COMPUTER NETWORKS PROJECT

SUBJECT CODE - UE17EC351

Agar io Multiplayer Game Using Socket Programming

Submitted by

MAHIMA BHUSHAN – PES1201701617

SAI SUSHMITHA Y – PES1201700826

Table of Contents

1. PROBLEM STATEMENT

Gaming is one of those activities we can do with people without having to physically be present. And since we are all physically distancing ourselves from one another right now, there is no better way to help keep each other amused than by playing some games together. Hence, our project aims to build a simple multiplayer game using socket programming and pygame.

2. INTRODUCTION

As the title suggests it is a game entirely coded in python language using sockets, networking and the python module pygame. Socket programming is a way of connecting two nodes on a network to communicate with each other. Here, these two nodes of the network are clients or in specific players communicating with the help of server. In this game a single server is run by the main user (again using his/her mobile hotspot) and multiple clients (players) connect to them using IPv6 addressing .We have implemented this game in a cross-platform set of Python modules designed for writing video games called the pygame. It includes computer graphics designed to be used with Python programming language.

3. CONFIGURATION

A duplication of the game Agar.io written in python language. The server and client programs are coded using Socket Programming.

3.1 SERVER CONFIGURATION

The server setup is as follows:

- A laptop/PC connected wirelessly using personal mobile hotspot that is connected to an active GPRS network.
- Its Public IPv6. (128-bit address)
- A port number common to both server and client.
- Python 3.8.x installed along with its IDE.
- The python file named “server.py”.

3.2 CLIENT CONFIGURATION

The client setup is as follows:

- A laptop/PC connected wirelessly using personal mobile hotspot.
- Server’s Public IPv6 address.
- A port number common to both server and client.
- Python 3.8.x installed along with its IDE.
- The python file named “client.py”.

4 COMPARING IPv4 and IPv6 ADDRESSING

As this a project related to gaming which involves multiple players from outside world, we need to use Network Address Translation (NAT). The primary use of this is to conserve public IPv4 addresses. It does this by allowing networks to use private IPv4 addresses internally and providing translation to a public address only when needed. In IPv4 each device has a private IP address and a private port assigned in its LAN. If one has to connect to a laptop which is connected to the internet through a different LAN using IPv4, then it's hard to do so, as we can only get Public IP address and public port of that network but not the computer's Private IP and private port.

This issue we can overcome by using a IPv6 addressing because it provides a substantially larger IP address space than IPv4. IPv6(Internet Protocol version 6) is the next version after IPv4. In IPv6 addressing, we don't have Network Address Translation and thus, the concept of Private address doesn't exist. The laptop gets the same IP address as that of the phone with only the ports changed.

The IPv6 address of the PC can be found out by typing "What is my IP" on Google Chrome browser or simply the command "ipconfig" on the cmd.

An example of IPv6 (128-bit address) address is as shown below.
2001:4C48:100:162:8C40: CCB: 1FC0:1723

As a result of using IPv6, all devices connected to network will have a public, globally unique IPv6 address that can be sent over the Internet, which renders the Network Address Translation service used by IPv4 networks.

5 WORKFLOW

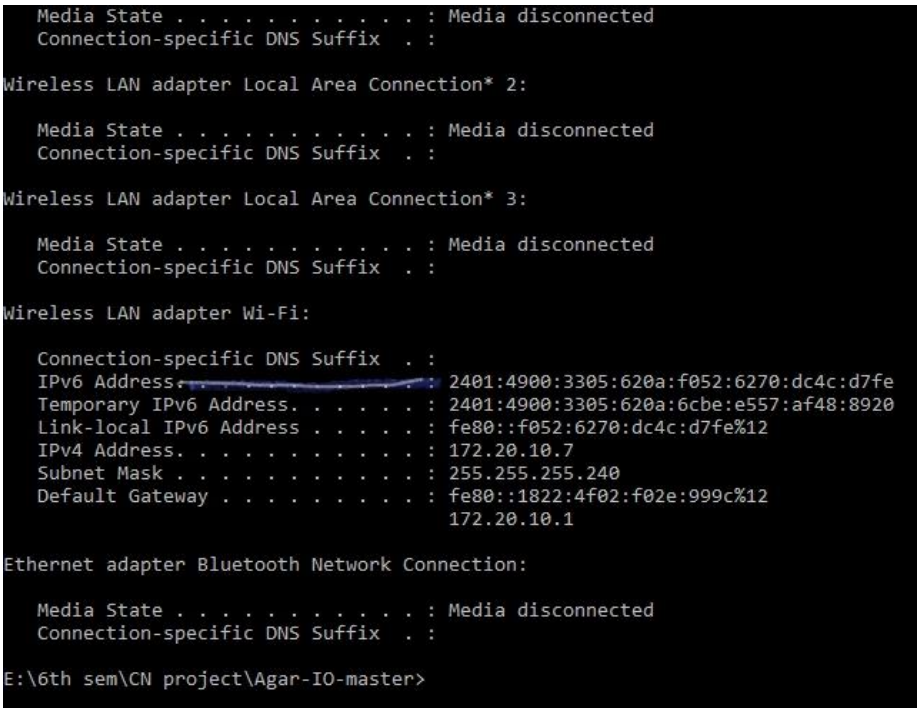
The following steps shall be performed by the client and server while running the game:

1) Open command prompt window (cmd) if it is windows or Open terminal in Ubuntu.

