

Individual Portfolio Assignment 1

Author: Charlie Vo

Student id: s188910

Oslo Metropolitan University

Introduction	3
How to use the Application	3
The Game/Chat experience.	4
Method	6
The Clients	6
The Admin role	7
The BOT role	9
The Server	11
Configuring the server	11
Start game logic	13
The command detection logic	14
Communication logic	15
Get username and get connection logic	15
Connection close logic	16
Result	17
Summary	17
References	18
Videos	18
Websites	18

Introduction

In this assignment, I have created a chat application where a server and a client are configured to act as a send and receive messaging service. The two main files are `client.py` and `server.py`. My choice of programming language is Python.

The application in itself is built on basic send and receive functions where the server accepts 5 connections. The 5 client connections should consist of 1 Lord (Which is you!) and 4 bots either named or randomly selected from a pool of names (From the Game of Thrones universe).

How to use the Application

To launch the application in its entirety, you need to access the folder from a CLI and launch the `server.py` by writing `python3 server.py`. Once the server is running, you can proceed with connecting 1 Admin client with the command `python3 client Lord`. Lastly, you need to connect 4 bots by typing `python3 client` for each of the bots. You also have the alternative to name your bots by simply appending a username like this `*python3 clients.py <username>*`.

Once the Server, Admin, and Bots are up and running you will see an announcement that 5/5 connections have been established and the main event is about to start.

The Game/Chat experience.

Once the game has 5 connections, you are presented with a new event that describes the situation. Each person announces a series of options that you can decide to proceed with. Below you will see different events that are used to showcase the variety of options and functions of this chat application.

“Nothing is too difficult for you and your team but beware of what you say and wish for. For based on your choice of action your affiliates will either show appreciation or distaste against you as the Lord. Command them carefully or they will doubt your leadership. Good luck on your journey and may the odds ever be in your favor!”

```
1 ----- Welcome to the Game of Thrones -----
2 You now have 5/5 individuals present in your domain to heeding your every command.
3 If you need assistance, --help will provide you with the information you need.
4
5 ----- [New Event] -----
6 Tyrion: We caught this thief 🗡️ drinking around the streets, which is not allowed
7 by your rules my Lord! We should sentence him to eternal 1 slaughtering or 2
8 eating. What will it be?
9
10 You:
```

“You of course sentence him to eternal eating, and all your brothers and sisters in arms are happy with your decision. So happy that they are joining in on this wonderful feast.”

```
1 You: Let him eat to his death!
2
3 Danny: 🟩 eat? Proposal accepted! 🗡️🗡️🗡️
4 Ramsey: 🟩 Wait a minute sire, eat? I cannot believe this day has come. 🐎🐎
5 Jamie: 🟩 Thats right my Lord! Today is the day of eating!! 🔥🔥🔥
6 Joff: 🟩 EAT!!EAT!!EAT!! 🍽️🍽️🍽️
```

“There is a new event that has surfaced and your friend Tyrion is caught red-handed. Your crew knows you are very strict with the rules and are enforcing them by suggesting you to punish the guilty.”

```
1 ----- [New Event] -----
2 Khal: Not again! We are all out of food because Tyrion chose to stroll all day! We sho
   uld punish the lazy by a forceful 1 torture or 2 hunting session. What will it be?
3
4 You: None of those! I wish him no harm.
```

“Despite receiving wise words from your men, you wish Tyrion no harm so you pardon his crimes. Your affiliates did not like your decision and voiced their thoughts against your leadership.”

```
1 You: None of those! I wish him no harm.
2
3 Danny:  Please my lord, may you reconsider this? 😞
4 Jamie:  Request unhead of! 😞
5 Ramsey: Oh please no! I have to get back to my wife and kids! 😞
6 Joff:   Not to be rude, but I had other thoughts! 😞
```

“Your crew's response to your choice of action angers you and you believe they should all face punishment for not heeding your call. But you like Ramsey, and always have. So you allow him to go home to his wife and kids.”

```
1 You: --dismiss Ramsey
2 You have dismissed Ramsey, 4/5 left on your team.
```

These are some of the responses that the server-client chat application has to offer. All the responses have parts of the message randomly chosen when called, allowing the game to

give a unique experience every time. Commands are also available during an event and will tell you to use the right one if you do happen to spell it incorrectly.

Method

As presented in the introduction, this is a chat application based on the Client-Server Architecture. To elaborate, the clients who are acting as hosts for this network are connected to one centralized server which provides data distribution service as well as host handling functionality. The hosts are configured to be communicating with each other by sending messages to the server where the data is formatted and forwarded to the clients.

