```
# Superfox pong.py

import pygame
import math
import sys

class GameStatus(object):
    def __init__(self):
        return dpm.dumps(self, defaultfunc=lambda o: o.__dict__, sort_keys=True, indent=2)

    def __init__(self):'''Init self.'''
        p1x0, p1y0=0,0
        p2x0, p2y0=0,0
        p3x0, p3y0=0,0
        p4x0, p4y0=0,0
        p5x0, p5y0=0,0
        self.ttl=None
        self.lastcmd=None
        self.p1password,p2password,p3password,p4password,p5password=None,None,None,None,None
        self.p1x=p1x0
        self.p1y=p1y0
        self.p1vx=p1vx0
        self.p1vy=p1vy0
        self.p1ammo=p1ammo0
        self.p1ammo_max=p1ammo_max0
        self.p1ammo_rate=p1ammo_rate0
        self.p1ammo_cost=p1ammo_cost0
        self.p1alive=p1alive0
        self.p1dx=p1dx0
        self.p1dy=p1dy0
        self.p1dvx=p1dvx0
        self.p1dvy=p1dvy0
        self.p1dammo=p1dammo0
        self.p1dammo_max=p1dammo_max0
        self.p1dammo_rate=p1dammo_rate0
        self.p1dammo_cost=p1dammo_cost0
        self.p1dalive=p1dalive0
        self.p1dwidth=p1dwidth0
        self.p1dheight=p1dheight0
        self.p1dammo_width=p1dammo_width0
        self.p1dammo_height=p1dammo_height0
        self.p1dammo_vx=p1dammo_vx0
        self.p1dammo_vy=p1dammo_vy0
        self.p1dammo_dmg=p1dammo_dmg0
        self.p1dammo_dmg_rate=p1dammo_dmg_rate0
        self.p1dammo_dmg_cost=p1dammo_dmg_cost0
        self.p1dammo_dmg_max=p1dammo_dmg_max0
        self.p1dammo_dmg_min=p1dammo_dmg_min0
        self.p1dammo_dmg_dmg=p1dammo_dmg_dmg0
        self.p1dammo_dmg_dmg_rate=p1dammo_dmg_dmg_rate0
        self.p1dammo_dmg_dmg_cost=p1dammo_dmg_dmg_cost0
        self.p1dammo_dmg_dmg_max=p1dammo_dmg_dmg_max0
        self.p1dammo_dmg_dmg_min=p1dammo_dmg_dmg_min0
        self.p1dammo_dmg_dmg_dmg=p1dammo_dmg_dmg_dmg0
        self.p1dammo_dmg_dmg_dmg_rate=p1dammo_dmg_dmg_dmg_rate0
        self.p1dammo_dmg_dmg_dmg_cost=p1dammo_dmg_dmg_dmg_cost0
        self.p1dammo_dmg_dmg_dmg_max=p1dammo_dmg_dmg_dmg_max0
        self.p1dammo_dmg_dmg_dmg_min=p1dammo_dmg_dmg_dmg_min0
        self.p1dammo_dmg_dmg_dmg_dmg=p1dammo_dmg_dmg_dmg_dmg0
        self.p1dammo_dmg_dmg_dmg_dmg_rate=p1dammo_dmg_dmg_dmg_dmg_rate0
        self.p1dammo_dmg_dmg_dmg_dmg_cost=p1dammo_dmg_dmg_dmg_dmg_cost0
        self.p1dammo_dmg_dmg_dmg_dmg_max=p1dammo_dmg_dmg_dmg_dmg_max0
        self.p1dammo_dmg_dmg_dmg_dmg_min=p1dammo_dmg_dmg_dmg_dmg_min0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg=p1dammo_dmg_dmg_dmg_dmg_dmg0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_rate=p1dammo_dmg_dmg_dmg_dmg_dmg_rate0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_cost=p1dammo_dmg_dmg_dmg_dmg_dmg_cost0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_max=p1dammo_dmg_dmg_dmg_dmg_dmg_max0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_min=p1dammo_dmg_dmg_dmg_dmg_dmg_min0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_rate=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_rate0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_cost=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_cost0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_max=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_max0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_min=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_min0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_max=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_max0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_min=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_min0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_max=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_max0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_min=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_min0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_max=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_max0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_min=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_min0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_max=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_max0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_min=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_min0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_max=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_max0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_min=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_min0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_max=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_max0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_min=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_min0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_max=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_max0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_min=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_min0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_max=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_max0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_min=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_min0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_max=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_max0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_min=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_min0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_max=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_max0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_min=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_min0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_rate0