2) The server shall inform its Public IP to all the clients prior to the gaming session. This is done by typing ipconfig on the cmd

or

The game organizer runs the server file by the following command entered on his/her cmd.
>> python server.py



Before using client.py make the change in IP address accordingly as highlighted above in the picture.

3) Server uses the socket.socket(socket.AF_INET6, socket.SOCK_STREAM) function to create a socket. The first argument specifies the Internet address family for IPv6 and the second one indicates TCP sockets used to transport messages in the network layer.

4) The bind((host, port)) function associates the socket with a specified network interface and port number on the computer.

5) The listen() function enables the server to accept connections and makes it a “listening socket”.

6) Like the server, the client creates a socket using the socket.socket(socket.AF_INET6, socket.SOCK_STREAM) function and uses connect((host, port)) method to connect to the server.

7) At the same time several players run the client file by the typing the following command on his/her cmd. >>python game.py

```
C:\Users\Dell\Desktop\Agar-IO-master>python game.py
Please enter your name: player1
```

The game begins once a client connects on the same machine other servers is receiving ON

8) The accept() method in the server blocks and waits for an incoming connection. When a client connects, it returns a new socket object representing the connection and a tuple holding the address of the client. This socket is used to communicate with the client and is distinct from the listening socket that the server is using to accept new connections.

9) The two tasks of the server:a)sending commands to an already connected client b) listen and accept connections from other clients.

These two tasks are done from the same program at the same time by using the concept of threading

10) The client programme has been designed in such a way, if any of the player in the session presses escape key then the client programme is closed by client.close().

