Individual portfolio assignment 1

Kader Hussein Abdi

S354379

DATA2410

Table of Contents

Introduction

I have made a chat bot server that follows the specifications given by the assignment. The server sends out a suggestion to all the connected clients, these clients take message and look for known actions. Once the action is found the clients use the command line given bot function to construct an answer. The answer is than sent back to the server and the server broadcasts it to all other connected clients.

Bot explanation

I have four bot functions and a user input function. All these functions return a string which is than used to send a response to the server.

```python
import random

# All the bot functions
def alice(a, b = None):
    return "I think {} sounds awesome!".format(a + "ing")

def bob(a, b = None):
    if b is None:
        return "Not sure about {}. Don't I get a choice?".format(a + "ing")
    return "Sure, both {} and {} seems ok to me".format(a, b + "ing")

def dora(a, b = None):
    alternatives = ["coding", "singing", "sleeping", "fighting"]
    b = random.choice(alternatives)
    res = f"Yea, {a}ing is an option. Or we could do some {b}."
    return res

def chuck(a, b = None):
    action = a + "ing"
    bad_things = ["fighting", "bickering", "yelling", "complaining"]
    good_things = ["singing", "hugging", "playing", "working", "eating", "crying", "sleeping", "coding"]

    if action in bad_things:
        return "YESS! Time for {}".format(action)
    elif action in good_things:
        return "What? {} sucks. Not doing that.".format(action)
    return "I don't care!"

# User function that allows user to communicate with the bots
def user(a, b = None):
    print(a)
    return input("What is your response?: ")
```

The bots take inn two arguments. The b parameter is set to None as default because it is an optional argument.

```
alice(a, b = None):
```

The A parameter is an action found in the suggestion sent by the server. This is the primary action used to construct the bot's answer. The B parameter is an action found in the response of another client. This parameter is optional.

Client explanation

The client starts of by taking inn three command line arguments. The first two are host and port. These two values will be where the client looks for a server.

The last argument is which bot the server will be using, if a unknown bot name is given, the program defaults to a user controlled client.

```
# List to turn string to function poiunter.
bots = {"bob": bob, "alice": alice, "dora": dora, "chuck": chuck, "user": user}
host, port, bot = sys.argv[1], int(sys.argv[2]), sys.argv[3]

# Make all unkown bots to user.
if (bot not in bots): bots[bot] = user
```

Once the program has been provided with those arguments, it starts trying to connect to a server. The connection type it will be looking for is an ipv4 TCP connection. (Data2410, 2022)

```
# Connect to the given ip and port
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((host, port))
```

Once it has connected to the server, the clients sets blocking off so that receiving messages doesn't block the program and it sends the server the name of the bot/user.

```
client_socket.setblocking(False)
# Send your name to the server
client_socket.send((bot.capitalize()).encode())
```

With that the client is done with setup and is ready to get into the loop of receiving and sending messages. To make this loop easier the client program has to helper functions, one for receiving messages and another for sending messages.

```
# Function that returns a tuple of actions from server and actions from client
def receive_message():
    sAction, cAction = "", None
    msg = client_socket.recv(1024)
    if not len(msg):
        print('Connection closed by server')
        sys.exit()

    if (msg.decode().find("Host") != -1):
        sAction = [action for action in actions if msg.decode().find(action) != -1][0]
        return (sAction, cAction)
    if bot == "user":
        sAction = msg.decode()
        print(f"Got message from server!")
        return (sAction, cAction)
    else:
        cAction = [action for action in actions if msg.decode().find(action) != -1][0]
        print(f"Got message from another client!")
        (sAction, cAction)

# Send a message to the server
def send_message(sAction, cAction):
    if cAction == sAction: cAction = None
    msg = bots[bot](sAction, cAction).encode()
    print(f"Sending response: {msg.decode()}\n")
    client_socket.send(msg)
```

The receive function takes the received message and checks it. It first checks if the message is empty, if the message is empty, it means the connection to the server has been cut, in which case the functions sets the systems exit flag and prints relevant message to the user. If the message has a “Host” in it string it means the message is from the server, in which case the server_action variable is set to the action that is found in the sentence.

If there is not “Host” in the message the client assumes it received a message from another client and sets the client_action to the relevant action found in the message.

With either case of the if sentence, the function returns a tuple of both values.

