Socket Bots

Individual Portfolio Assignment 1

Siem Fitwi Fissuh
s351913
DATA2410

Table of Contents

Introduction

This project is a portfolio assignment in Data network and cloud computing topic at OsloMet university. The goal in this project is to create a communication channel with bots. The programing language used in this project is **python**. Based on what is mentioned in the assignment, I have solved the task by having the clients signed as a bots. Before “opening” the chat room, the server waits for four clients to be connected on the same server. When a client is connected it get identified with random bot. Then, based on the bot`s interest they will respond to suggested action to the chatroom.

GitHub repository: https://github.com/siemff/data2410_potrfolio.git

Bot explanation

We got four insane bots in this project. Madelene, the sporty type. Felix, the nurd type. Marija, the lazy type, and Rateb, the likable type. They are the bots that will identify the clients and respond to the suggested message.

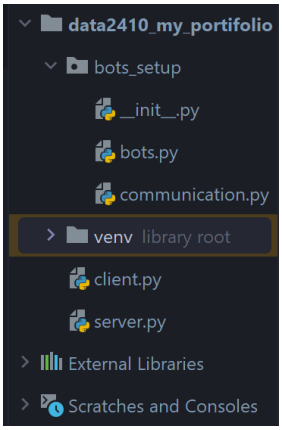
Client explanation

Clients connect to the same server. IPv4 address: 127.10.10.10. Port: 8080. When all four clients are connected the chat room opens. They will receive suggestion from the host. Once all clients/bots have responded to the suggested action the bots leave the chat room and the connection ends.

Server explanation

The server acts as chat room. Accept four connections. Maintain a list of connected clients. Initiate a round of dialogue by suggesting an action. Send suggestion and response to each of the connected clients and disconnect once all clients have responded the suggestion. All those actions can be seen at the server.

Code readability



The code contains four python files. Communication.py, bots.py, server.py and client.py. To solve the task best possible I have used the following important interfaces:

- ***Socket***: to enable two networks/sockets to send and receive data, bi-directionally, at any given moment. It works by connecting two nodes/sockets together and allowing them to communicate in real time.
- ***Serialization/deserializing***: for making/receiving requests/responses - effectively transporting "messages". Converting data which include arrays into a string so it can be stored as well as transmitted easily. The pickle module is used here for implementing a binary protocol for serializing and de-serializing a python object structure. I have used all four methods the pickle interface provides.

