CS434 - INTERNETWORKING PROTOCOL

SOCKET PROGRAMMING REPORT

Team members:

Nguyễn Hữu Khang 18125086

Đặng Phương Nam 18125141

Lâm Đức Huy 18125131

A. Team Information and contribution:

Our last student ID digits are 1,1 and 6. So our subject will be Game 02: **Racing Arena**

No.	Name	Student ID	Contribution (%)
1	Nguyễn Hữu Khang	18125086	33.3%
2	Đặng Phương Nam	18125141	33.3%
3	Lâm Đức Huy	18125131	33.3%

B. Self-evaluation:

We use Python to build our socket and Pygame to build our UI. All the connections are handled by threads instead of Select.

We completed our project terminal version in C++ but completely stuck with MFC. So we decided to build everything again in Python.

We have fulfilled all of the requirements. However, there are some small bugs that we didn't have enough time to fix.

No.	Requirements	Score	Evaluate

1	Use C/C++, Java, C#, Python, Rust	2	2
2	Implement whole gameplay properly	3	2.5
3	Socket Non-blocking	2	2
4	Have a good GUI (MFC, WPF, Swing, etc.)	3	3
	Total	10	9.5

C. REPORT DETAIL:

I. Gameplay story:

- First, we choose the number of players and race length at the server.
- We initialize the socket at the server and wait for the player to connect.
- For each player connected, the server will create a separate thread to handle all messages from this player.
- When there are enough players, the server stops waiting for new connections and start gameplay.
- For each turn, the server generates a question and broadcasts it to all available players. The server will wait for 20 seconds then evaluate the player answers and send the results.
- The game will end when there is no player left or a player's point reach the race length.
- The server will start a new game after that.

II. Important methods:

- SendToOne(receiver, message): send a message to a receiver
- Broadcast(message): send a message to all known socket
- ReceiveMessage(sender): wait and receive a message from the sender

III. Client-Server communication:

1. Server socket initialize:

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setblocking(False)
server.settimeout(1000)

IP_address = 'localhost'
Port = 9099
server.bind((IP_address, Port))
server.listen(100)
```

- Init socket to AF_INET and SOCK_STREAM
- Set socket to non-blocking
- Bind socket to port and start listening

```
while True:
    if numConnections < number_of_players:
        clientSocket, address = server.accept()
        if number_joined == number_of_players:
            sendToOne(clientSocket, "Maximum number of players joined!")
            clientSocket.close()
        numConnections += 1
        clientList.append(clientSocket)
        print(address[0] + " connected")
        start_new_thread(startClientThread, (clientSocket, address,))
    elif startGame:
        break

gamePlay()
```

- While there are not enough players in the room, the server will accept new connections and start a new thread to handle that player.

- When enough players come, the server stops listening to the new connection and starts the game.

2. Handle client by using thread:

For every client accepted, the server creates a new thread. This thread will be used for handling player nicknames and receiving the player answers.

To maintain the game state, we use multiple global variables:

+ clientList[] : store the client socket
+ playerList{}: store the client name
+ answerList{}: store answers from client
+ scoreList{}: store scores of client
+ num_health{}: store how many health each player has left
+ answerOrder[]: store the answer order to get the fastest right player.

The function to create a thread is **startClientThread(clientSocket, address)**

3. Handle server by using thread:

```
listen_thread = start_new_thread(listening_to_server, ())