The send function is much simpler, it takes the action given as arguments and sends those actions to the bots and uses the returned message to send to the server. (Data2410, 2022)

```
# Send a message to the server
def send_message(sAction, cAction):
    if cAction == sAction: cAction = None
    msg = bots[bot](sAction, cAction).encode()
    print(f"Sending response: {msg.decode()}\n")
    client_socket.send(msg)
```

Finally, we get to the main part of the program.

```
# Current actions
action_server, action_client = "", None

# Endless loop to receive and send messages
while True:
    # If an suggestion is given by server, respond
    if action_server:
        send_message(action_server, action_client)

    action_server, action_client = "", None

    # Since the clients receive is non blocking an error will be thrown if there is no message, ignore this.
    # But do not ignore the system exit flag.
    try:
        while True:
            action_server, action_client = receive_message()
    except Exception as e:
        if e is SystemExit:
            break
```

This is an endless loop which checks for messages and responds. Since the client is non-blocking the receive message function throws an error if there is nothing to receive so the try and except ignores the errors except for the system exit flag.

Server explanation

The server starts off by taking inn a command line parameter for the port it will be binding and listening to.

```
# The ip and port adresees the server will be listening for
host, port = '', int(sys.argv[1])
```

After it has gotten the information needed, it sets up a socket IPv4 TCP socket on the corresponding IP and port. (Data2410, 2022)

```
# Bind the server to ip and port and listen for clients.
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((host, port))
server_socket.listen()
```

The way I have decided to setup the communication the server does to the bots is by having an array of pickable suggestions that the user chooses by typing inn a index in the command line. This is the array of suggestions.

```
# The list of all avaiable suggestions
suggestions = ["Why don't we sing", "Let's take a walk"]
```

The server I made uses select to manage the communication between the server and the client. Select allows for an event driven system for communication between server and client. The server sets up its socket list for all sockets it wants to check for an event.

```
# Make a list of sockets and clients.
sList = [server_socket]
active_clients = {}
```

The reason we have to separate data structures for the same data is because the socket list includes the server socket and in many operations we want to separate between the server and the clients. The socket list is for the select module, once a client connects to the server the server the select function will notify us. (Python docs, 2022) We can use this to easily connect new client while still communicating with others.

```python
# Create a endless loop for back and fourth between server and client.
while True:
    read_sockets, write_sockets, error_sockets = select.select(sList, [], sList)

    # Looping over all reading socket and preforming an action
    for sock in read_sockets:
        # If the readable socket is server, we know it is a new connection
        # Add this connection to list of all sockets, and list of clients.
        if sock == server_socket:
            c, addr = server_socket.accept()
            sList.append(c)
            name = c.recv(1024).decode()
            names.append(name)
            print(f"{name} has joined the chat!")
```

When ever the select module provides us with readable sockets, we loop over them and check if the readable socket is the server, if it is we know that a new connection has been made. We add the client socket the socket list so that we can check that client for read events. (Python docs, 2022) We do not yet add the client socket to the active_clients list because we are still in a “communication round” with the other clients. We add the client name to an array to keep track of which clients we have to add in the next round.