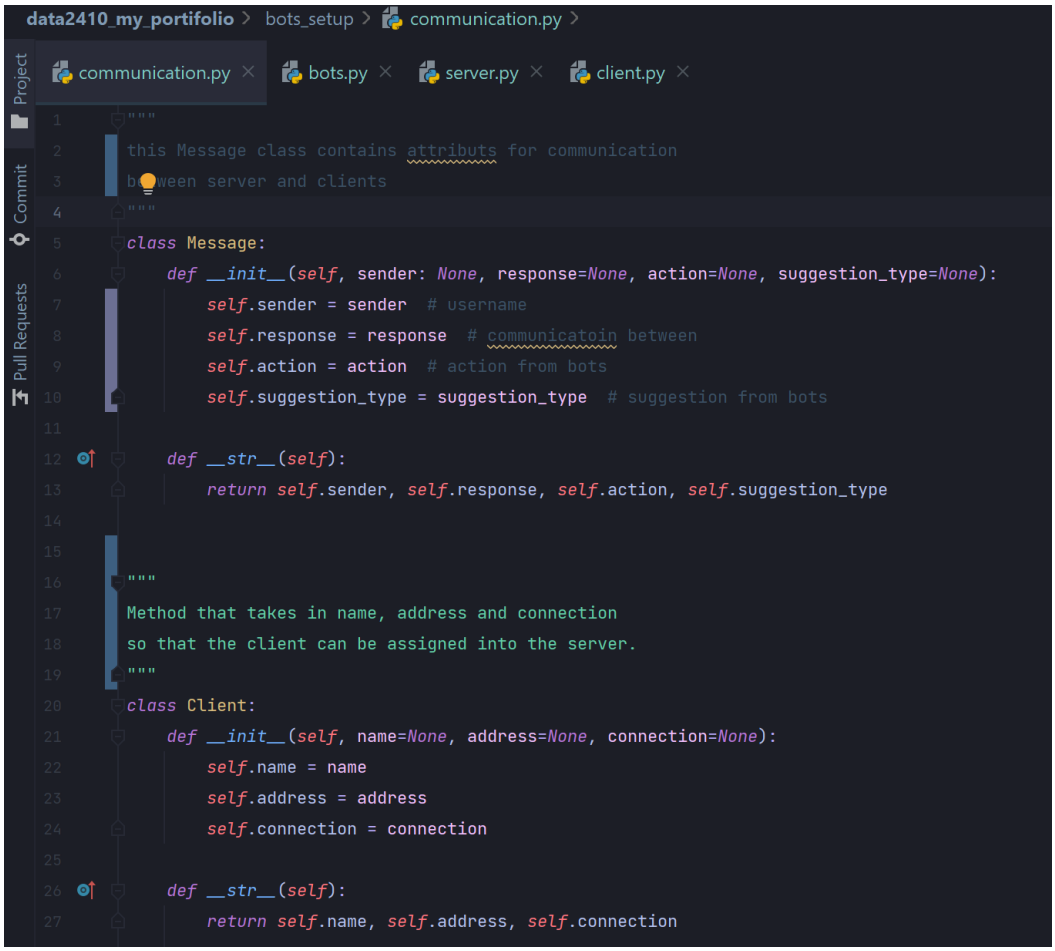
```
Dump(object, file): serialize and store a python object into a file
Dumps(object) -> string: serialize a python object into a byte stream/string
Load(file) -> object: deserializes a python object to its original form.
Loads(string) -> object: deserialize a byte stream into a python object.
```

- ***Multithreading***: to allow threads to communicate and share resources such as message/data, to the same server. So that server and all clients receive alle communications between.

Run the code

```
Run from terminal:
1. Open terminal (CMD) on the project file.
2. Run server: py -3.9 server.py
3. Run 4 separate clients: py -3.9 client.py
Run on the python file.
#!/usr/bin/env python
At top of the server and client file click the run icon.
```

Communication.py explanation

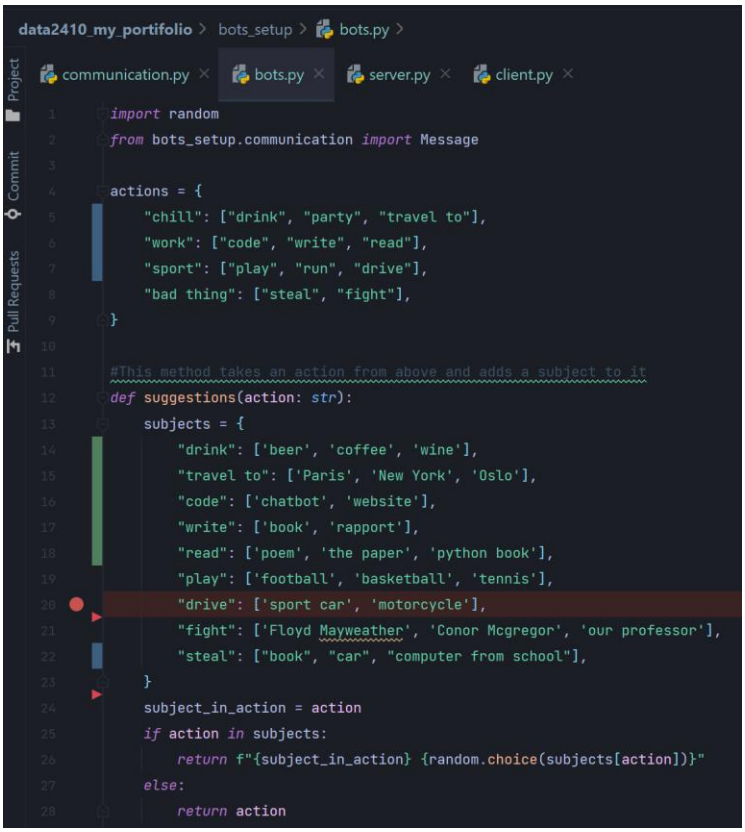


Bots.py explanation

This file contains all response actions from host and bots, as well as actions and suggestion which is important part of the task.

