ROCK PAPER SCISSOR GAME USING SOCKET PROGRAMMING

A COURSE PROJECT REPORT

By

Shashwat Prasad [RA2111026010143]
Roshan Priyadarshi [RA2111026010139]
Munshi Afroze Alam [RA2111026010142]
Mehak Agrawal [RA2111026010152]
Harshitha Nunna [RA2111026010154]

Under the guidance of

Dr.B. Hariharan

(Associate Professor, CINTEL Department)

In partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE ENGINEERING
With specialization in Artificial Intelligence & Machine Learning

Of

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM UNIVERSITY (Under section 3 of UGC Act 1956)

S.R.M. Nagar, Kattankulathur, Chengalpattu District

November, 2023

SRM UNIVERSITY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report titled "**ROCK PAPER SCISSOR GAME USING SOCKET PROGRAMMING**" is the bonafide work of "**SHASHWAT PRASAD [RA2111026010143], ROSHAN PRIYADARSHI [RA2111026010139], MUNSHI AFROZE ALAM [RA2111026010142], MEHAK AGRAWAL [RA2111026010152] & HARSHITHA NUNNA [RA2111026010154]**", who carried out the project work under our supervision. Certified further, that to the best of our knowledge the work reported herein does not form any other project report or dissertation based on which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE	**SIGNATURE**
Dr.B. HARIHARAN	**Dr.R. ANNIE UTHRA**
Associate Professor	Professor (Head of Department)
CINTEL Department	CINTEL Department

TABLE OF CONTENTS

01. ABSTRACT

- A socket is a type of software structure that allows two nodes to communicate in both directions. When it is necessary to create real-time communication, sockets are typically employed.

- For example, Socket is used internally by chat programmes, real-time databases, and online multiplayer games. WhatsApp, a well-known texting programme that leverages Socket for real-time chatting, is one example.

- A single person may play the game Rock Paper Scissors, but in this project, we are going to give it a whole new meaning.

- We have demonstrated this by utilising the most recent programming codes and the incredibly effective TCP/IP protocol.

- The game is being spun to become a “Multiplayer game!”

02. ACKNOWLEDGEMENT

We express our heartfelt thanks to our honourable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavours. We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement. We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V. Gopal,** for bringing out novelty in all executions. We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman,** for imparting confidence to complete my course project. We wish to express my sincere thanks to **Course Audit Professor Dr. Annapurani Panaiyappan, Professor and Head, Department of Networking and Communications** and **Course Coordinators** for their constant encouragement and support. We are highly thankful to my Course project Faculty **Dr.B. Hariharan, Associate Professor, Cintel,** for his/her assistance, timely suggestion, and guidance throughout the duration of this course project. We extend my gratitude to our **HoD Dr.R. Annie Uthra, Professor & Head, Cintel** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

03. INTRODUCTION

- In a distributed computer system (servers), the client-server programming technique divides information producers from information consumers.

- A client is a piece of software that asks a server for resources, such IP addresses or web pages.

- Clients are always welcome to request this information from a server. Customers are information users.

- A server is an application that provides users with access to data or resources. It needs to be always running and prepared to meet customer needs.

- Data interaction is limited to server programmes and their client counterparts. There is no direct interaction between clients.

- Here, our code has been executed using the Python compiler.

- One programming language that may be used to create games is Python. The game "rock, paper, scissors" is made with socket programming and no other libraries.

- The first opportunity to choose between rock, paper, or scissors is given to the player in this game. Next, a player or the computer would choose between rock, paper, or scissors. The winner will receive a point if the winning circumstances are verified. When the game is over, the victor will be announced.

04. REQUIREMENTS

Based on the above situation, we deduce the subsequent prerequisites:

- Determining the suitable hardware to be utilized (PyCharm)
- The code needs to be accessible to internet users.
- Only the server's public IP address should be accessible to internet users; the private IP address should be kept hidden.
- The organization's users ought to have complete access to the server.
- IP addressing in TCP/IP network design
- Hardware setup and features that are necessary, along with an explanation

It is necessary to construct a network architecture with consideration for the following needs.

05. ARCHITECTURE AND DESIGN

This sequence diagram displays the events' chronological order. Before we begin coding, it is critical that you get the flow, so make sure you fully comprehend the diagram.
If you are unfamiliar with sequence diagrams, go over the points below:

- Time is descending.

