of game details like which rounds are being played, who is winning, or the end outcomes. This is due to the fact that things are efficiently handled on the client side.

IV. Results

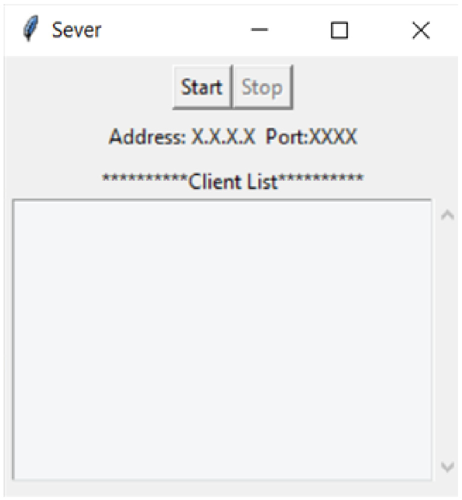
To start the game, launch the server application at the command prompt.



```
C:\Windows\System32\cmd.exe - python  server.py
Microsoft Windows [Version 10.0.19044.1706]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Karthik Maddala\Desktop\icn>python server.py
AddressFamily.AF_INET
SocketKind.SOCK_STREAM
```

The following is the output from executing the server file.

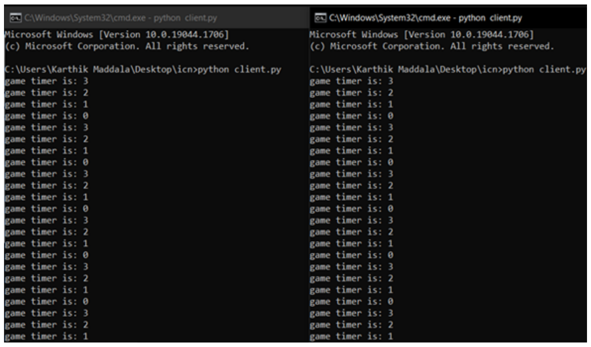


Sever

Address: X.X.X.X Port:XXXX

**********Client List**********

Run two client programmes on the command prompt after that to set up two players.



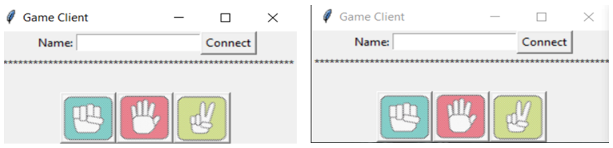
```
C:\Windows\System32\cmd.exe - python client.py
Microsoft Windows [Version 10.0.19044.1706]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Karthik Maddala\Desktop\icn>python client.py
game timer is: 3
game timer is: 2
game timer is: 1
game timer is: 0
game timer is: 3
game timer is: 2
game timer is: 1
game timer is: 0
game timer is: 3
game timer is: 2
game timer is: 1
game timer is: 0
game timer is: 3
game timer is: 2
game timer is: 1
game timer is: 0
game timer is: 3
game timer is: 2
game timer is: 1
game timer is: 0
game timer is: 3
game timer is: 2
game timer is: 1
```

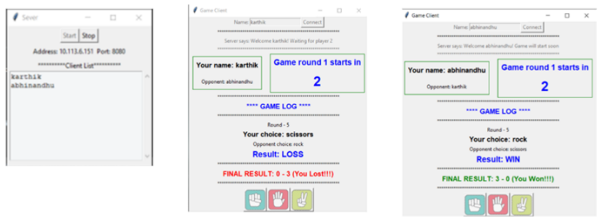
```
C:\Windows\System32\cmd.exe - python client.py
Microsoft Windows [Version 10.0.19044.1706]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Karthik Maddala\Desktop\icn>python client.py
game timer is: 3
game timer is: 2
game timer is: 1
game timer is: 0
game timer is: 3
game timer is: 2
game timer is: 1
game timer is: 0
game timer is: 3
game timer is: 2
game timer is: 1
game timer is: 0
game timer is: 3
game timer is: 2
game timer is: 1
game timer is: 0
game timer is: 3
game timer is: 2
game timer is: 1
game timer is: 0
game timer is: 3
game timer is: 2
game timer is: 1
```

The following are the results of executing the two client files.



The last game's output is shown below.