The suggestion message takes in action from the user/host and add a subject to it.

By using *random.choice()* If for example the "host" is interested in sport (action) and would like to play (suggestion) football (subject), by calling Message class it returns sender name and response.




```python
def my_bot():
    name = 'Host'                                          # username
    suggestion_types = ["sport", "chill", "work", "bad thing"]    # interests
    suggestion = random.choice(suggestion_types)           # random choise of interests
    action = random.choice(actions[suggestion])            # random choise action of chosen suggesion
    print("Host suggested " + suggestion)                  # print Host suggestion
    subject = suggestions(action)                          # action choices
    responses = [                                          # message from host
        f"Do you guys want to {subject}?",
        f"Let's {subject} guys!!",
    ]
    raondom_response =random.choice(responses)             #Random choise of messages
    return Message(sender=name, response=raondom_response,     # Return usenname, message and
                   action=action, suggestion_type=suggestion)    #chosen action and suggestion

def madelen(response: Message):
    name = 'Madelen'                                       # username
    suggestion = "sport"                                   # interests
    change_suggestion = random.choice(actions[suggestion])     # random choise action of chosen suggesion
    responses = [                                          # response from Madelen
        f"I dont like it, but I will join",
        f"I have some other stuff to do.",
        f"That one isn't fair, I would like to {change_suggestion}",
    ]
    random_response = random.choice(responses)             #Random choise of messages
    return Message(sender=name, response=random_response)      # Return usenname and response
```

We can see here the method that returns suggestion of the host and the response from the bots by calling the class Message.