The server in this case is configured to only handle the logical parts of the application while the chat protocols are present and visible only to the admin client.

The Clients

To be able to connect to the client, there are three parameters that can be passed in using `sys.args[#]` allowing the admin to define the variable `HOST`, `PORT`, and `USER_NAME`. I have in this case only provided the option to pass in one parameter, the `USER_NAME`, as it is more convenient for the admin to run and test the program. While the `USER_NAME` can be passed in as an input parameter it is not required to launch the script, as a random name is fetched from a list of names if no argument is present when launching the script.

Each of the clients will then have the remaining parameters `HOST` and `PORT` bound to the socket along with the socket network type `AF_INET` and `SOCK_STREAM` for TCP.




```

1  USER_NAME = sys.argv[1] if len(sys.argv) >= 2 else event.random_botname()
2  HOST = socket.gethostname()
3  PORT = 2345
4
5  client_connection = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6  client_connection.connect((HOST, PORT))

```

There are two roles that the client can possess. Either the admin role is specified by naming the client “Lord” or the Bot by passing in any parameter e.g “Rolf”. The script will upon execution call a role check to decide which one to give the client and start the corresponding thread.



```

1  def connect_client(USER_NAME):
2      # Registers the clients username on the server.
3      username = USER_NAME.capitalize()
4      client_connection.sendall(username.encode())
5
6      if username.lower() == 'lord':
7          threading.Thread(target=admin_rcv).start()
8          print(f'Connecting as Lord of the realm')
9      else:
10         threading.Thread(target=bot_rcv).start()
11         print(f'Connecting as {username}.')

```

Now with the socket explanation and role delegations out of the way let's continue with the logical part of the script.

The Admin role

The purpose of the Admin role is mainly directing and giving the right to proceed with protocols on the chat server. Once the server has registered 5 connected client sockets it will send a command starting a thread that invokes the ability to type in the input box for anyone with the Admin role assigned. The Admin is then able to send messages to the

server triggering responses from the connected bots, in addition to sending commands which will affect other clients.

```
1 def admin_recv():
2     while True:
3         try:
4             # Listens to messages sent from the server.
5             message = client_connection.recv(1024).decode()
6             if message == 'request_input':
7                 threading.Thread(target=admin_send).start()
8             else:
9                 print(f'{message}')
10        except:
11            print('Connection has been closed.')
12            client_connection.close()
13            break
```

The admin receiver is actively listening to the “request_input” string from the server ready to fire off a thread initiating the admin_send() function as seen below. In addition to this functionality, the send function is wrapped in a Try-/Except block that terminates the client connection upon detecting errors. The purpose of this is to terminate the client connection when it is idling without serving valuable to the other clients. Many of the errors that occur during .recv() and .sendall() are when the client is trying to send messages after disconnecting from the server, and we want to terminate the connection when that happens.

```
1 def admin_send():
2     while True:
3         sleep(2)
4         event.get_event()
5         try:
6             response = input('\nYou: ')
7             client_connection.sendall(response.encode())
8         except:
9             client_connection.close()
10            break
```


The last logical bit of code given to the admin role is the send function. The function is running on a thread allowing the admin to continuously send messages through the socket. The `sleep()` function is included to allow the client to receive and print messages without too eagerly prompting an input request. `event.get_event()` is the protocol explained in the game/experience section.

The BOT role

The Bot role is made so it passively translates messages sent from the server and will trigger actions only if it detects a message from the admin with the “Lord:” prefix. This is the method I chose to configure the bots to ignore messages sent from other clients.

```
1  def bot_recv():
2      while True:
3          try:
4              # Handles only messages sent from the "Lord" and ignores the rest.
5              message = client_connection.recv(1024).decode()
6              user_reply = message.split()
7              if user_reply[0].lower() == 'lord:':
8                  response = bot_send(message)
9                  client_connection.send(f'{response}'.encode())
10         except:
11             print('Connection has been closed.')
12             client_connection.close()
13         break
```

Note that this code block in figure 2.1.4 also calls upon a local function called `bot_send()`. This function serves the same function as the one integrated into the admin role with the only difference being that the messages are already pre-defined. The function `bot_send()` as seen in figure 2.1.5 below, takes a string argument `message` which consists of the entire response from the admin and splits it into substrings, and stores it in an array. This way it allows the message to be traversable and comparable.

```

1 def bot_send(message):
2     # Splits the message and stores it in an array while filtering away symbols using rege
   X.
3     word_list = re.split(r'\s+|[,;?!.\s*', message.lower())
4
5     # Loops through the array and compares with a list of actions imported from event.py
6     for action in event.action_list:
7         for word in word_list:
8             if word == action or word == f'{action}ing':
9
10                bot_response = f'█ {event.positive_response(word)}'
11                return bot_response
12
13    # If word is not found, return generic message.
14    bot_response = f'█ {event.negative_response()}'
15    return bot_response