6 FLOWCHAT AND NETWORK DIAGRAM

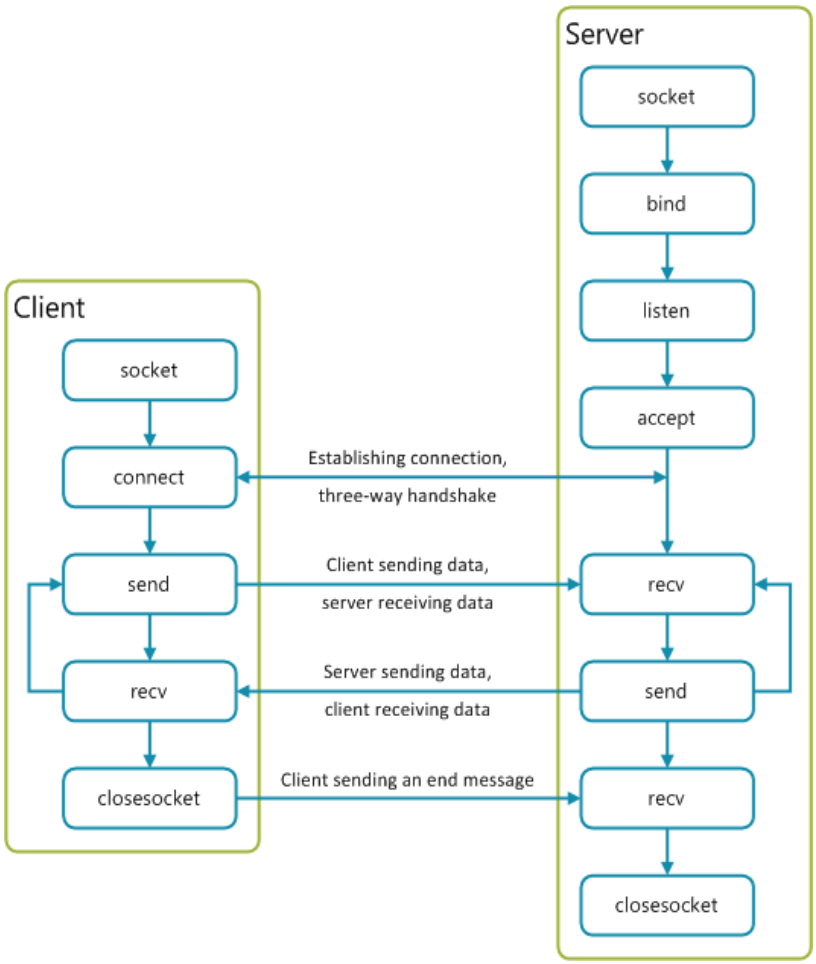


Fig.1. Server-client working flow-chart

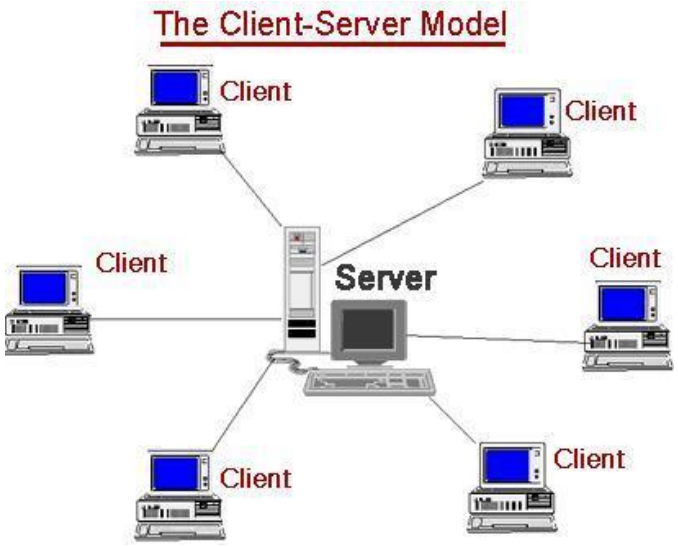


Fig.2. Network model

7 GAME MECHANICS

- The game will be in "lobby" mode until started, this means all each player can do is move.
- The game will begin once a client connects on the same machine the server is running on.

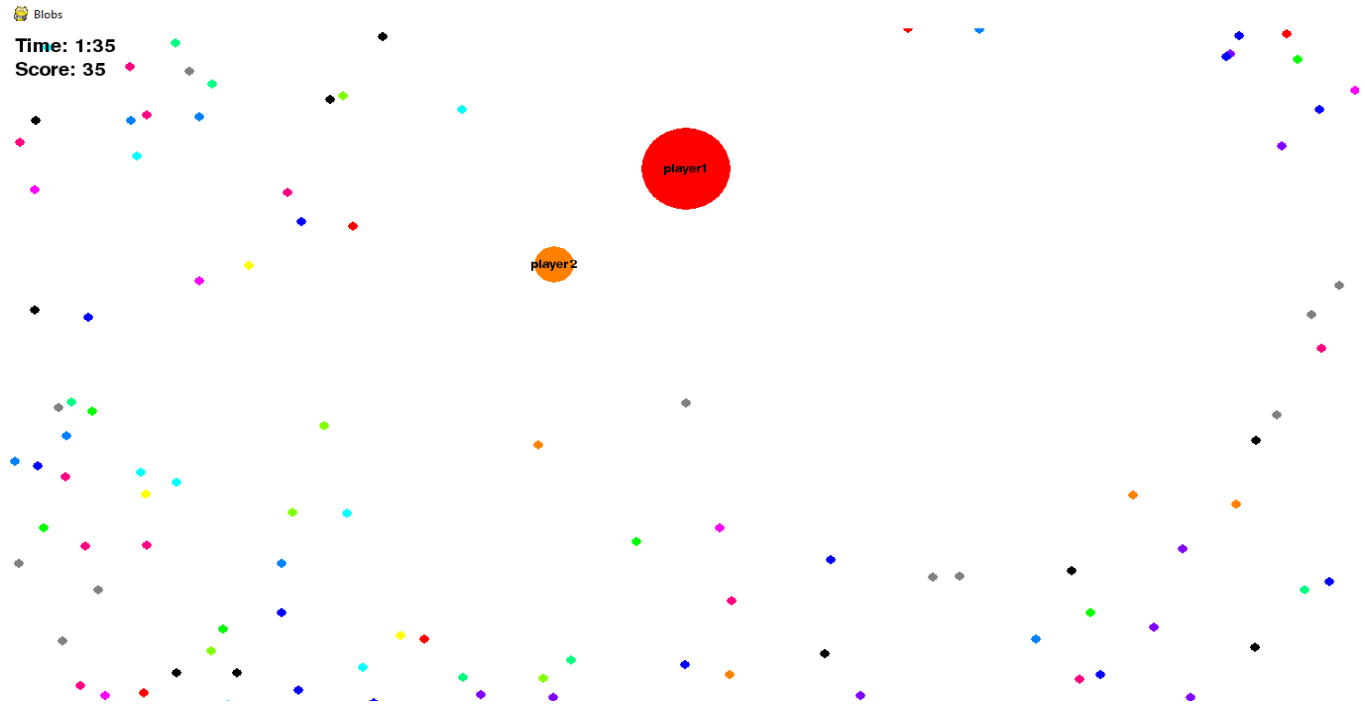
- Player will be denoted by a circle with his name attached.
- There are about (200 – 250) balls that has been scattered on the screen and the player's task is to eat as much balls as he can by overlapping on those balls.
- On consuming the balls the player gains score points and starts increasing his radius.
- As the player enlarges in size his velocity is decreased
- If player1 is of a larger size compared to player2 he can overlap on player2 and eat him.
- Each player will constantly lose 5% of his mass after every 7 seconds.
- As the balls are being constantly consumed by the players ,it get's refilled when the ball count drops below 150.