Server.py explanation

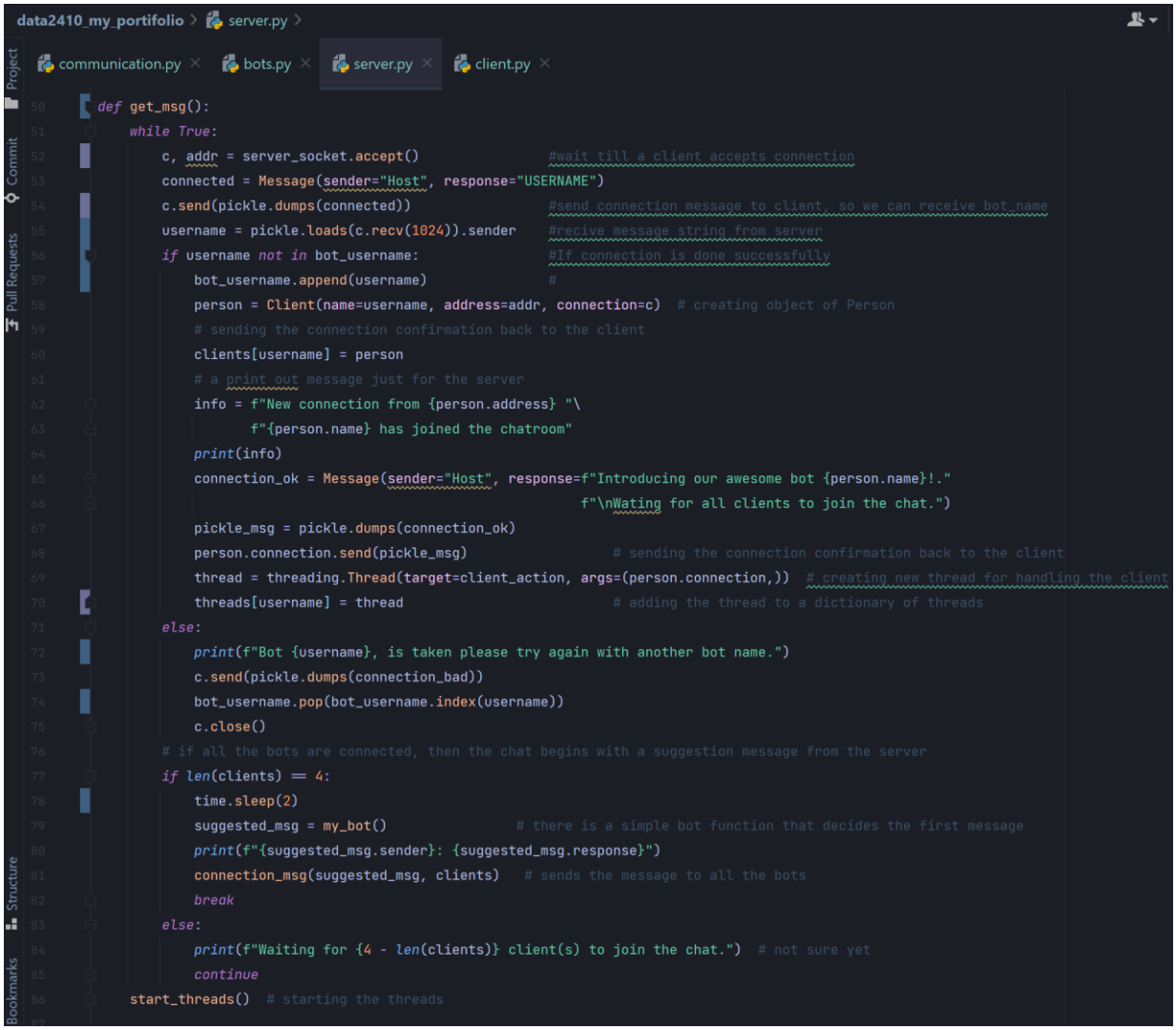
In client_action method the method takes in connection from the client and cheks if the client is still in the chatroom. If clients leaves (“bye”), the chatroom it will be informed in the server and the connection between ends. The connection ends automatically when alle clients have responded.

```python
def start_threads():                        #the client handler_threds will begin after connection loop
    for count in threads:
        threads[count].start()               # Make the new thread ready.
        time.sleep(2.5)                     # puts a thread to sleep for 2.0 seconds

def connection_msg(msg, clients):
    pickle_msg = pickle.dumps(msg)
    for client in clients:
        clients[client].connection.send(pickle_msg)

#This method checks if the client is still connected
#as long as it's connected it prints out responses of the client in server.
def client_action(connection):
    while True:
        msg = connection.recv(1024)             #Recive message from server
        unpickle_msg = pickle.loads(msg)        #deserialize a byte stream into a python object
        if unpickle_msg.response == "Bye":
            client = clients.pop(unpickle_msg.sender)
            print(client.name, "left the chatroom.")
            time.sleep(5)                       #after 10s the server clothes the chatroom
            client.connection.close()
            threads.pop(unpickle_msg.sender)
            break
        else:
            connection_msg(unpickle_msg, clients)
            print(f"{unpickle_msg.sender}: {unpickle_msg.response}")
```

In get_msg() function the server connects with client first by accepting the connection. Then I used pickle and unpickle as I mentioned over for transporting “messages” effectively and to implement binary protocol for pickling/Serializing and unpickling/deserializing an object. The method introduces clients to the chatroom check if all clients are connected and prints all messages at the server once all 4 clients are connected.



```python
def get_msg():
    while True:
        c, addr = server_socket.accept()              #wait till a client accepts connection
        connected = Message(sender="Host", response="USERNAME")
        c.send(pickle.dumps(connected))               #send connection message to client, so we can receive bot name
        username = pickle.loads(c.recv(1024)).sender  #recive message string from server
        if username not in bot_username:              #If connection is done successfully
            bot_username.append(username)             #
            person = Client(name=username, address=addr, connection=c)  # creating object of Person
            # sending the connection confirmation back to the client
            clients[username] = person
            # a print out message just for the server
            info = f"New connection from {person.address} "\
                   f"{person.name} has joined the chatroom"
            print(info)
            connection_ok = Message(sender="Host", response=f"Introducing our awesome bot {person.name}!."
                                                            f"\nWating for all clients to join the chat.")
            pickle_msg = pickle.dumps(connection_ok)
            person.connection.send(pickle_msg)                # sending the connection confirmation back to the client
            thread = threading.Thread(target=client_action, args=(person.connection,))  # creating new thread for handling the client
            threads[username] = thread                        # adding the thread to a dictionary of threads
        else:
            print(f"Bot {username}, is taken please try again with another bot name.")
            c.send(pickle.dumps(connection_bad))
            bot_username.pop(bot_username.index(username))
            c.close()
        # if all the bots are connected, then the chat begins with a suggestion message from the server
        if len(clients) == 4:
            time.sleep(2)
            suggested_msg = my_bot()              # there is a simple bot function that decides the first message
            print(f"{suggested_msg.sender}: {suggested_msg.response}")
            connection_msg(suggested_msg, clients)   # sends the message to all the bots
            break
        else:
            print(f"Waiting for {4 - len(clients)} client(s) to join the chat.")  # not sure yet
            continue
    start_threads()  # starting the threads
```