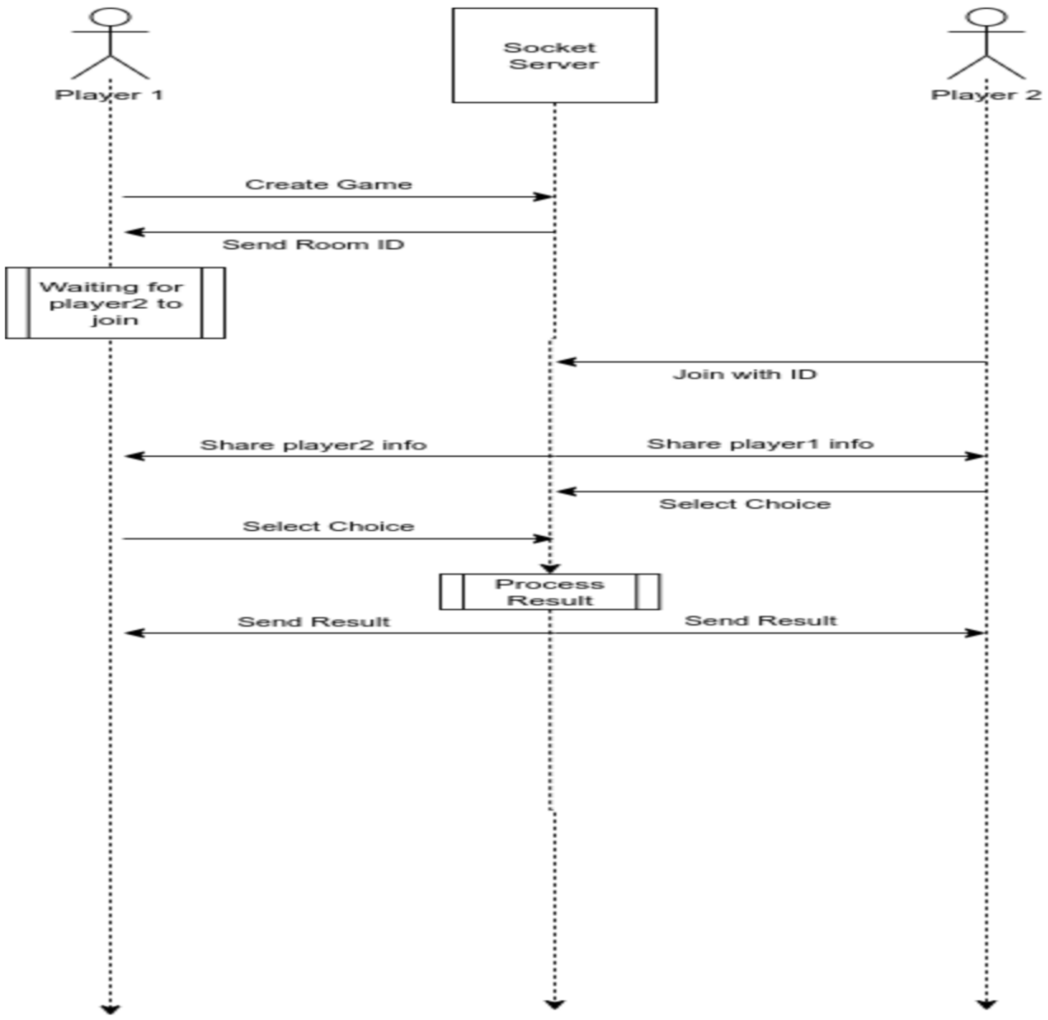
- Arrows stand for events. An arrow's beginning indicates the emitter, while its ending indicates the listener. As an illustration, the Socket Server receives the create Game Event that Player1 transmits.

- A rectangle box represents the processing/waiting time.

Initially, Player 1 sends out a create game event, which the server recognizes and returns with a room ID.

The server recognizes that both players have entered the room when player2 sends out a join game event with the same room ID. At that point, the participants exchange player information.

After both players have selected their option, the server processes the outcome and returns it to them.