        self.p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost=p1dammo_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_dmg_cost0
        self.p1dammo_dmg
```

```python
# ServerProgram: server.py

import socket
import threading
import os
import time
from dataclasses import dataclass

import pygame
from pong4 import *

# IP = socket.gethostbyname(socket.gethostname())
IP = ''
PORT = 53535
ADDR = (IP, PORT)
SIZE = 4096
FORMAT = "utf-8"
DISCONNECT_MESSAGE = "QUIT!"

clients = [] # list of clients connected to the server

@dataclass
class Client:
    '''Client class to store client information'''
    conn: socket.socket
    addr: str
    player: int


def get_next_player():
    all_players = sorted([c.player for c in clients])
    player = 1
    for p in all_players:
        if p == player:
            player += 1
        else:
            break
    return player


def players_full():
    return ((len(clients) >= 4 and gs.FourPlayers) or
            (len(clients) >= 2 and not gs.FourPlayers))


def disconnect_client(client: Client):
    '''Disconnect client'''
    global gs
    print(f"[DISCONNECT CLIENT] {client.addr} disconnected.")
    print(f"[ACTIVE CLIENTS] {threading.active_count() - 3}")

    gs.paused = True
    clients.remove(client)
    client.conn.close()


def handle_msg(client: Client, msg: str):
    print(f"[{client.addr}] {msg}")
    if "GET" in msg:
        client.conn.send(gs.toJSON().encode(FORMAT))
    else:
        key, press = msg.split(":")
        keys_map = {
            1: {
                "left": pygame.K_w,
                "right": pygame.K_s,
            },
            2: {
                "left": pygame.K_UP,
                "right": pygame.K_DOWN,
            },
            3: {
                "left": pygame.K_a,
                "right": pygame.K_d,
            },
            4: {
                "left": pygame.K_LEFT,
                "right": pygame.K_RIGHT,
            },
        }
        presses_map = {
            "down": pygame.KEYDOWN,
            "up": pygame.KEYUP,
        }
        handle_events(presses_map[press], keys_map[client.player][key])


def handle_client(client: Client):
    addr = client.addr
    conn = client.conn
    print(f"[NEW CLIENT] {addr} connected.")

    conn.send(f"{client.player}".encode(FORMAT))

    while True: # recieve message from client
        from_msg = conn.recv(SIZE).decode(FORMAT)
        if from_msg == DISCONNECT_MESSAGE:
            break
        elif not from_msg:
            continue

        try: # handle recieved message
            while ";" in from_msg:
                msg, from_msg = from_msg.split(";", 1)
                handle_msg(client, msg)
        except Exception as e:
            print(f"[ERROR] {e}")

    try:
        disconnect_client(client)
    except Exception as e:
        pass


def server_broadcast(msg: str):
    for client in clients:
        client.conn.send(msg.encode(FORMAT))


def server_loop():
    print(f"[STARTING] Server is starting...")

    # create socket, bind to address, and listen
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(ADDR)
    server.listen()
    print(f"[LISTENING] Server is listening on {IP}:{PORT}")

    print(f"[ACTIVE CLIENTS] {threading.active_count() - 2}")

    global gs
    gs.paused = True

    while True: # accept new connection
        conn, addr = server.accept()
        addr = f"{addr[0]}:{addr[1]}"

        if players_full():
            conn.send(DISCONNECT_MESSAGE.encode(FORMAT))
            continue

        current_client = Client(conn, addr, get_next_player())
        clients.append(current_client)

        # start new thread to handle client
        thread = threading.Thread(target=handle_client, args=(current_client,))
        thread.start()

        print(f"[ACTIVE CLIENTS] {threading.active_count() - 2}")

        time.sleep(0.1)
        if gs.paused and players_full():
            gs.paused = False



def main():
    game_thread = threading.Thread(target=game_loop, args=(True,))
    game_thread.start()

    server_loop()


if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt: # handle keyboard interrupt
        print("\n[EXITING] Server is shutting down...")
        for client in clients: # send disconnect message to all clients
            client.conn.close()
        os._exit(0)
    except OSError as e:
        print(f"[ERROR] {e}")
        os._exit(0)
```