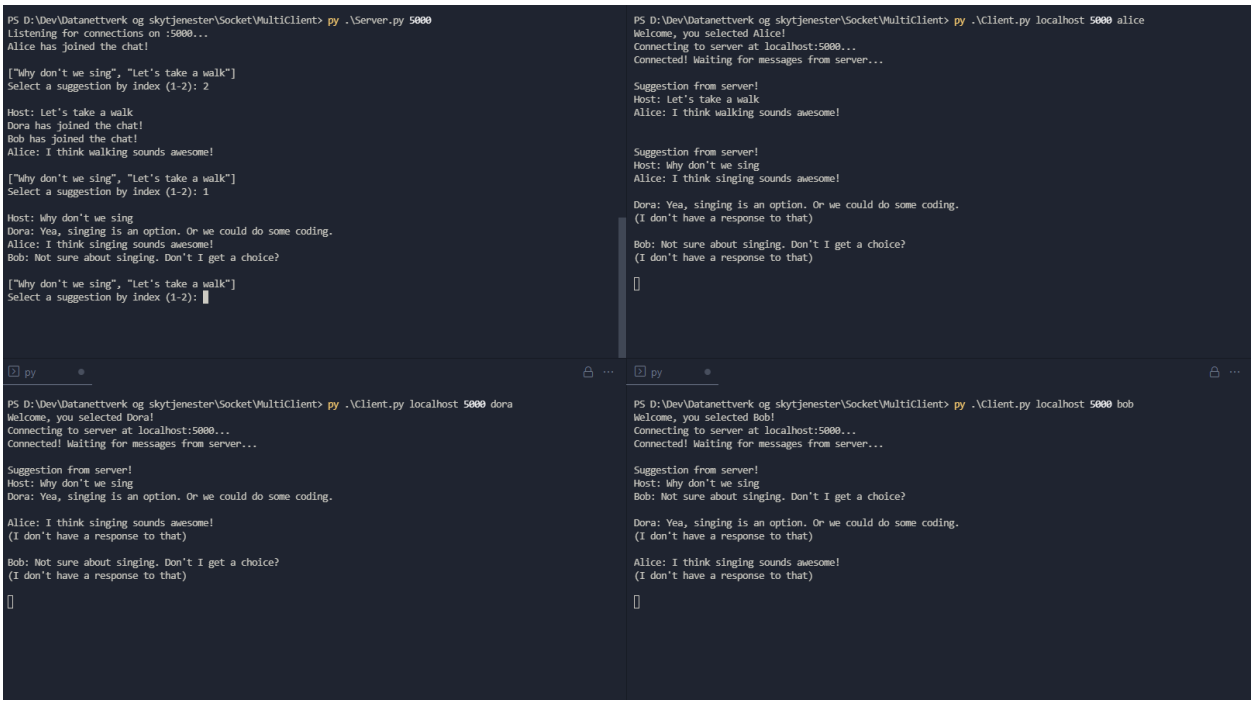
```
        # If the readable socket isn't the server we know it is message
        # from one of the clients. Read the message and broadcast it to all clients.
        else:
            msg = sock.recv(1024)
            # If the message is empty it is a clean exit from the client.
            # Remove the corresponding client from socket list and client list.
            if not len(msg):
                print(f"Closed connection to {active_clients[sock]}")
                sList.remove(sock)
                del active_clients[sock]
                continue

            # Print the message and keep count over how many message has been received
            print(f"{active_clients[sock]}: {msg.decode()}")
            message_count += 1

            # Boradcast to all clients
            for client in active_clients:
                if (client == sock): continue
                client.send(msg)
```

If the readable socket isn't the server, we know that it is a response to our suggestion. We read the response and broadcast it to all other clients.

```
    # Check for if all the messages has been received before sending a suggestion.
    if message_count == expected_message_count:
        for index, name in enumerate(names):
            active_clients[sList[-1 - index]] = name

        names.clear()

        message_count = 0
        expected_message_count = 0
        # Show all avaiable suggestions to user
        print("\n" + str(suggestions))

        # Let user choose which suggestion to pick
        inpt = int(input("Select a suggestion by index (1-2): "))

        # Assume user picks valid index, select that message as the suggestion.
        suggestion = "Host: " + suggestions[inpt-1]

        # Print the suggestion and send it to all connected clients,
        # and keep track of how many answers you expect
        print("\n" + suggestion)
        for client in active_clients:
            client.send((suggestion).encode())
            expected_message_count += 1
```

We loop over all the client names we added to the names array and add these clients to the active_clients array so that they can join in the next round of communication. After that we ask

the user to input an index into the suggestions array and use that suggestion for the next round of communication. We loop over all active_clients and send a suggestion. Once we are done we go back to the top of the endless loop where we have our Select function which will give us all the readable sockets.