Player 1
Socket Server
Player 2
Create Game
Send Room ID
Waiting for player2 to join
Join with ID
Share player2 info
Share player1 info
Select Choice
Select Choice
Process Result
Send Result
Send Result

06. IMPLEMENTATION

1. After entering their name, the client selects between rock, paper, and scissors.

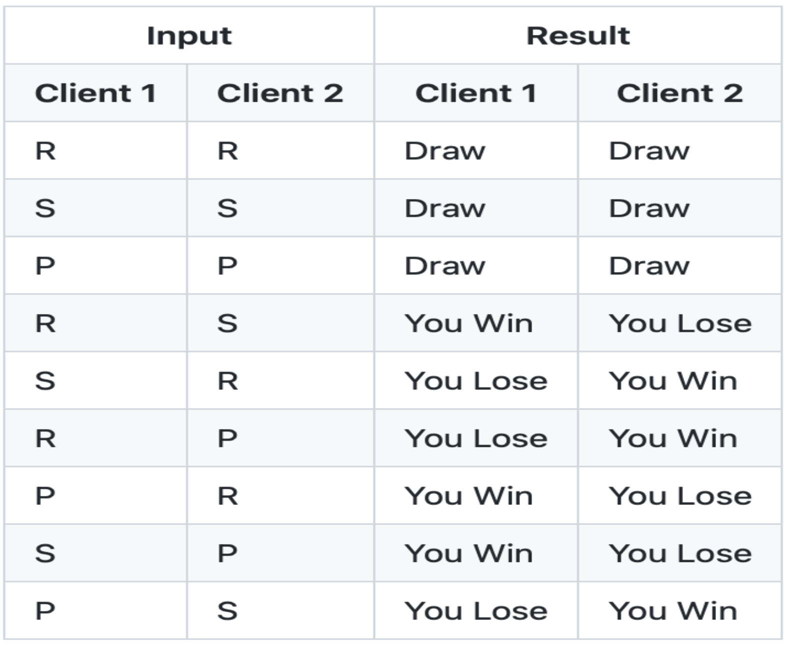
2. The user will be prompted to provide a port number upon launching the server class; this port number must be an integer strictly larger than zero and less than or equal to 65535.

3. As an alternative, the standard port (1337) is accepted with "0".

4. The script sets the port value and does some validation after submission to see whether the value is within the specified range (0 > x <= 65535).

5. As of right now, no additional validation has been applied; for instance, we have not checked to see if the port that was chosen is reserved or just busy.

6. The application outputs a status message to the user informing them that the server is operating on the designated port number if no exception is raised.



Input		Result	
Client 1	**Client 2**	**Client 1**	**Client 2**
R	R	Draw	Draw
S	S	Draw	Draw
P	P	Draw	Draw
R	S	You Win	You Lose
S	R	You Lose	You Win
R	P	You Lose	You Win
P	R	You Win	You Lose
S	P	You Win	You Lose
P	S	You Lose	You Win