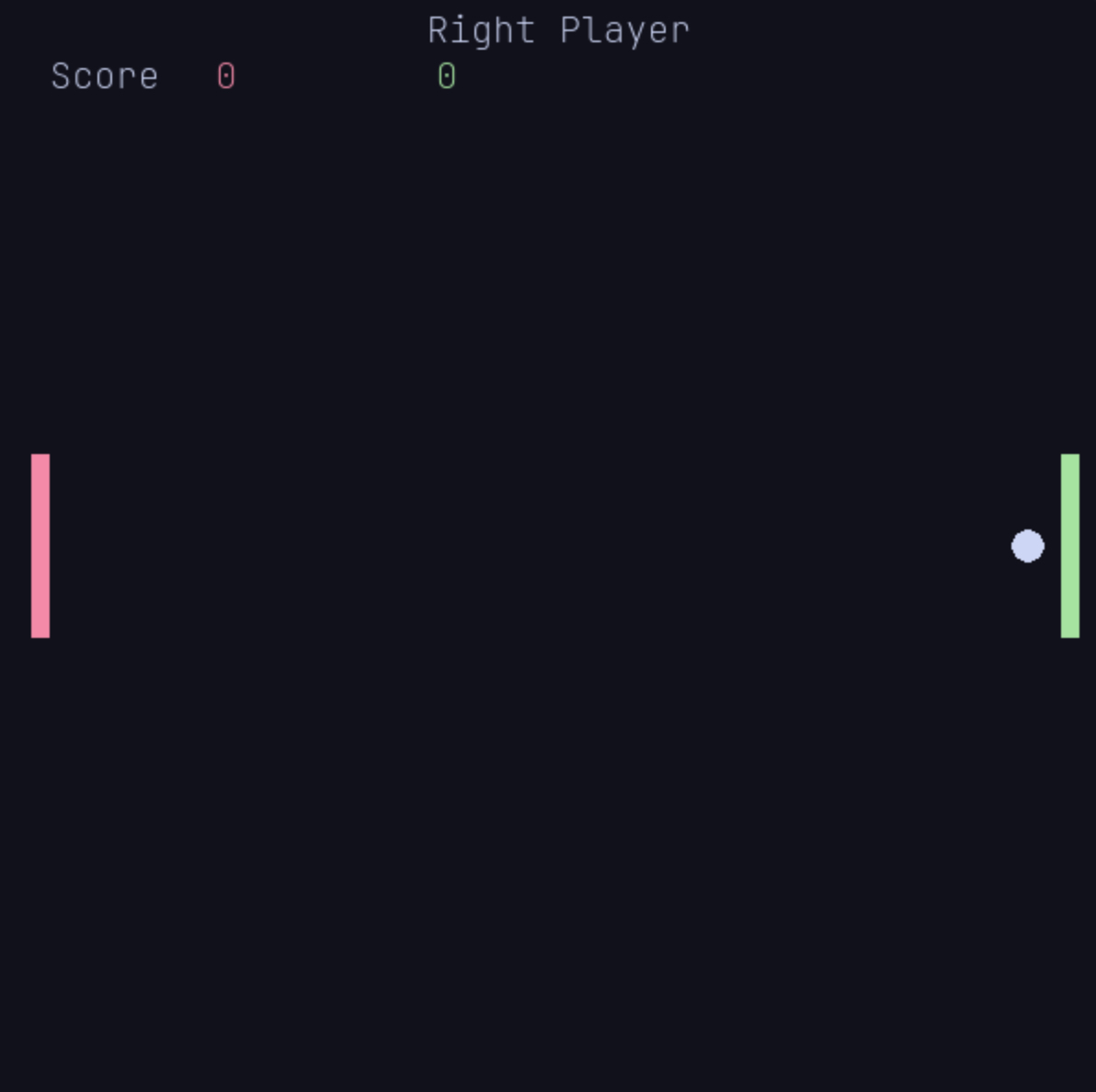
```python
# ClientProgram: client.py

import socket
import threading
import os

import pygame
from pong4 import *

IP = socket.gethostbyname(socket.gethostname())
# IP = '192.168.12.219'
PORT = 53535
ADDR = (IP, PORT)
SIZE = 4096
FORMAT = "utf-8"
DISCONNECT_MESSAGE = "QUIT!"

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
player = 0


def disconnect_server(client: socket.socket, recv_from: str):
    client.send(DISCONNECT_MESSAGE.encode(FORMAT))
    if recv_from == "client":
        print(f"[DISCONNECTED] Client disconnected from {IP}:{PORT}")
    elif recv_from == "server":
        print(f"[DISCONNECTED] Server disconnected from Client.")
    client.close()
    os._exit(0)


def recv_msg(client: socket.socket, disconnect_info: str = ""):
    msg = client.recv(SIZE).decode(FORMAT)
    if msg == DISCONNECT_MESSAGE:
        print(disconnect_info)
        disconnect_server(client, "server")
    return msg


def update_loop():
    global gs
    prev_paused = False
    while True:
        try:
            client.send("GET;".encode(FORMAT))

            msg = recv_msg(client)

            gs = GameState.from_json(msg)
        except (BrokenPipeError, ConnectionResetError):
            print(f"[EXCEPTION] Connection error: Server not connected")
            os._exit(0)
        except json.decoder.JSONDecodeError:
            print("[EXCEPTION] JSON Decode Error")
            continue

        if gs.paused:
            screen.blit(
                font.render("Waiting for other players...", True, WHITE),
                (gs.W//2-150,gs.H//2)
            )
            pygame.display.flip()
            prev_paused = True
            continue

        if prev_paused and not gs.paused:
            prev_paused = False
            game_countdown()

        screen.fill(BLACK)

        tag_pos = 50 if gs.FourPlayers else 5
        screen.blit(
            font.render(f"{pl[player]} Player", True, WHITE),
            (gs.W//2-65,tag_pos)
        )
        drawscore(screen, font, gs.H, gs.FourPlayers, gs)
        drawball(screen, gs.bx, gs.by, gs.bw)

        drawpaddle(screen,
                   gs.p1x, gs.p1y,
                   gs.paddle_width_v, gs.paddle_height_v,
                   py1_Color
        )
        drawpaddle(screen,
                   gs.p2x, gs.p2y,
                   gs.paddle_width_v, gs.paddle_height_v,
                   py2_Color
        )

        if gs.FourPlayers:
            drawpaddle(screen,
                       gs.p3x, gs.p3y,
                       gs.paddle_width_h, gs.paddle_height_h,