- Game lasts 5 minutes for each player.

8 RESULTS

8.1 OUTPUT SNAPSHOTS

The following screenshots were taken during the time of an actual trial gaming session. Here, for trail case player 1 and player 2 are the clients.




```
E:\6th sem\CN project\Agar-IO-master>python server.py
[SERVER] Server Started with local ip 2401:4900:3305:620a:f052:6270:dc4c:d7fe
[GAME] Setting up level
[SERVER] Waiting for connections
[CONNECTION] Connected to: ('2401:4900:3305:620a:f052:6270:dc4c:d7fe', 54943, 0, 0)
[STARTED] Game Started
[LOG] player1 connected to the server.
[GAME] player1's Mass depleting
[GAME] player1's Mass depleting
[CONNECTION] Connected to: ('2401:4900:3308:d3f1:3c4d:8126:8ca5:ac99', 65039, 0, 0)
[LOG] player2 connected to the server.
[GAME] player1's Mass depleting
[GAME] player2's Mass depleting
[GAME] player1's Mass depleting
[GAME] Generating more orbs
[GAME] player1's Mass depleting
[GAME] player1's Mass depleting
[GAME] player1's Mass depleting
[GAME] player1's Mass depleting
[GAME] player1's Mass depleting
[GAME] player1's Mass depleting
[GAME] player1's Mass depleting
[GAME] player1's Mass depleting
[DISCONNECT] Name: player1 , Client Id: 0 disconnected
[GAME] player2's Mass depleting
[DISCONNECT] Name: player2 , Client Id: 1 disconnected
```

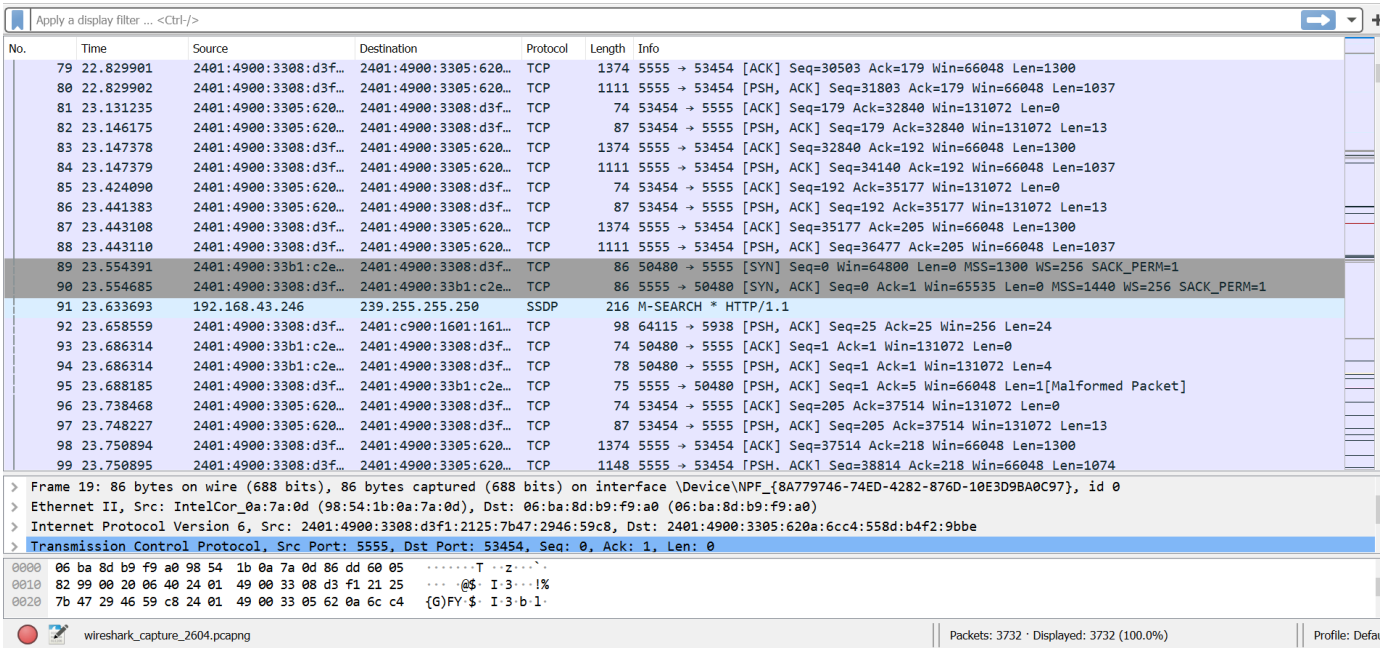
This appears on the cmd window. It indicates whenever the mass of the player depleting. And when game end client ID is disconnected

8.2 WIRESHARK CAPTURE

The wireshark capture below shows the initial TCP connections of 2 clients and a single server while the gaming session is live.