07. CODE

- <u>Client Side</u>: -

```
import pygame
from network import Network
import pickle
pygame.font.init()

width = 700
height = 700
win = pygame.display.set_mode((width, height))
pygame.display.set_caption("Client")

class Button:
    def __init__(self, text, x, y, color):
        self.text = text
        self.x = x
        self.y = y
        self.color = color
        self.width = 150
        self.height = 100

    def draw(self, win):
        pygame.draw.rect(win, self.color, (self.x, self.y, self.width, self.height))
        font = pygame.font.SysFont("comicsans", 40)
        text = font.render(self.text, 1, (255,255,255))
        win.blit(text, (self.x + round(self.width/2) - round(text.get_width()/2),
self.y + round(self.height/2) - round(text.get_height()/2)))

    def click(self, pos):
        x1 = pos[0]
        y1 = pos[1]
        if self.x <= x1 <= self.x + self.width and self.y <= y1 <= self.y +
self.height:
            return True
        else:
            return False


def redrawWindow(win, game, p):
```

```
win.fill((128,128,128))

if not(game.connected()):
    font = pygame.font.SysFont("comicsans", 80)
    text = font.render("Waiting for Player...", 1, (255,0,0), True)
    win.blit(text, (width/2 - text.get_width()/2, height/2 - text.get_height()/2))
else:
    font = pygame.font.SysFont("comicsans", 60)
    text = font.render("Your Move", 1, (0, 255,255))
    win.blit(text, (80, 200))

    text = font.render("Opponents", 1, (0, 255, 255))
    win.blit(text, (380, 200))

    move1 = game.get_player_move(0)
    move2 = game.get_player_move(1)
    if game.bothWent():
        text1 = font.render(move1, 1, (0,0,0))
        text2 = font.render(move2, 1, (0, 0, 0))
    else:
        if game.p1Went and p == 0:
            text1 = font.render(move1, 1, (0,0,0))
        elif game.p1Went:
            text1 = font.render("Locked In", 1, (0, 0, 0))
        else:
            text1 = font.render("Waiting...", 1, (0, 0, 0))

        if game.p2Went and p == 1:
            text2 = font.render(move2, 1, (0,0,0))
        elif game.p2Went:
            text2 = font.render("Locked In", 1, (0, 0, 0))
        else:
            text2 = font.render("Waiting...", 1, (0, 0, 0))

    if p == 1:
        win.blit(text2, (100, 350))
        win.blit(text1, (400, 350))
    else:
        win.blit(text1, (100, 350))
        win.blit(text2, (400, 350))

    for btn in btns:
        btn.draw(win)
    pygame.display.update()

btns = [Button("Rock", 50, 500, (0,0,0)), Button("Scissors", 250, 500,
(255,0,0)), Button("Paper", 450, 500, (0,255,0))]
def main():
    run = True
    clock = pygame.time.Clock()
    n = Network()
    player = int(n.getP())
    print("You are player", player)

    while run:
        clock.tick(60)
        try:
            game = n.send("get")
        except:
            run = False
            print("Couldn't get game")
            break

        if game.bothWent():
            redrawWindow(win, game, player)
            pygame.time.delay(500)
            try:
                game = n.send("reset")
            except:
                run = False
                print("Couldn't get game")
                break

            font = pygame.font.SysFont("comicsans", 90)
            if (game.winner() == 1 and player == 1) or (game.winner() == 0 and
player == 0):
                text = font.render("You Won!", 1, (255,0,0))
            elif game.winner() == -1:
                text = font.render("Tie Game!", 1, (255,0,0))
            else:
                text = font.render("You Lost...", 1, (255, 0, 0))

            win.blit(text, (width/2 - text.get_width()/2, height/2 -
text.get_height()/2))
            pygame.display.update()
            pygame.time.delay(2000)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False
                pygame.quit()

            if event.type == pygame.MOUSEBUTTONDOWN:
                pos = pygame.mouse.get_pos()
                for btn in btns:
                    if btn.click(pos) and game.connected():
                        if player == 0:
                            if not game.p1Went:
                                n.send(btn.text)
                        else:
                            if not game.p2Went:
                                n.send(btn.text)

        redrawWindow(win, game, player)

def menu_screen():
    run = True
    clock = pygame.time.Clock()

    while run:
        clock.tick(60)
        win.fill((128, 128, 128))
        font = pygame.font.SysFont("comicsans", 60)
        text = font.render("Click to Play!", 1, (255,0,0))
        win.blit(text, (100,200))
        pygame.display.update()

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                run = False
            if event.type == pygame.MOUSEBUTTONDOWN:
                run = False

    main()

while True:
    menu_screen()
```

- Server Side: -

```
import socket
from _thread import *
import pickle
from game import Game

server = "10.11.250.207"
port = 5555

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    s.bind((server, port))
except socket.error as e:
    str(e)

s.listen(2)
print("Waiting for a connection, Server Started")

connected = set()
games = {}
idCount = 0


def threaded_client(conn, p, gameId):
    global idCount
    conn.send(str.encode(str(p)))

    reply = ""
    while True:
        try:
            data = conn.recv(4096).decode()

            if gameId in games:
                game = games[gameId]

                if not data:
                    break
                else:
                    if data == "reset":
                        game.resetWent()
                    elif data != "get":
```

```
                    game.play(p, data)

                conn.sendall(pickle.dumps(game))
            else:
                break
        except:
            break

    print("Lost connection")
    try:
        del games[gameId]
        print("Closing Game", gameId)
    except:
        pass
    idCount -= 1
    conn.close()



while True:
    conn, addr = s.accept()
    print("Connected to:", addr)

    idCount += 1
    p = 0
    gameId = (idCount - 1)//2
    if idCount % 2 == 1:
        games[gameId] = Game(gameId)
        print("Creating a new game...")
    else:
        games[gameId].ready = True
        p = 1


    start_new_thread(threaded_client, (conn, p, gameId))
```