                       py3_Color
            )
            drawpaddle(screen,
                       gs.p4x, gs.p4y,
                       gs.paddle_width_h, gs.paddle_height_h,
                       py4_Color
            )

        pygame.display.flip()

        if gs.winner != 0:
            screen.blit(
                font.render(f"{pl[gs.winner]} Player Wins!", True, WHITE),
                (gs.W//2-100,gs.H//2)
            )
            if gs.winner == player:
                screen.blit(
                    font.render("You Win!", True, WHITE),
                    (gs.W//2-50,gs.H//2-50)
                )
            else:
                screen.blit(
                    font.render("You Lose!", True, WHITE),
                    (gs.W//2-50,gs.H//2-50)
                )
            pygame.display.flip()
            break
        # pygame.time.wait(wt)


def game_loop():
    global gs, client

    keys = {
        pygame.K_w: "left",
        pygame.K_s: "right",
        pygame.K_UP: "left",
        pygame.K_DOWN: "right",
        pygame.K_a: "left",
        pygame.K_d: "right",
        pygame.K_LEFT: "left",
        pygame.K_RIGHT: "right",
    }

    while True:
        try:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    disconnect_server(client, "client")
                if event.type == pygame.KEYDOWN:
                    if event.key in keys:
                        key = keys[event.key]
                        client.send(f"{key}:down;".encode(FORMAT))
                elif event.type == pygame.KEYUP:
                    if event.key in keys:
                        key = keys[event.key]
                        client.send(f"{key}:up;".encode(FORMAT))
        except BrokenPipeError:
            print(f"[EXCEPTION] Broken pipe error: Server not connected")
            os._exit(0)
        # pygame.time.wait(wt)


def main():
    # connect to server
    global client, player, gs

    client.connect(ADDR)
    print(f"[CONNECTED] Client connected to {IP}:{PORT}")

    # Player number
    msg = recv_msg(client)
    player = int(msg)
    print(f"[PLAYER] Player {player}")

    update_thread = threading.Thread(target=update_loop)
    update_thread.start()

    game_loop()

    disconnect_server(client, "client")

if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt:
        print("[EXCEPTION] Keyboard Interrupt.")
        disconnect_server(client, "client")
    except ConnectionRefusedError:
        print(f"[EXCEPTION] Connection Refused: Server not connected.")
        os._exit(0)
```