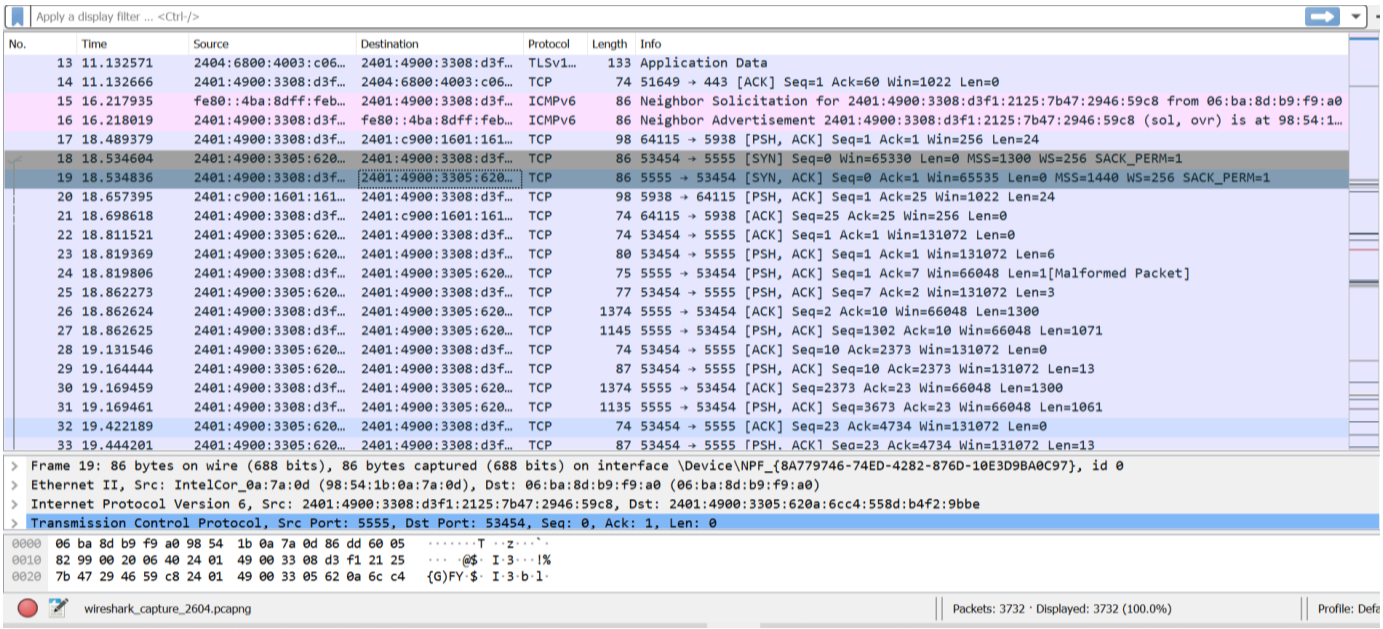
Client 1- 2401:4900:33b1:c2e…

Server – 2401:4900:3308:d3f…



Client 2 – 2401:4900:3305:620…

Server – 2401:4900:3308:d3f…



9 CONCLUSIONS

We have successfully designed an online-multiplayer gaming system using Socket programming and IPv6 addressing. By using this Python application, the main organizer of the game can player staying anywhere in the world with his friends or family by simply using his/her laptop and mobile hotspot. This makes it a much faster and a more efficient (cost and time effective) way of playing online with many players. This is completely for the entertainment purpose because gaming has become so important in today's life.

Sometimes, apps installed using app store can contain malware , which is a kind of software that can damage the smart phone or the device which we are using .So this kinds of malware can be avoided by playing online games like this. Additionally, all the unexpected events like sudden hanging up issues are handled.