Client.py explanation

Msg_fromServer takes connection socket. Receive message from the server and save messages from server/host in a list get_server_message[] and get_bot_responses[]. get_server_message[] contains suggestion message from host which will be printed out as a suggestion from host in clients side. get_bot_responses[] contains bots response which will be printed out as well. The connection ends after all messages from the list has been sent out.

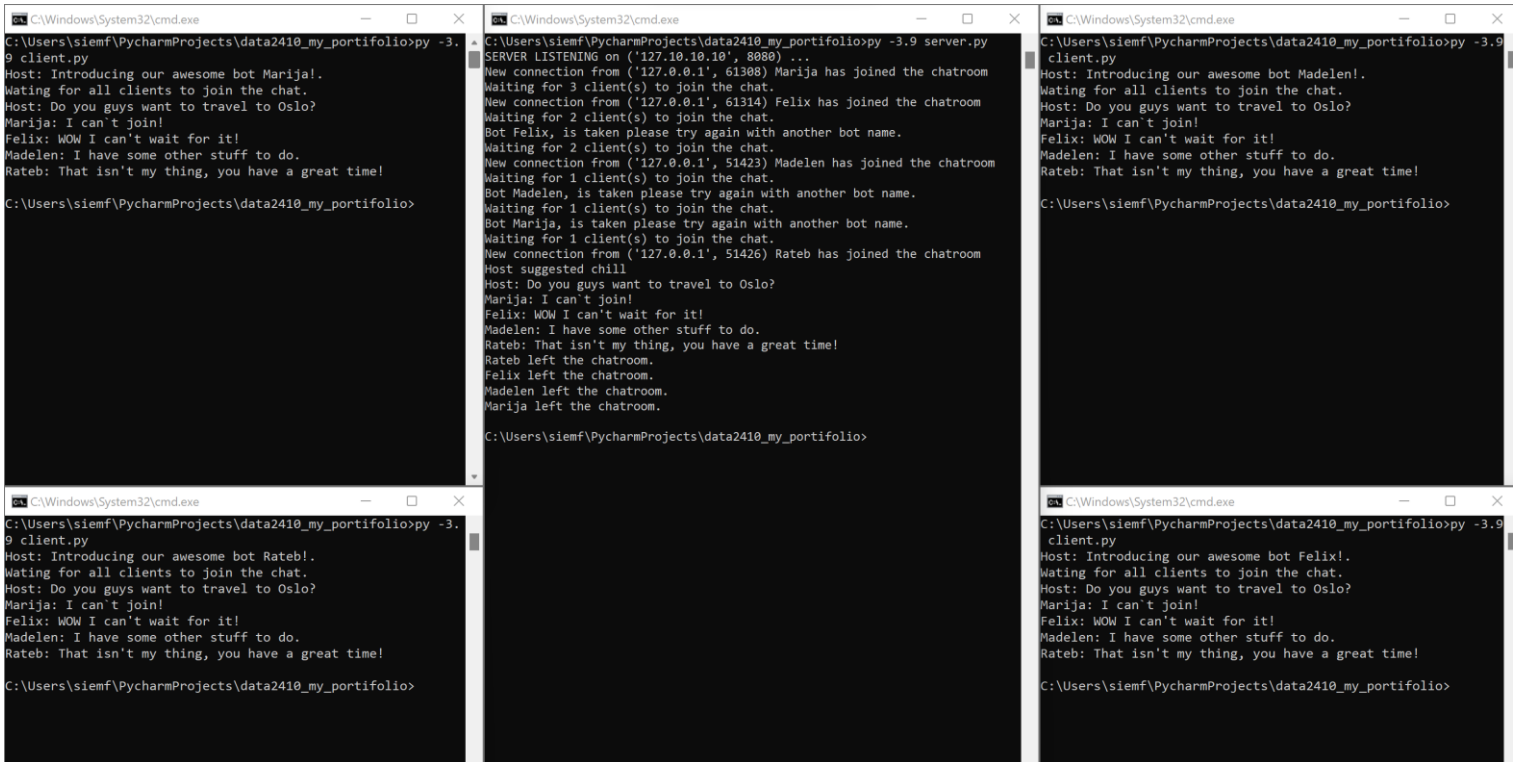
```python
def msg_fromServer(connection: socket):
    while True:
        try:
            msg_received = connection.recv(1024)  # receive as long as message strings are not empty
            if msg_received:
                msg: Message = pickle.loads(msg_received)  # deserialize byte stream to python object
            else:
                continue
            if msg.sender == 'Host':
                if msg.response == 'USERNAME':
                    random_usr = Message(sender=USERNAME)  # serialize Usernames into byte stream
                    connection.send(pickle.dumps(random_usr))
                else:
                    get_server_messages.append(msg)  # appends message from host to be added in the list
                    print(f"{msg.sender}: {msg.response}")  # Message from host, suggesting...
            else:
                get_bot_responses.append(msg)  # appends replays from bots to be added in list
                time.sleep(2)  # suspends executaion of the thread for 2s
                print(f"{msg.sender}: {msg.response}")  # Message from bots

            if len(get_bot_responses) == 4:  # When all clients have responded the connection ends
                disconnect = Message(sender=USERNAME, response="Bye")
                connection.send(pickle.dumps(disconnect))
                break
        except OSError as e:
            print(e)
            break
    connection.close()
```

```python
# function that imports bots responses by using choose_bot function
# which contains the attributes: bot name and response
def w_msg(connection: socket):
    while True:
        if len(get_server_messages) == 1:
            msg = choose_bot(USERNAME, get_server_messages[0])
            pickle_msg = pickle.dumps(msg)
            connection.send(pickle_msg)
            break
        else:
            continue
```