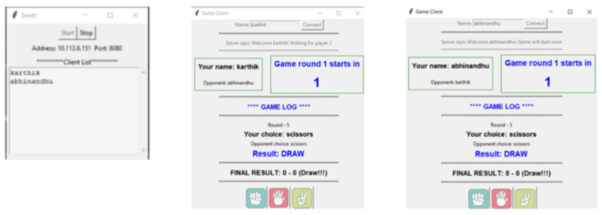


```
Sever
Start Stop
Address: 10.113.6.151  Port: 8080
**********Client List**********
karthik
abhinandhu
```

```
Game Client
Name: karthik  Connect
Server says: Welcome karthik! Waiting for player 2
Your name: karthik
Opponent: abhinandhu
Game round 1 starts in
2
**** GAME LOG ****
Round - 5
Your choice: scissors
Opponent choice: rock
Result: LOSS
FINAL RESULT: 0 - 3 (You Lost!!!)
```

```
Game Client
Name: abhinandhu  Connect
Server says: Welcome abhinandhu! Game will start soon
Your name: abhinandhu
Opponent: karthik
Game round 1 starts in
2
**** GAME LOG ****
Round - 5
Your choice: rock
Opponent choice: scissors
Result: WIN
FINAL RESULT: 3 - 0 (You Won!!!)
```

Following the conclusion of the next game, there is another output.



```
Sever
Start Stop
Address: 10.113.6.151  Port: 8080
**********Client List**********
karthik
abhinandhu
```

```
Game Client
Name: karthik  Connect
Server says: Welcome karthik! Waiting for player 2
Your name: karthik
Opponent: abhinandhu
Game round 1 starts in
1
**** GAME LOG ****
Round - 5
Your choice: scissors
Opponent choice: scissors
Result: DRAW
FINAL RESULT: 0 - 0 (Draw!!!)
```

```
Game Client
Name: abhinandhu  Connect
Server says: Welcome abhinandhu! Game will start soon
Your name: abhinandhu
Opponent: karthik
Game round 1 starts in
1
**** GAME LOG ****
Round - 5
Your choice: scissors
Opponent choice: scissors
Result: DRAW
FINAL RESULT: 0 - 0 (Draw!!!)
```

V. Conclusion And Future scope

The implementation of the well-known rock, paper, scissors game demonstrates how to develop a network game with Python and socket programming. For the purpose of developing network/online games and applications, this project offers best practises and guiding principles. Other amazing networking apps and games can be made using the same techniques that were used to create the network game. Typically, more than two participants can participate in a rock, paper, scissors game. The current network game can only accommodate two players, but it can be made larger by including feature support for extra people. For instance, we could design a networking game that could initially accommodate 5 players. The rules of the game then alter.

References

[1] Maata, Rolou Lyn Cordova, Ronald Sudramurthy, Balaji Halibas, Alrence. (2017). Design and Implementation of Client-Server Based Application Using Socket Programming in a Distributed Computing Environment. 10.1109/ICCIC.2017.8524573.

[2] Z. Yuqing, W. Wu, D. Li, "Efficient Client Assignment for Client-Server Systems", IEEE Transactions on Network and Service Management, Vol. 13 Issue 4, pp. 835-847, August 2016.

[3] M. Xue, C. Zhu, "The Socket Programming and Software Design for Communication Based on Client/Server", IEEE Pacific-Asia Conference on Circuits, Communications and Systems, pp. 775-777, China, May 2009.

[4] Xue, M. and Zhu, C. , "The Socket Programming and Software Design for Communication Based on Client/Server"," 2009 Pacific-Asia Conference on Circuits, Communications and Systems, pp. 774-777, 2009.

[5] Jeff, C. "Sockets and Client-Server Communication", Duke University, [Online] Available: http://db.cs.duke.edu/courses/spring06/cps196/slides/sockets.pdf [Accessed: 02-May-2017]

[6] K. Calvert, M. Donahoo, "TCP/IP Sockets in Java: Practical Guide for Programmers 2nd edition", Morgan Kaufmann, pp. 15-50 USA 2008.

[7] "Socket Programming in .NET", Springer", [Online] Available: https://link.springer.com/chapter/10.1007/978-1-4302-0660-6_4 [Accessed: 17-July-2017]

[8] J. McManis, S. Hashmi, M Ahsan, and J. Haider, "Developing Intelligent Software Interface for Wireless Monitoring of Vehicle Speed and Management of Associated Data", IET Wireless Sensor Systems, 2016.