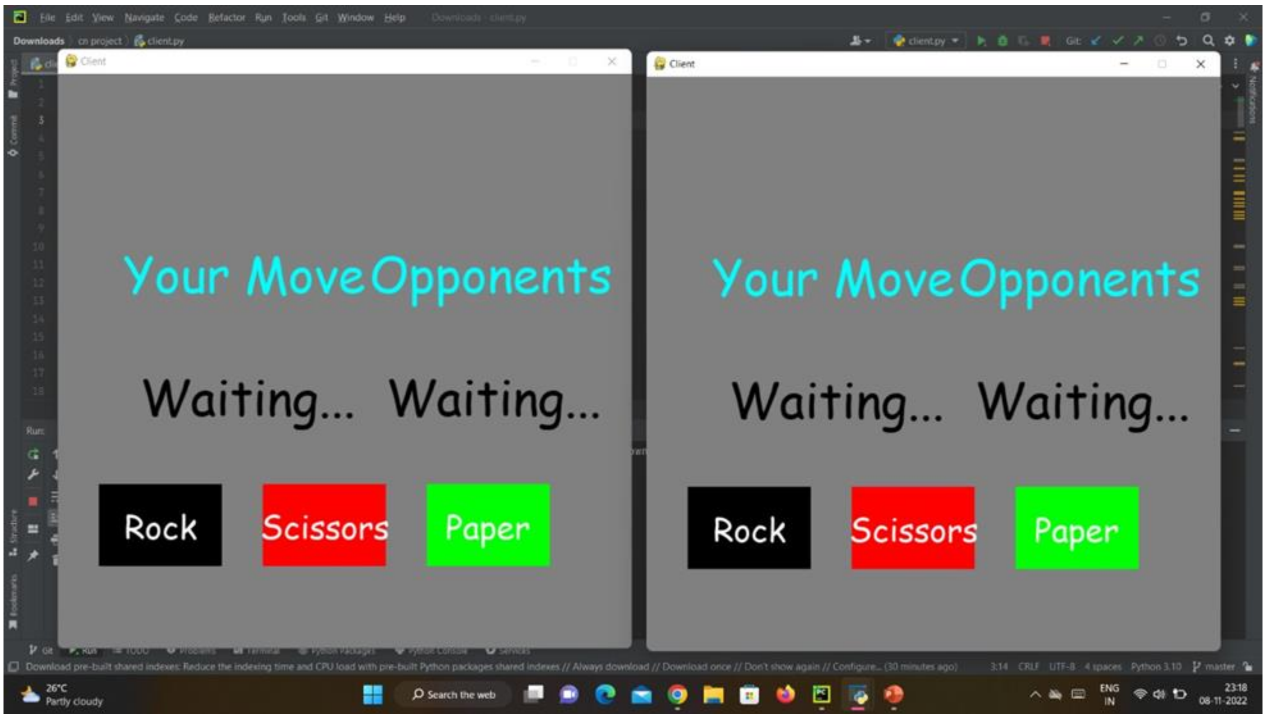
- Game Side: -

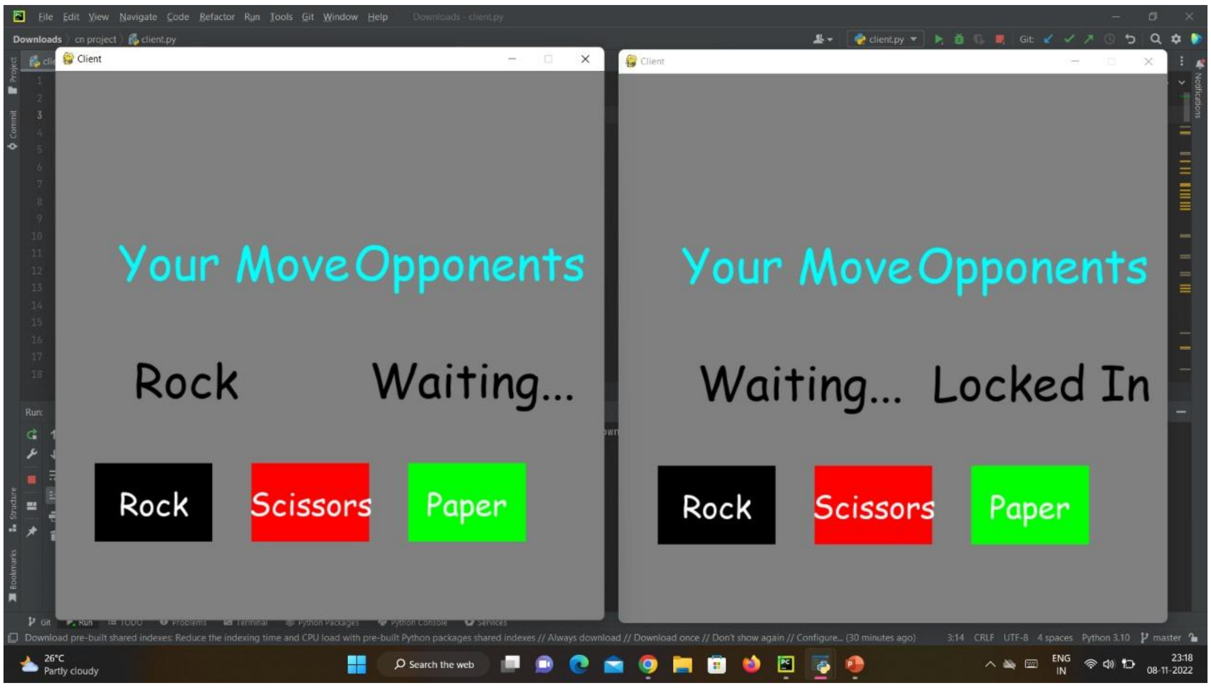
```
class Game:
    def __init__(self, id):
        self.p1Went = False
        self.p2Went = False
        self.ready = False
        self.id = id
        self.moves = [None, None]
        self.wins = [0,0]
        self.ties = 0

    def get_player_move(self, p):
        """
        :param p: [0,1]
        :return: Move
        """
        return self.moves[p]

    def play(self, player, move):
        self.moves[player] = move
        if player == 0:
            self.p1Went = True
        else:
            self.p2Went = True
    def connected(self):
        return self.ready

    def bothWent(self):
        return self.p1Went and self.p2Went

    def winner(self):

        p1 = self.moves[0].upper()[0]
        p2 = self.moves[1].upper()[0]

        winner = -1
        if p1 == "R" and p2 == "S":
            winner = 0
        elif p1 == "S" and p2 == "R":
            winner = 1
        elif p1 == "P" and p2 == "R":
            winner = 0
        elif p1 == "R" and p2 == "P":
            winner = 1
        elif p1 == "S" and p2 == "P":
            winner = 0
        elif p1 == "P" and p2 == "S":
            winner = 1
        return winner

    def resetWent(self):
        self.p1Went = False
        self.p2Went = False
```