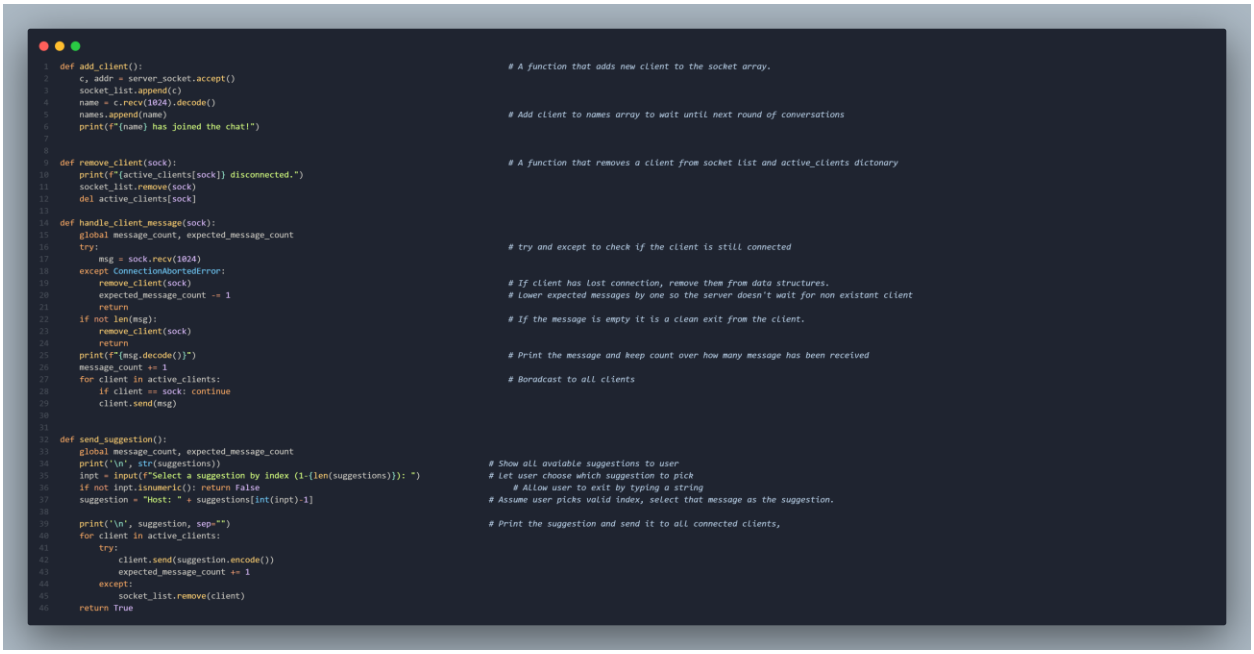


Code readability

All necessary variables and data structures for server

```python
host, port = '', int(sys.argv[1])                                   # The ip and port adresees the server will be listening for

suggestions = ["Why don't we sing", "Let's take a walk",
               "Let's yell!", "I feel like fighting right now",
               "Maybe we could try some bickering?",
               "What do you say about we start hugging?",
               "Let's start working"]

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)   # Bind the server to ip and port and listen for clients.
server_socket.bind((host, port))
server_socket.listen()

socket_list = [server_socket]                                       # Make a list of sockets and clients.
active_clients = {}
names = []

expected_message_count = 0                                          # variables for keeping track of how many messages you expect, and the current message count
message_count = 0
```

All necessary functions for server



```python
def add_client():                                                   # A function that adds new client to the socket array.
    c, addr = server_socket.accept()
    socket_list.append(c)
    name = c.recv(1024).decode()
    names.append(name)                                              # Add client to names array to wait until next round of conversations
    print(f"{name} has joined the chat!")


def remove_client(sock):                                            # A function that removes a client from socket list and active_clients dictonary
    print(f"{active_clients[sock]} disconnected.")
    socket_list.remove(sock)
    del active_clients[sock]

def handle_client_message(sock):
    global message_count, expected_message_count
    try:                                                            # try and except to check if the client is still connected
        msg = sock.recv(1024)
    except ConnectionAbortedError:
        remove_client(sock)                                         # If client has lost connection, remove them from data structures.
        expected_message_count -= 1                                 # Lower expected messages by one so the server doesn't wait for non existant client
        return
    if not len(msg):
        remove_client(sock)                                         # If the message is empty it is a clean exit from the client.
        return
    print(f"{msg.decode()}")                                        # Print the message and keep count over how many message has been received
    message_count += 1
    for client in active_clients:                                   # Boradcast to all clients
        if client == sock: continue
        client.send(msg)


def send_suggestion():
    global message_count, expected_message_count
    print('\n', str(suggestions))                                   # Show all available suggestions to user
    inpt = input(f"Select a suggestion by index (1-{len(suggestions)}): ")  # Let user choose which suggestion to pick
    if not inpt.isnumeric(): return False                           # Allow user to exit by typing a string
    suggestion = "Host: " + suggestions[int(inpt)-1]                # Assume user picks valid index, select that message as the suggestion.

    print('\n', suggestion, sep="")                                 # Print the suggestion and send it to all connected clients,
    for client in active_clients:
        try:
            client.send(suggestion.encode())
            expected_message_count += 1
        except:
            socket_list.remove(client)
    return True
```