10. CODE

10.1 client.py code

```
 import socket

import _pickle as pickle
class Network:
    """
    class to connect, send and recieve information from the server

    need to hardcode the host attirbute to be the server's ip
    """
    def __init__(self):
        self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #creating a socket
        #self.client.settimeout(10.0)
        self.host = "192.168.0.187" #give the IP address of server pc
        self.port = 5555  # assign a port
        self.addr = (self.host, self.port) # to establish TCP connection with the host


    def connect(self, name):
        """
        connects to server and returns the id of the client that connected
        :param name: str
        :return: int representing id
        """
        self.client.connect(self.addr)
        self.client.send(str.encode(name))
        val = self.client.recv(8) # client receives information from the server
        return int(val.decode()) # can be int because will be an int id


    def disconnect(self):
        """
        disconnects from the server
        :return: None
        """
        self.client.close()


    def send(self, data, pick=False):
        """
        sends information to the server

        :param data: str
        :param pick: boolean if should pickle or not
        :return: str
        """
        try:
            if pick:
                self.client.send(pickle.dumps(data))
            else:
                self.client.send(str.encode(data))
            reply = self.client.recv(2048*4)
            try:
                reply = pickle.loads(reply)
            except Exception as e:
                print(e)


            return reply
        except socket.error as e:
            print(e)
```

10.2 server.py code

```
"""
        main server script for running agar.io server

        can handle multiple/infinite connections on the same
        local network
        """
        import socket
        from _thread import *
        import _pickle as pickle
        import time
        import random
        import math


        # setup sockets
        S = socket.socket(socket.AF_INET, socket.SOCK_STREAM)   # creating a socket
        S.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)


        # Set constants
        PORT = 5555 # assign a port number

        BALL_RADIUS = 5
        START_RADIUS = 7

        ROUND_TIME = 60 * 5 # the game finishes in 5 minutes

        MASS_LOSS_TIME = 7 # mass depletion in seconds

        W, H = 1600, 830 # height and width of the coordinates

        HOST_NAME = socket.gethostname()
        SERVER_IP = socket.gethostbyname(HOST_NAME)

        # try to connect to server
        try:
            S.bind((SERVER_IP, PORT)) # assigns and IP address and a port number to a socket instance
        except socket.error as e:
            print(str(e))
            print("[SERVER] Server could not start")
            quit()

        S.listen()  # listen for connections
```

```
print(f"[SERVER] Server Started with local ip {SERVER_IP}")


# dynamic variables
players = {} # dictonary containing ip address of specific clients/players
balls = [] # list of balls scattered on the screen
connections = 0
_id = 0
colors = [(255,0,0), (255, 128, 0), (255,255,0), (128,255,0),(0,255,0),(0,255,128),(0,255,255),(0, 128, 255), (0,0,255), (0,0,255),
(128,0,255),(255,0,255), (255,0,128),(128,128,128), (0,0,0)]
start = False
stat_time = 0
game_time = "Starting Soon"
nxt = 1



# FUNCTIONS


def release_mass(players):
    """
    releases the mass of players

    :param players: dict
    :return: None
    """
    for player in players:
        p = players[player]
        if p["score"] > 8:
            p["score"] = math.floor(p["score"]*0.95)



def check_collision(players, balls):
    """
    checks if any of the player have collided with any of the balls

    :param players: a dictonary of players
    :param balls: a list of balls
    :return: None
    """
    to_delete = []
    for player in players:
        p = players[player]
        x = p["x"]
        y = p["y"]
        for ball in balls:
            bx = ball[0]
            by = ball[1]
```

```
                dis = math.sqrt((x - bx)**2 + (y-by)**2)
                if dis <= START_RADIUS + p["score"]:
                        p["score"] = p["score"] + 0.5  # for each collision with ball 0.5
                        balls.remove(ball)                  score increase



def player_collision(players):
        """
        checks for player collision and handles that collision

        :param players: dict
        :return: None
        """
        sort_players = sorted(players, key=lambda x: players[x]["score"])
        for x, player1 in enumerate(sort_players):
                for player2 in sort_players[x+1:]:
                        p1x = players[player1]["x"]
                        p1y = players[player1]["y"]


                        p2x = players[player2]["x"]
                        p2y = players[player2]["y"]


                        dis = math.sqrt((p1x - p2x)**2 + (p1y-p2y)**2)
                        if dis < players[player2]["score"] - players[player1]["score"]*0.85:
                                players[player2]["score"] = math.sqrt(players[player2]["score"]**2
+ players[player1]["score"]**2) # adding areas instead of radii
                                players[player1]["score"] = 0
                                players[player1]["x"], players[player1]["y"] =   get_start_location(players)
                                print(f"[GAME] " + players[player2]["name"] + " ATE " + players[player1]["name"])



def create_balls(balls, n):
        """
        creates orbs/balls on the screen

        :param balls: a list to add balls/orbs to
        :param n: the amount of balls to make
        :return: None
        """