Conclusion

As the result shows below int the picture. When we run the code, the server waits for four connections. When a client is connected, it prints out confirmation of the connection address and port number as well as the username of the client. On the client's side the host welcome the client and inform to wait for all clients to be connected. When all clients are connected the host suggest action (Do you guys want to travel to Oslo), and all the clients see and respond in the chat. After one round of action and response the clients leave the chatroom, and server ends the connection between.



References

Geeksforgeeks.org, (01 Jun, 2021), Python object serialization.

https://www.geeksforgeeks.org/pickle-python-object-serialization/?ref=gcse

Geeksforgeeks.org, (13 Nov, 2018), Understanding python pickling with example

https://www.geeksforgeeks.org/understanding-python-pickling-example/?ref=gcse

Gigi Sayfan, (25 Aug, 2016), Serialization and Deserialization of Python Objects.

https://code.tutsplus.com/tutorials/serialization-and-deserialization-of-python-objects-part-1--cms-26183

knowledgehut.com, (04 Juni, 2020), Python Tutorial (various codes)

https://www.knowledgehut.com/tutorials/python-tutorial

sentdex, Youtube, (02 Apr, 2019), Socket Chatroom server python

https://www.youtube.com/watch?v=CV7_stUWvBQ&list=RDCMUCfzlCWGWYyIQ0aLC5w48gBQ&start_radio=1&t=1299s

sentdex, Youtube, (22 mai 2015), Python Pickle Module for saving objects (serialization)

https://www.youtube.com/watch?v=2Tw39kZIbhs

NeuralNine, Youtbe (), Simple TCP Chat Room in Python

https://www.youtube.com/watch?v=3UOyky9sEQY&t=163s

+ notes from the lecture, labs, and published files on canvas.