Server program loop

```python
print(f'Listening for connections on {host}:{port}...')
while True:                                                         # Create a endless loop for back and fourth between server and client.
    read_sockets, write_sockets, error_sockets = select.select(socket_list, [], socket_list)

    for sock in read_sockets:                                       # Looping over all reading socket and preforming an action
        if sock == server_socket:                                   # If the readable socket is server, we know it is a new connection
            add_client()                                            # Add this connection to list of all sockets, and list of clients.
        else:                                                       # If the readable socket isn't the server we know it is message from one of the clients. Read the message and broadcast it to all clients.
            handle_client_message(sock)

    for sock in error_sockets:                                      # Looping over all error event sockets and removing them
        remove_client(sock)

    if message_count == expected_message_count:                     # Check for if all the messages has been received before sending a suggestion.
        for index, name in enumerate(names):                        # Loop over all names and add them to the active_clients
            active_clients[socket_list[-len(names) + index]] = name
        names.clear()

        message_count = 0
        expected_message_count = 0

        if not send_suggestion(): break
```

Program setup for client

```python
actions = ["work", "play", "eat", "cry", "sleep",                    # List of known actions
           "fight", "sing", "hug", "bicker", "yell",
           "complain", "walk"]


bots = {"bob": bob, "alice": alice, "dora": dora, "chuck": chuck, "user": user}    # List to turn string to function poiunter.
host, port, bot = sys.argv[1], int(sys.argv[2]), sys.argv[3]
if (bot not in bots): bots[bot] = user                               # Make all unkown bots to user.

print(f"Welcome, you selected {bot.capitalize()}!")
print(f"Connecting to server at {host}:{port}...")

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)    # Connect to the given ip and port
client_socket.connect((host, port))

client_socket.setblocking(False)
client_socket.send((bot.capitalize()).encode())                      # Send your name to the server

print("Connected! Waiting for messages from server...")

```

All necacery functions for Client.py

```python
def receive_message():                                               # Function that returns a tuple of actions from server and actions from client
    server_action, client_action = "", None
    msg = client_socket.recv(1024)
    if not len(msg):                                                 # if the message is an empty string, assume server is disconnecting
        print('Connection closed by server')
        os._exit(0)

    if (msg.decode().find("Host") != -1):                            # Check the message for the "Host"
        print(f"\nSuggestion from server!")
        print(f"{msg.decode()}")
        server_action = [action for action in actions if msg.decode().find(action) != -1][0]    # Loop over all actions and check if that action can be found in the message
        return (server_action, client_action)
    else:
        client_action = [action for action in actions if msg.decode().find(action) != -1][0]    # Loop over all actions and check if that action can be found in the message
        print(f"{msg.decode()}\n(I don't have a response to that)\n")
        return (server_action, client_action)

def send_message(server_action, client_action):                      # Send a message to the server
    if client_action == server_action: client_action = None
    msg = (f"{bot.capitalize()}: " + bots[bot](server_action, client_action)).encode()
    print(f"{msg.decode()}\n")
    client_socket.send(msg)

def command_line_input_thread():
    while True:
        inpt = input()
        if inpt == 'q' or inpt == 'quit':
            client_socket.close()
            os._exit(0)

```

Client program loop.

```
threading.Thread(target=command_line_input_thread).start()

action_server, action_client = "", None                    # Current actions
while True:                                                # Endless loop to receive and send messages
    if action_server:                                      # If an suggestion is given by server, respond
        send_message(action_server, action_client)
    action_server, action_client = "", None

    try:                                                   # Since the clients receive is non blocking an error will be thrown if there is no message, ignore this.
        while True:
            action_server, action_client = receive_message()
    except Exception as e:
        if e is SystemExit:                                # But do not ignore the system exit flag.
            break
```

Conclusion

In conclusion I made a program that takes inn an index from the user and sends the corresponding suggestion to the client. The client sends back a response that is based on which bot function you pick. The server forwards the message to all other clients.

Referanser

Alzarqawee, A. N. (2022). No lecture(8): Portfolio 1 guidance session. Oslo, Norge.

Data2410. (2022). Lecture 7 - Python for network programming . Oslo, Norge.

Python docs. (2022, March 24). socket — Low-level networking interface.