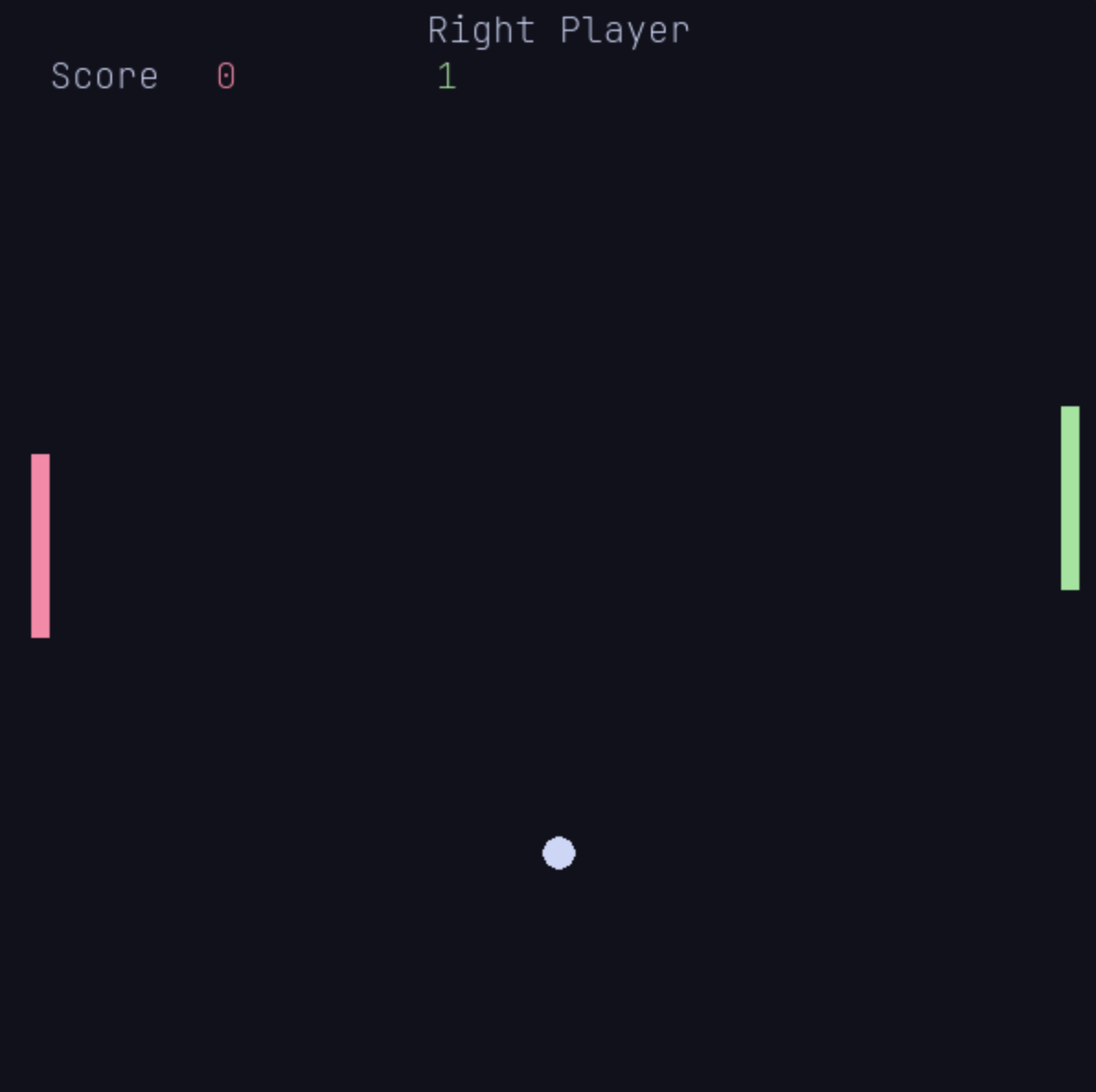
Waiting for other players...