```

The function then takes every substring and compares it to every entry in the `action_list` stored in the `event.py` file. The `event.py` file consists of many pre-formatted responses that the bot can use to send a meaningful message back to the admin through the socket if the given substring matches an action in the `action_list`. Since the word is passed through to get the response, the message returned will also contain the same message making the bots feel more life-like and organic.

```

import random

# Responses has refactored to unclutter and separate logic from trivial code.

name_list = [
    'Ago', 'Samsa', 'Dante', 'Robb', 'Geri', 'Jon', 'Tyron', 'Bran',
    'Edward', 'Danny', 'Robert', 'Theon', 'Joff', 'Khal', 'Marga', 'Sansa'
]

action_list = [
    'love', 'eat', 'sleep', 'play', 'fight', 'hunt', 'read',
    'right', 'brawl', 'swim', 'ride', 'stroll', 'hide', 'conquer',
    'drink', 'torture', 'slaughter',
]

icon_list = [
    '🐘', '🐘', '🐘', '🐘', '🐘', '🐘', '🐘', '🐘', '🐘', '🐘',
    '🐘', '🐘', '🐘', '🐘', '🐘', '🐘', '🐘', '🐘', '🐘', '🐘',
]

def positive_response(word):
    response_list = [
        f'Thats right my lord! Today is the day of {word}! 🐘🐘🐘',
        f'{word}? Proposal accepted! 🐘🐘🐘',
        f'Your order is my command, my lord! {word} is all I wish for us to do. 🐘🐘🐘',
        f'Wait a minute sire, {word}? I cannot believe this day has come. 🐘🐘🐘',
        f'If {word} is what my king says, then {word} is what his loyal men will do. 🐘🐘🐘',
        f'{word.upper()}!!!! lets goooo!! 🐘🐘🐘',
        f'Its a party! Everyone, lets go its time for {word}! 🐘🐘🐘',
        f'{word.upper()} is {word.upper()}!! if {word.upper()} is 🐘🐘🐘',
    ]
    return random.choice(response_list)

def negative_response():
    response_list = [
        f'I dont know what that is! 🐘',
        f'Please my lord, may you reconsider this? 🐘',
        f'I believe we could do better. 🐘',
        f'Not to be rude, but I had other thoughts! 🐘',
        f'Oh please no! I have to get back to my wife and kids! 🐘',
        f'Pardon me lord? 🐘',
        f'I heed your call, but please allow to to say my farewells? 🐘',
        f'Mmm ... That might not be too wise ... 🐘',
        f'Request ahead of! 🐘',
        f'! 🐘 🐘',
    ]
    return random.choice(response_list)

```

```


1 def random_botname():
2     return random.choice(name_list)
3
4
5 def get_event():
6     event_list = [
7         f'{random.choice(name_list)}: There is a sudden attack from the north! Their tactic this
   time seem to be {random.choice(icon_list)} {random.choice(action_list)}ing while {random.choice(
   icon_list)} {random.choice(action_list)}ing! What do you want us to do? Our best options are ei
   ther 🐘 {random.choice(action_list)}ing or 🐘 {random.choice(action_list)}ing the invaders. C
   hoose quickly!',
8         f'{random.choice(name_list)}: My Lord! We are about to get raided, the attackers are {ra
   ndom.choice(icon_list)} {random.choice(action_list)}ing outside waiting for us to {random.choice(
   icon_list)} {random.choice(action_list)} on the inside. Best course of action should be to reta
   liate by 🐘 {random.choice(action_list)}ing, we could also 🐘 {random.choice(action_list)} t
   he enemy, the choice is yours!',
9         f'{random.choice(name_list)}: Not again! We're all out of food because {random.choice(na
   me_list)} chose to {random.choice(action_list)} all day! We should punish the lazy by a forceful
   🐘 {random.choice(action_list)}ing or 🐘 {random.choice(action_list)}ing session. What will i
   t be?',
10        f'{random.choice(name_list)}: We caught this thief {random.choice(icon_list)} {random.ch
   oice(action_list)}ing around the streets, which is not allowed by your rules my lord! We should
   sentence him to eternal 🐘 {random.choice(action_list)}ing or 🐘 {random.choice(action_list)}
   ing. What will it be?',
11        f'[SPECIAL EVENT] Horseriding is coming up 🐘🐘🐘. How should we sabotage their horses?
   Choose one: 🐘 {random.choice(action_list)}, 🐘 {random.choice(action_list)}, 🐘 {random.ch
   oice(action_list)} or 🐘 {random.choice(action_list)} the horses.',
12        f'[SPECIAL EVENT] The army is recruiting specialists to fight on the battlefield
   🐘🐘🐘. What does your team specialize in? Choose one: 🐘 {random.choice(action_list)}, 🐘 {
   random.choice(action_list)}, 🐘 {random.choice(action_list)}, 🐘 {random.choice(action_list)}
   ',
13    ]
14    print(f'\n----- [New Event] -----')
15    print(random.choice(event_list))