        for i in range(n):
                while True:
                        stop = True
                        x = random.randrange(0,W) #random width of the ball is chosen
                        y = random.randrange(0,H)
                        for player in players:
                                p = players[player]
```

```
                dis = math.sqrt((x - p["x"])**2 + (y-p["y"])**2)
                if dis <= START_RADIUS + p["score"]: # some initial distance is
                    stop = False                    maintained between
            if stop:                                and ball
                break


        balls.append((x,y, random.choice(colors))) #random colours are appended for ball



def get_start_location(players):
    """
    picks a start location for a player based on other player
    locations. It will ensure it does not spawn inside another player

    :param players: dict
    :return: tuple (x,y)
    """
    while True:
        stop = True
        x = random.randrange(0,W)
        y = random.randrange(0,H)
        for player in players:
            p = players[player]
            dis = math.sqrt((x - p["x"])**2 + (y-p["y"])**2)
            if dis <= START_RADIUS + p["score"]:
                stop = False
                break
        if stop:
            break
    return (x,y)



def threaded_client(conn, _id):
    """
    runs in a new thread for each player connected to the server

    :param con: ip address of connection
    :param _id: int
    :return: None
    """
    global connections, players, balls, game_time, nxt, start


    current_id = _id # returns Id of the clients


    # recieve a name from the client
    data = conn.recv(16) # receiving client data by establishing a connection
```

```
name = data.decode("utf-8") # conversion made for decoding
print("[LOG]", name, "connected to the server.")


# Setup properties for each new player
color = colors[current_id]
x, y = get_start_location(players)
players[current_id] = {"x":x, "y":y,"color":color,"score":0,"name":name}  # x, y color, score, name


# pickle data and send initial info to clients
conn.send(str.encode(str(current_id)))


# server will recieve basic commands from client
# it will send back all of the other clients info
'''
commands start with:
move
jump
get
id - returns id of client
'''
while True:


        if start:
                game_time = round(time.time()-start_time)
                # if the game time passes the round time the game will stop
                if game_time >= ROUND_TIME:
                        start = False
                else:
                        if game_time // MASS_LOSS_TIME == nxt:
                                nxt += 1
                                release_mass(players)
                                print(f"[GAME] {name}'s Mass depleting")
        try:
                # Recieve data from client
                data = conn.recv(32)


                if not data:
                        break


                data = data.decode("utf-8")
                #print("[DATA] Received", data, "from client id:", current_id)


                # look for specific commands from received data
                if data.split(" ")[0] == "move": # it splits the string into a list
                        split_data = data.split(" ")
                        x = int(split_data[1])
```

```
                        y = int(split_data[2])
                        players[current_id]["x"] = x
                        players[current_id]["y"] = y

                        # only check for collison if the game has started
                        if start:
                                check_collision(players, balls)
                                player_collision(players)

                        # if the amount of balls is less than 150 create more
                        if len(balls) < 150:
                                create_balls(balls, random.randrange(100,150))
                                print("[GAME] Generating more orbs")

                        send_data = pickle.dumps((balls,players, game_time))

                elif data.split(" ")[0] == "id":
                        send_data = str.encode(str(current_id))  # if user requests id
                                then send it
                elif data.split(" ")[0] == "jump":
                        send_data = pickle.dumps((balls,players, game_time))
                else:
                        # any other command just send back list of players
                        send_data = pickle.dumps((balls,players, game_time))

                # send data back to clients
                conn.send(send_data)

        except Exception as e:
                print(e)
                break  # if an exception has been reached disconnect client

        time.sleep(0.001)

# When user disconnects
print("[DISCONNECT] Name:", name, ", Client Id:", current_id, "disconnected")

connections -= 1
del players[current_id]  # remove client information from players list
conn.close()  # close connection


# MAINLOOP
```

```
# setup level with balls
create_balls(balls, random.randrange(200,250))


print("[GAME] Setting up level")
print("[SERVER] Waiting for connections")


# Keep looping to accept new connections
while True:

        host, addr = S.accept()
        print("[CONNECTION] Connected to:", addr)


        # start game when a client on the server computer connects
        if addr[0] == SERVER_IP and not(start):
                start = True
                start_time = time.time()
                print("[STARTED] Game Started")


        # increment connections start new thread then increment ids
        connections += 1
        start_new_thread(threaded_client,(host,_id))
        _id += 1


# when program ends
print("[SERVER] Server offline")
```