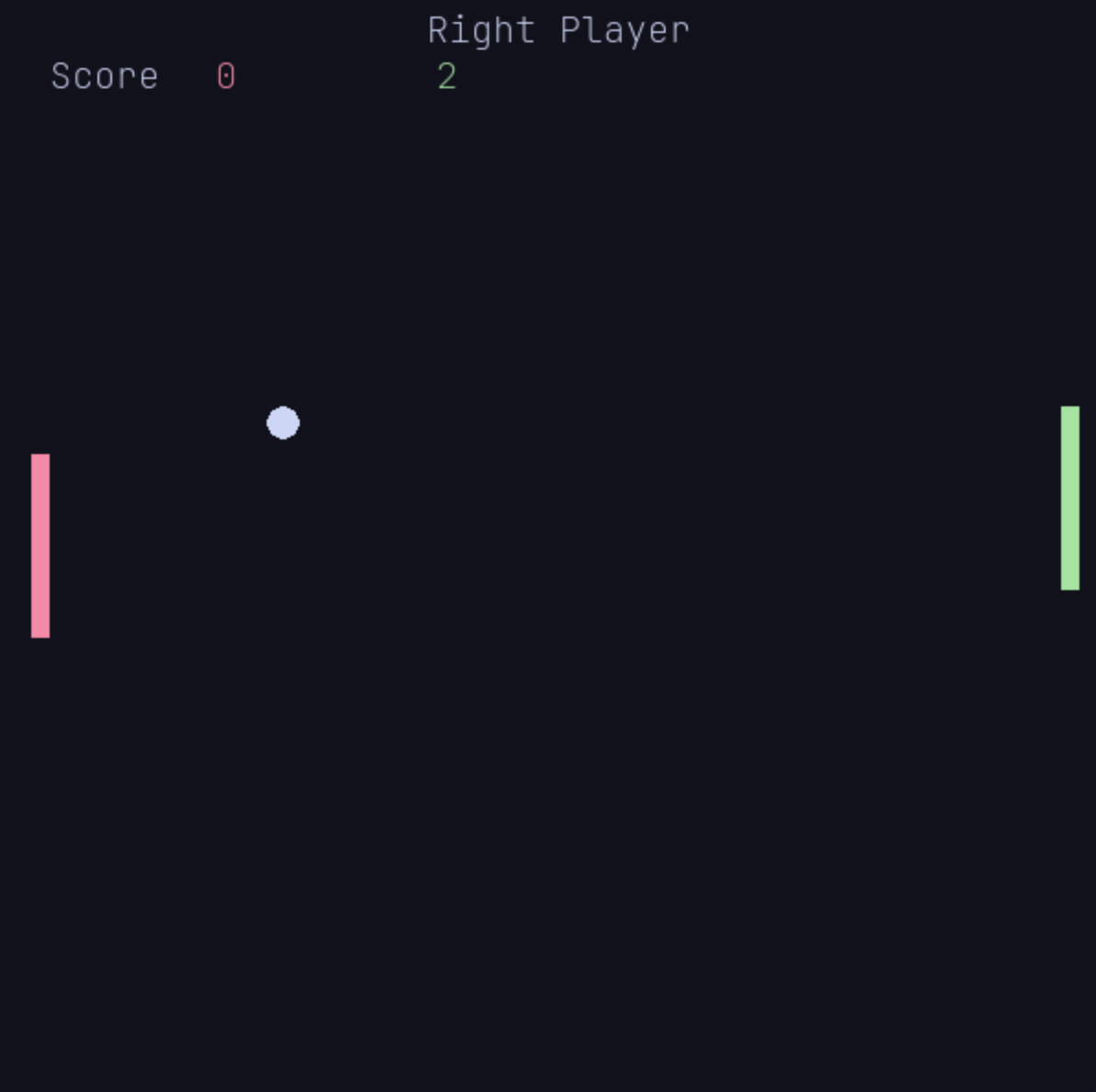
Starting Game in 2...

Right Player

Score 0 0




Right Player
Score 0
1


Right Player
Score 0
2

Right Player

Score 0

3

You Win!

Right Player Wins!