while True:
    if game_state == GameState.LOGIN:
        handle_login_state()
        render_login_scene()
    elif game_state == GameState.STARTING:
        render_waiting_scene()
        handle_starting_state()
    elif game_state == GameState.STARTED:
        render_game_scene()
        handle_started_state()
    elif game_state == GameState.OVER:
        render_gameover_scene()
        handle_starting_state()
    elif game_state == GameState.WIN:
        render_victory_scene()
        handle_starting_state()
    pg.display.update()
```

The client is managed by using GameState. Each GameState uses a different function to render GUI.

Because the main thread is occupied by the GUI, we have to use a new thread to receive messages from the server

There are several global variables used like: game_state, question, answer, result, life, rank,...

The function to create a new thread at the client-side is **listening_to_server()**

4. **Handle client nickname:**
   + Nickname format will be checked on the client-side. If it is valid, the client will send that to the server.

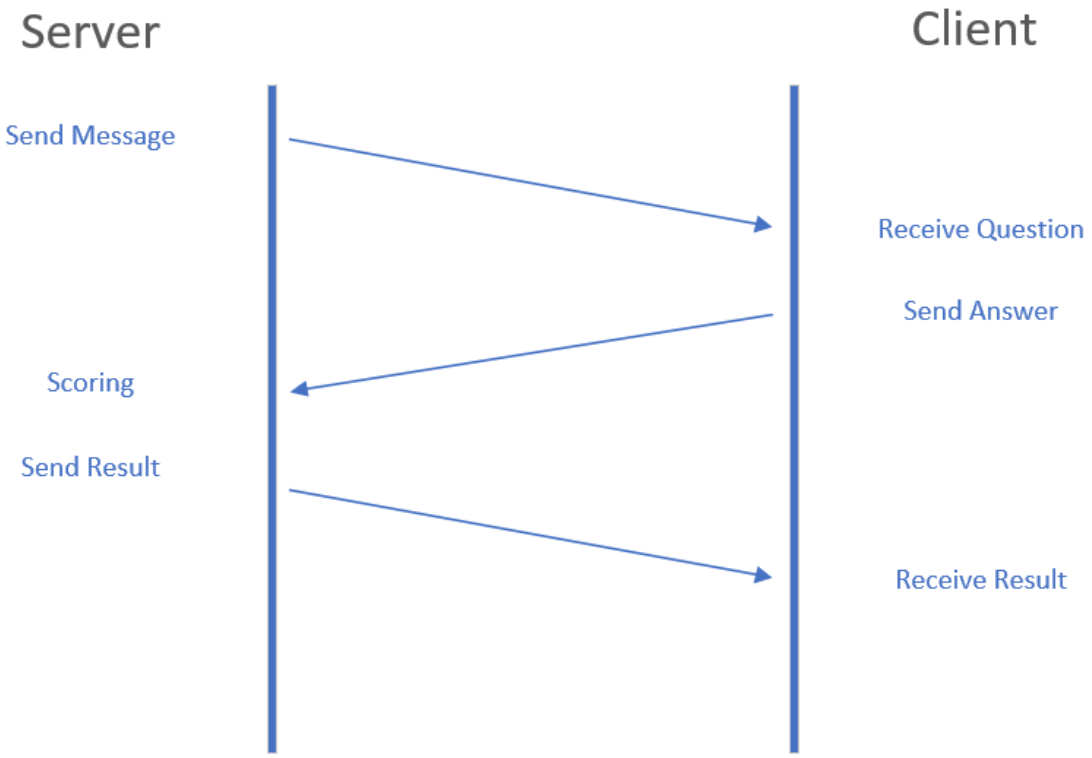
```
name = clientSocket.recv(2048).decode('utf-8')
if name:
    if name in playerList.values():
        sendToOne(clientList, clientSocket, "Name already taken")
    else:
        playerList[clientSocket] = name
        scoreList[name] = 0
        num_health[name] = 3
        number_joined += 1
        print("Player connected: " + str(address) + " [ " + playerList[clientSocket] + " ]")
        if number_joined < number_of_players:
            sendToOne(clientList, clientSocket,
                      "Welcome to the quiz " + name + "!\nPlease wait for other participants to join...")
            break
        elif number_joined == number_of_players:
            startGame = True
            break
```

+ The server will receive the nickname and check if it's already existed, then respond to the client whether to choose another name or allow the client to enter the lobby.

```
message = receive_message(client)
print(message)
if not message:
    print("DISCONNECTED")
    sys.exit()
else:
    if message == "Name already taken":
        notice = "Name already taken. Please choose another one!!!"
        print("Name already taken. Please choose another one!!!")
        nickname = ""
    elif "start game" in message:
        print("game_state: starting", )
        continue
```

+ The response from the server is quite easy so we don't use any flag here. We just check the entire message on the client-side. There is a bug in Python that 2 consecutive messages sent can be concatenated as 1 message so we sometimes have to use "in" instead of "==".
+ This will loop several times until the client can enter a valid nickname.

5. **Handle question - answer:**



5.1 Flags:

The server uses several flags to help the client distinguish the message types:

+ `RL| : Race length`
+ `Q| : the question`
+ `A| : the answer`
+ `S| : the score`
+ `Win : a player win the game`
+ `Gameover: a player lose the game`
+ `MS| and |L|: a player answer wrong and lose 1 health`
+ `Top: a player is currently leading`
+ `P| x |B| y: a player is currently at position x and behind player y`

5.2 Server:

```
broadcast(clientList, "\nParticipant(s) joined:")
for i in playerList:
    broadcast(clientList, ">> " + playerList[i] + "\n")
broadcast(clientList, "\nThe quiz will begin in 20 seconds. Quickly go through the instructions\n")
broadcast(clientList, f"RL|{race_length}")
print("\n" + str(
    number_of_players) + " participant(s) connected! The quiz will begin in 20 seconds\n\nRace length is " +
     str(race_length) + "\n")
time.sleep(10)
broadcast(clientList, "start game")
time.sleep(0.5)
```

- First, the server broadcast some information about the game for players

```
broadcast(clientList, "Q|" + question)
time.sleep(20)
broadcast(clientList, "A|" + str(answer))
time.sleep(2)
```

- After that, it will send the question and wait for 20s then send the answer in the main thread.

```
while True:
    try:
        message = clientSocket.recv(2048).decode('utf-8')
        if message:
            print(playerList[clientSocket] + " answered:" + message)
            answerList[clientSocket] = message
            answerOrder.append(clientSocket)
    except:
        print("Player disconnected")
        break
```

- It will wait for the answers in other threads.
- After 20s, it will check the result, update the game state and send back the result to players (including score, winner, game over, ranking,...).
- When the game end, the server will close all connection and start waiting for clients again.

5.3 Clients:

```
elif len(message) > 2:
    if "Q|" in message:
        answer = ''
        result = ''
        rank = ''
        question = str(message.split("Q|")[-1])
        game_state = GameState.STARTED
    elif "RL|" in message:
        race_length = int(message.split("RL|")[-1])
        game_state = GameState.STARTING
    elif "A|" in message:
        result = message.split("A|")[-1]
    elif "MS|" in message:
        temp = message.split("MS|")[-1].split("|L|")
        score = temp[0]
        life = temp[1]
    elif "S|" in message:
        score = int(message.split("S|")[-1])
    elif "Top" in message:
        rank = 'You are NO.1. No one is better than you!'
    elif "P|" in message:
        temp = message.split("P|")[-1].split("|B|")
        rank = f'You are NO.{temp[0]}. Behind {temp[1]}'
    elif "Win" in message:
        game_state = GameState.WIN
    elif "Gameover" in message:
        game_state = GameState.OVER
```

- The client will listen to the server and update the state using the thread. Each message is recognized by a flag.