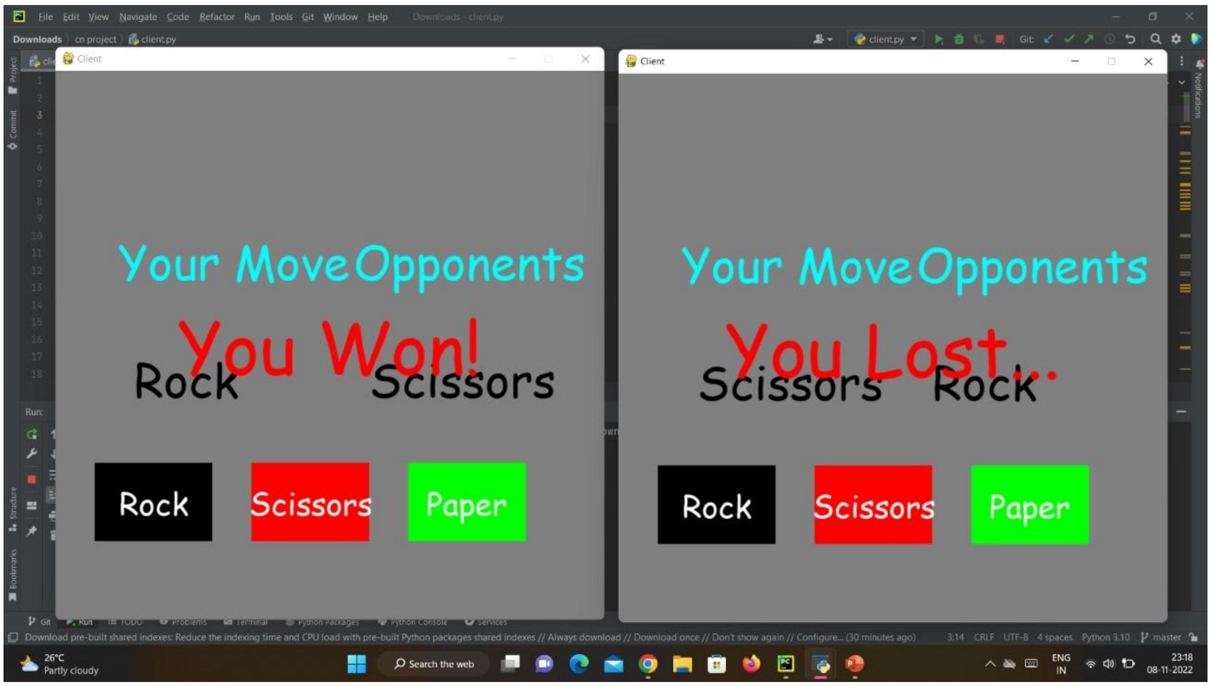
- Network Side: -

```
import socket
import pickle

class Network:
    def __init__(self):
        self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server = "10.11.250.207"
        self.port = 5555
        self.addr = (self.server, self.port)
        self.p = self.connect()

    def getP(self):
        return self.p

    def connect(self):
        try:
            self.client.connect(self.addr)
            return self.client.recv(2048).decode()
        except:
            pass

    def send(self, data):
        try:
            self.client.send(str.encode(data))
            return pickle.loads(self.client.recv(2048*2))
        except socket.error as e:
            print(e)
```

08. Result & Output






Client
Your Move Opponents
You Won!
Rock
Scissors
Rock
Scissors
Paper
Client
Your Move Opponents
You Lost...
Scissors
Rock
Rock
Scissors
Paper

09. CONCLUSION

In conclusion, the socket-programmed version of the Rock, Paper, Scissors game offers an entertaining and engaging way for players to compete against one another in a multiplayer setting. The following are the project's main conclusions and lessons learned:

- Engaging Multiplayer Experience: Through network connections, users may connect to the game and challenge one another to Rock, Paper, Scissors matches. This makes for an exciting multiplayer experience. This increases the sense of rivalry and excitement.

- Networking and Socket Programming: The project demonstrates how to communicate between a central server and many clients using socket programming. Players may make decisions and get real-time game outcomes because to this design.

- Standard Game Rules: Rock, Paper, Scissors is a well-known game with basic rules that participants of all ages may play. It combines the thrill of strategy with the unpredictability of a classic game format.

- Interactive Gameplay: The server effectively controls game logic and communication while offering players an interactive interface via which they may make decisions. Players are kept interested by the instantaneous input on game outcomes.

- Tracking and score: To monitor each player's performance across several rounds, the game has a score system. This gives the game a competitive edge and motivates participants to aim for victory.

- Error Handling and Validation: The solution provides error handling and validation to prevent problems like erroneous movements or multiple login registrations, hence ensuring a seamless and dependable game experience.

- Customization and Extensions: Based on the project's scope, further features can be added to improve the gameplay experience, such as chat support, computer-controlled opponents, numerous games running at once, or a scoreboard.

- Documentation: Detailed instructions are necessary to help users install and launch the game, clarify the rules, and introduce any new or enhanced features.

- Concluding the game: The game should be structured to end gracefully either at the player's request or after a predetermined number of rounds. Making sure the game is simple to start and stop improves the user experience.

In conclusion, the socket-programmed version of the Rock, Paper, Scissors game is a well-rounded project that shows how network communication may be used to create an engaging and competitive gaming environment. It's a great base for multiplayer gaming applications since it implements the basic game and leaves flexibility for modification and growth. The project's capacity to provide gamers with a smooth and pleasurable gaming experience will determine its success.

10. REFERENCE

- Section.io : https://www.section.io/engineering-education/rock-paper-scissor-online/

- Stackoverflow

- Ajith singh- Socket programming with python