```

The `client.py` contains a lot of interesting responses that use `random.choice()` operator to generate many different permutations of each response, making it more interesting and engaging for the user. Although they are nice and neat they provide no functional value for the application, thus being separated and refactored to a separate file.

The Server

The server socket is configured using a `gethostname()` from the socket library to bind the HOST and PORT to create a server connection. In addition, the server is allowed to accept up to 5 queued hosts as a buffer in case of rapid host connections. The server configurations can also be changed to accept CLI parameters HOST and PORT.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains a Python code snippet for configuring a server socket.

```
1  # Can be replaced with sys.argv[int] to accept CLI input parameter.
2  HOST = socket.gethostname()
3  PORT = 2345
4
5  server_connection = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6  server_connection.bind((HOST, PORT))
7  server_connection.listen(5)
```

Configuring the server

Configuration of the `server_accept` function serves a vital role in its functionality. In this case, I chose to configure the server to be able to accept all incoming socket connections and start each connection with an independent thread. This thread will listen to any messages that the client sends and will parse, translate and check if any commands are detected in the string. The username and connection data are stored in an array both sharing the same index number to allow the admin to request actions performed on both from the server. The server then keeps accepting all incoming connections until there are 5 clients then proceeds to initialize the game protocol on the admin Client.

```

1  def server_accept():
2      print('\nChat server is online! \nNow listening to connections (0/5).\n')
3
4      while len(username_list) < 5:
5          try:
6              # Accepts any new socket connections.
7              client_connection, client_address = server_connection.accept()
8              connection_list.append(client_connection)
9
10             # Accepts first message "username" sent by the client and registers
11             to array.
12             client_username = client_connection.recv(1024).decode()
13             username_list.append(client_username)
14
15             # Each connection has its own thread listening to responses.
16             threading.Thread(target=server_recv, args=(
17                 client_connection, client_username)).start()
18
19             # Prints status on server and notifies the Lord of the realm
20             server_broadcast(f"{client_username} has entered the realm.")
21             print(f"{client_username} has connected ({len(username_list)}/5).")
22
23         except:
24             print('Server error, closing down.')
25             server_connection.close()
26             break
27
28     print("\nThe server now has 5 active connections.")
29     start_game()

```

```

1  def server_recv(client_connection, client_username):
2      # Every message will pass a is_command check before being forwarded.
3      while True:
4          try:
5              message = client_connection.recv(1024).decode()
6              if len(message) > 0:
7                  if is_command(message):
8                      find_command(client_connection, message)
9                  else:
10                     message = f'{client_username}:\t {message}'
11                     client_broadcast(client_connection, message)
12
13             except:
14                 print(f'{client_username} has left the chat.')
15                 client_connection.close()
16                 break

```

The remaining part of the server will consist of logical functions to support the server in handling requests from the admin and will only be explained briefly as they are short and self-explanatory.

Start game logic

Start game will announce the beginning of the game protocol and activates the input function that is coded in `admin*_send()*`.

```
1  def start_game():
2      # Stops incoming connections and activates input on the admin client.
3      server_connection.close()
4      server_broadcast('request_input')
5
6      # A Pleasant announcement to the admin client that the game has begun.
7      server_broadcast(
8          f'\n ----- Welcome to the Game of Thrones -----')
9      server_broadcast(
10         "You now have 5/5 individuals present in your domain to heeding your ev
11         ery command.")
12         server_broadcast(
13             "If you need assistance, --help will provide you with the information y
14             ou need.")
```