```
def handle_started_state():
    global answer
    for event in pg.event.get():
        if event.type == pg.QUIT:
            pg.quit()
            sys.exit()
        elif event.type == pg.KEYDOWN:
            if event.key == pg.K_BACKSPACE:
                answer = answer[:-1]
            elif event.key == pg.K_SPACE:
                if len(answer) == 0:
                    answer = "None"
                print(answer)
                try:
                    client.send(bytes(answer, 'utf-8'))
                except:
                    client.close()
            else:
                answer = update_answer(answer, event)
```

- During questioning, a special input function is used to manage time for the input. After the player submitted the answer, the client will send that answer to the server. If timeout and player didn't enter anything, the client will send None.

IV. Language, tools and library:

- Language: Python, Pycharm, VSCode
- Library: Pygame 2.1.0

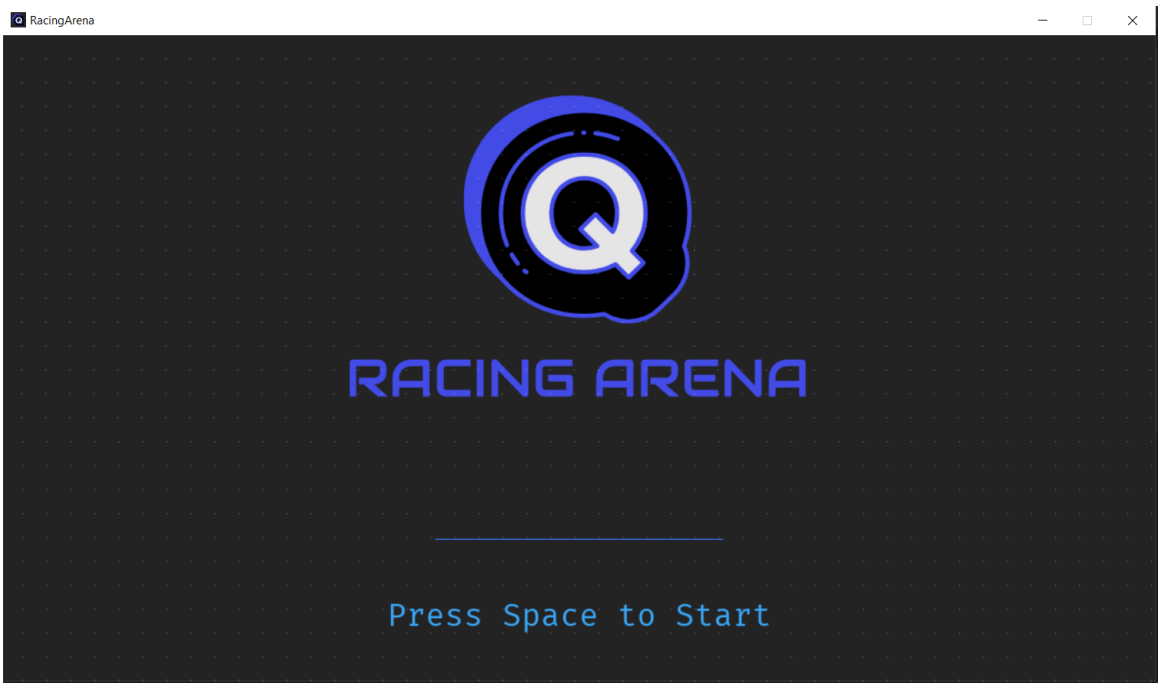
V. User Guide:

1. Server:

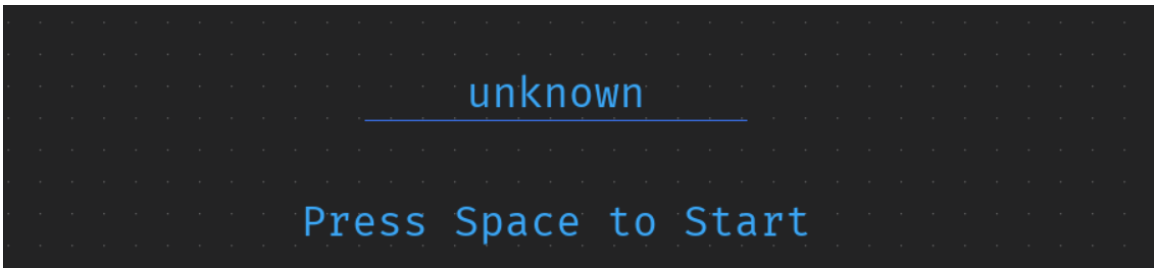
- Run Server.exe

2. Client:

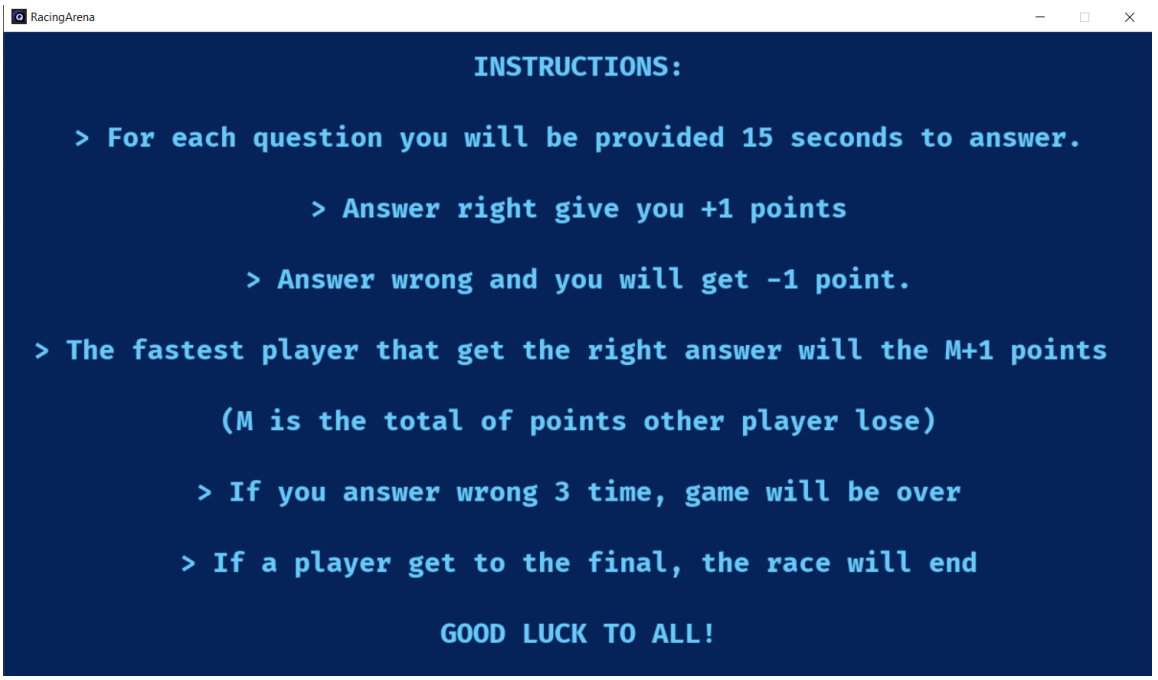
- Run Client.exe



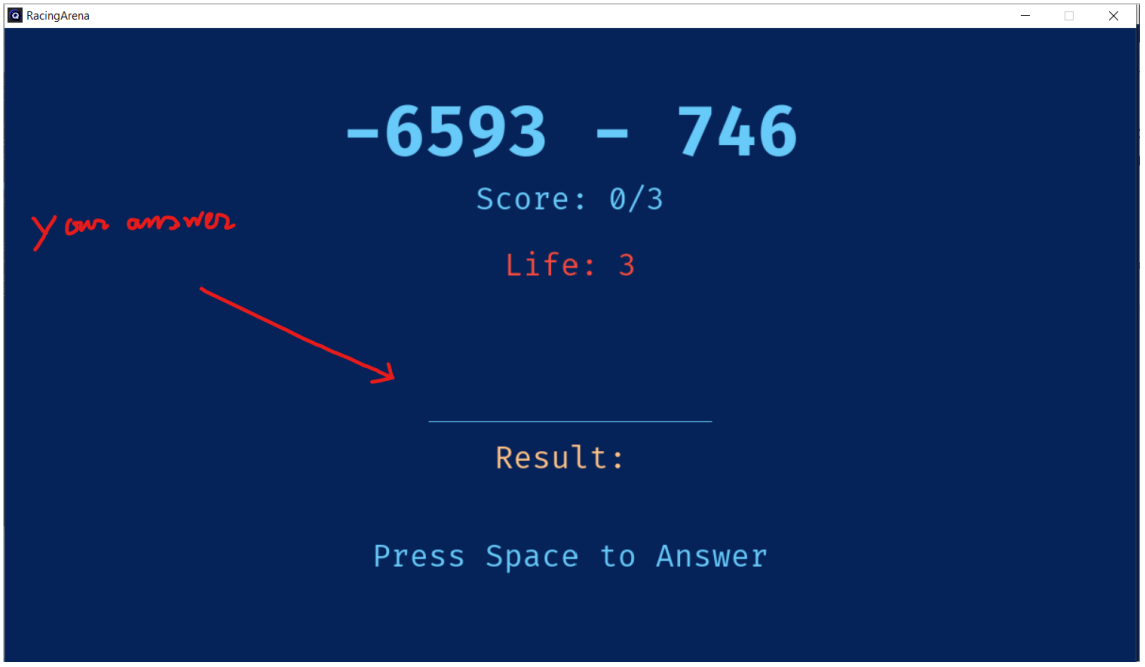
- Enter Your name



- Read the instruction then Play



- Read the instruction then Play



(Life , Ranking, and result will be display on screen)

REFERENCES:

[1] https://viblo.asia/p/build-1-chat-tool-voi-python-phan-1-socket-Qbq5QyGJ5D8

[2] https://github.com/DaKeiser/SocketQuiz

[3] https://github.com/MyreMylar/pygame_gui

[4] https://docs.python.org/3/howto/sockets.html

[5] https://www.studytonight.com/network-programming-in-python/blocking-and-nonblocking-socket-io