10.3 game.py code

```
# small
networ
k game
that has
differnt
blobs
        # moving around the screen


        import contextlib   # used to define a factory function
        with contextlib.redirect_stdout(None):
            import pygame
        from client import Network
        import random
        import os
        pygame.font.init() # required to initialize the attribute of the class


        # Constants
        PLAYER_RADIUS = 10
        START_VEL = 9
        BALL_RADIUS = 5


        W, H = 1600, 830


        NAME_FONT = pygame.font.SysFont("comicsans", 20)
        TIME_FONT = pygame.font.SysFont("comicsans", 30)
        SCORE_FONT = pygame.font.SysFont("comicsans", 26)


        COLORS = [(255,0,0), (255, 128, 0), (255,255,0), (128,255,0),(0,255,0),(0,255,128),(0,255,255),(0, 128, 255), (0,0,255),
        (0,0,255), (128,0,255),(255,0,255), (255,0,128),(128,128,128), (0,0,0)]


        # Dynamic Variables
        players = {}
        balls = []


        # FUNCTIONS
        def convert_time(t):
                """
                converts a time given in seconds to a time in
                minutes

                :param t: int
                :return: string
                """
                if type(t) == str:
                        return t


                if int(t) < 60:
                        return str(t) + "s"
                else:
                        minutes = str(t // 60)
                        seconds = str(t % 60)


                        if int(seconds) < 10:
                                seconds = "0" + seconds
```

```
                return minutes + ":" + seconds



def redraw_window(players, balls, game_time, score):
        """
        draws each frame
        :return: None
        """
        WIN.fill((255,255,255)) # fill screen white, to clear old frames

                # draw all the orbs/balls
        for ball in balls:
                pygame.draw.circle(WIN, ball[2], (ball[0], ball[1]), BALL_RADIUS)


        # draw each player in the list
        for player in sorted(players, key=lambda x: players[x]["score"]):
                p = players[player]
                pygame.draw.circle(WIN, p["color"], (p["x"], p["y"]), PLAYER_RADIUS + round(p["score"]))
                # render and draw name for each player
                text = NAME_FONT.render(p["name"], 1, (0,0,0))
                WIN.blit(text, (p["x"] - text.get_width()/2, p["y"] - text.get_height()/2))


        # draw scoreboard
        sort_players = list(reversed(sorted(players, key=lambda x: players[x]["score"])))
        title = TIME_FONT.render("Scoreboard", 1, (0,0,0))
        start_y = 25
        x = W - title.get_width() - 10
        WIN.blit(title, (x, 5))


        ran = min(len(players), 3)
        for count, i in enumerate(sort_players[:ran]):
                text = SCORE_FONT.render(str(count+1) + ". " + str(players[i]["name"]), 1, (0,0,0))
                WIN.blit(text, (x, start_y + count * 20))


        # draw time
        text = TIME_FONT.render("Time: " + convert_time(game_time), 1, (0,0,0))
        WIN.blit(text,(10,10))
        # draw score
        text = TIME_FONT.render("Score: " + str(round(score)),1,(0,0,0))
        WIN.blit(text,(10,15 + text.get_height()))



def main(name):
        """
        function for running the game,
        includes the main loop of the game

        :param players: a list of dicts represting a player
        :return: None
        """
        global players


        # start by connecting to the network
        server = Network()
        current_id = server.connect(name)
        balls, players, game_time = server.send("get")


        # setup the clock, limit to 30fps
```

```
clock = pygame.time.Clock()


run = True
while run:
	clock.tick(30) # 30 fps max
	player = players[current_id]
	vel = START_VEL - round(player["score"]/14)
	if vel <= 1:
		vel = 1


	# get key presses
	keys = pygame.key.get_pressed()


	data = ""
	# movement based on key presses
	if keys[pygame.K_LEFT] or keys[pygame.K_a]:
		if player["x"] - vel - PLAYER_RADIUS - player["score"] >= 0:
			player["x"] = player["x"] - vel


	if keys[pygame.K_RIGHT] or keys[pygame.K_d]:
		if player["x"] + vel + PLAYER_RADIUS + player["score"] <= W:
			player["x"] = player["x"] + vel


	if keys[pygame.K_UP] or keys[pygame.K_w]:
		if player["y"] - vel - PLAYER_RADIUS - player["score"] >= 0:
			player["y"] = player["y"] - vel


	if keys[pygame.K_DOWN] or keys[pygame.K_s]:
		if player["y"] + vel + PLAYER_RADIUS + player["score"] <= H:
			player["y"] = player["y"] + vel


	data = "move " + str(player["x"]) + " " + str(player["y"])


	# send data to server and recieve back all players information
	balls, players, game_time = server.send(data)


	for event in pygame.event.get():
		# if user hits red x button close window
		if event.type == pygame.QUIT:
			run = False


		if event.type == pygame.KEYDOWN:
			# if user hits a escape key close program
			if event.key == pygame.K_ESCAPE:
				run = False




	# redraw window then update the frame
	redraw_window(players, balls, game_time, player["score"])
	pygame.display.update()



server.disconnect()
pygame.quit()
```

```
        quit()



# get users name
while True:
        name = input("Please enter your name: ")
        if  0 < len(name) < 20:
                break
        else:
                print("Error, this name is not allowed (must be between 1 and 19 characters [inclusive])")


# make window start in top left hand corner
os.environ['SDL_VIDEO_WINDOW_POS'] = "%d,%d" % (0,30)


# setup pygame window
WIN = pygame.display.set_mode((W,H))
pygame.display.set_caption("Blobs")


# start game
main(name)
```