The command detection logic

The `is_command` first checks whether the string contains any prefix indicating an incoming command. If detected the string is passed in the `find_command` to see if it is matching any of the listed ones.

```
1 def is_command(message):
2     # If the message has three or less substring and includes symbols: '--'.
3     if len(message.split()) < 3 and message.find('--') == 0:
4         return True
5
6 def find_command(client_connection, message):
7     if '--help' in message:
8         response = ''
9         This is the Game of Thrones, you are presented with a variety of action
10        s can choose from.
11        If you dislike anyone from your team you can always ask them to leave t
12        hem with --dismiss <name>
13        or even --leave, to terminate the session. With that said lets try agai
14        n ..
15        ''
16        client_connection.sendall(response.encode())
17
18    elif '--dismiss' in message and len(message.split()) == 2:
19        target_username = message.split()[1]
20        dismiss_username(client_connection, target_username)
21
22    elif '--leave' in message:
23        print("Server close down has been requested, shutting down the server.")
24    )
25    response = '\nThe Lord has left the realm, everyone is allowed to leav
26    e.'
27    server_broadcast(response)
28    close_connections(client_connection)
29
30    else:
31        message = 'Unknown command. Try [--help/--leave/--dismiss <name>].'
32        client_connection.sendall(message.encode())
```

Communication logic

The communication logic is used by the server to distribute messages and make announcements based on specific given events. It is also used to generate feedback to users upon errors.

```
1 def client_broadcast(client_connection, message):
2     # Used by the server to distribute a message from the admin client.
3     for connection in connection_list:
4         if connection != client_connection:
5             connection.sendall(message.encode())
6             sleep(0.1)
7
8 def server_broadcast(message):
9     # Used by the server to broadcast message to all clients.
10    for connection in connection_list:
11        connection.sendall(message.encode())
12        sleep(0.1)
```

Get username and get connection logic

These getters allow the server to announce the name of those who have disconnected based on their connection id, as well as terminate connections id based on their username.

```
1 def get_username(client_connection):
2     index = connection_list.index(client_connection)
3     username = username_list[index]
4     return username
5
6 def get_connection(username):
7     index = username_list.index(username)
8     client_connection = connection_list[index]
9     return client_connection
```

Connection close logic

Connection close logic handles when the admin user requests a connection to be terminated. If the user is found, the connection, as well as the username, is removed from the socket and the lists containing their data.

close_connections will terminate all connections in turn and then the admin client in the end.

```
1 def dismiss_username(client_connection, target_username):
2     # Looks up if the requested username is in the list of active clients.
3     if target_username in username_list:
4         target_connection = get_connection(target_username)
5         target_connection.close()
6
7         connection_list.remove(target_connection)
8         username_list.remove(target_username)
9
10        server_response = f'You have dismissed {target_username}, {len(username
11        _list)}/5 left on your team.'
12        client_connection.sendall(server_response.encode())
13
14        # If not the client is notified that the user does not exist.
15    else:
16        error_response = (
17            f'There is no one on the realm named \'{target_username}\'.')
18        client_connection.sendall(error_response.encode())
19        print(
20            f'ERROR: Could not find {target_username} in list of connections.')
21
22 def close_connections(client_connection):
23     # This function terminates all the other clients first.
24     for connection in connection_list:
25         if connection != client_connection:
26             username = get_username(connection)
27             connection.close()
28     # Then terminates the requesters connection.
29     response = 'You have left the realm.'
30     client_connection.sendall(response.encode())
31     client_connection.close()
```


Result

Summary

With this, I have successfully managed to create a stable Server-Client Chat application with bots and admin communicating over sockets connections. The chat works as described and handles multiple logical operations both automatically and actively whenever called upon.

The game protocol is fun and engaging to the user, both generating interesting messages while also being meaningful. Bots do only reply to the admin client and take everything the admin client sends into consideration before replying.

References

Videos

Advanced TCP Chat Room in Python. (2020, September 21). [Video]. YouTube.
https://www.youtube.com/watch?v=F_JDA96AdEI

Sockets Tutorial with Python 3 part 1 - sending and receiving data. (2019, March 11). [Video]. YouTube. <https://www.youtube.com/watch?v=Lbfe3-v7yEO>

Websites

GeeksforGeeks. (2021, November 10). Socket Programming in Python. Retrieved March 17, 2022, from <https://www.geeksforgeeks.org/socket-programming-python/>

Python Programming Tutorials. (n.d.). Pythonprogramming.Net. Retrieved March 12, 2022, from <https://pythonprogramming.net/server-chatroom-sockets-tutorial-python-3/>

socket — Low-level networking interface — Python 3.10.4 documentation. (n.d.). Python Docs. Retrieved March 20, 2022, from <https://docs.python.org/3/library/socket.html>

TCP chat in python. (2019, September 24). NeuralNine. Retrieved March 21, 2022, from <https://www.neuralnine.com/tcp-chat-in-python/>