A CLEINT-SERVER CHAT PROGRAM

August 6, 2019

Jacques Huysamen - 20023669
Gerrit Burger - 21261687
Computer Science 354 - Networking

Contents

1 Project Description

This project required that we create a basic chat program based on the server - multiple client model. We had to use sockets in order to let clients connect to a server and easily communicate with each other. Threading was also required to update the GUI while messages are being sent and received all at the same time and so that the server can connect multiple clients at the same time and not freeze while waiting for clients to connect.

For the client we had to create a simple GUI where the following was a requirement:
* A panel where users can see messages being broadcast from other clients and also private messages coming from other users.
* A panel which dynamically updates when users connect or disconnect.
* A panel in which to type the message which a client wants to send.
* A quit button to disconnect a client from the server.

2 Project Features

2.1 Server Features

Features included in the Server:

- Multiple clients can connect at the same time.
- Server operations are printed out by the Server.
- Connections / Disconnections without interruptions.
- No duplicate names allowed.
- List of users updated when someone connects or disconnects.

2.2 Client Features

Features included in the Client:

- GUI that users can interact with.
- Popup on connect to ask for the userâĂŹs username.
- A list of online users that updates automatically when a new user connects or if a user disconnects.
- When a user joins or leaves the chat it is displayed.
- Private messages between users.
- Clients can send and receive at the same time.
- Disconnection without disrupting the Server.

2.3 Extra Features

Error handling for the following:

- Whispering to a non existent client (Automatic since you can only select a client from the list of online users.)
- Trying to connect to a non-existent server.

3 Algorithms and Data Structures

3.1 Algorithms

We did not use any algorithms in this project

3.2 Data Structures

3.2.1 Server Data Structures

On the server side of the program we used a global dictionary to store the information of the connected clients. The client's username was used as the key and the client's socket object as the value. By using this data structure we could easily see if a new connected user's provided username was already in use or not by looping through the dictionary and also easily get a client's socket object if a message has to be sent. The python socket package that we used stores the client's information in two objects when a new connection is created between the server and the client, we stored these objects in the above mentioned dictionary. We used the python pickle package to serialize and de-serialize data to send it as bytes over the socket connection. When as message had to be sent an array was created containing the data, the array was serialized with pickle and the array was sent over the socket connection. When the array was then received on the client side, it was de-serialized with pickle and the data could be used.

3.2.2 Client Data Structures

For the client we chose to use the python library PySide2 from the Qt software development toolkit for the front-end development. PySide allowed us to create QWidgets which act as Frames when ran. We run a client with our setup.py locally and then other users on other computers can run a run_client.py to run a GUI, both of these programs have the UI_Frame object which acts as our main window for our GUI which contains an area for showing messages, showing online users, typing messages and buttons to interact with the server. Both programs also have a pop-up on start to ask the user for their username, the only difference being that the run_client.py GUI also shows a pop-up where the user has to specify the IP and port of the server they want to connect to.

4 Experiments

During the development of the project we identified a number of experiments that we used to test our different features and their functionality. We tried to identify edge cases where our program could possibly fail and used these experiments to test them and find solutions.

4.1 We used the following experiments:

4.1.1 Duplicate client usernames

When a client connects to the server a pop up will be displayed where the client will input his or her username. When the username has been entered into the GUI the username will be sent from the client to the server as a string. The server will receive this string and check if the username is in the dictionary, if the username is already in the dictionary there is currently a user online with that username. The server will then send a message back to the client telling the client that the username is taken and the client will be prompted to enter a new username via the GUI. When the new username is then entered it will again be sent to the server to be validated, and if the username is then valid the server will send a message back to the client to tell the client it is a valid username. The valid username will then be added to the dictionary in the server, we used the username as the key and the clients address as the value in the dictionary. We stored these objects in the above mentioned dictionary.

As seen from the images below our program does not allow a client to connect if there is currently a user online with the username that the client entered, thus the experiment was a success and showed us that our program functions as intended.

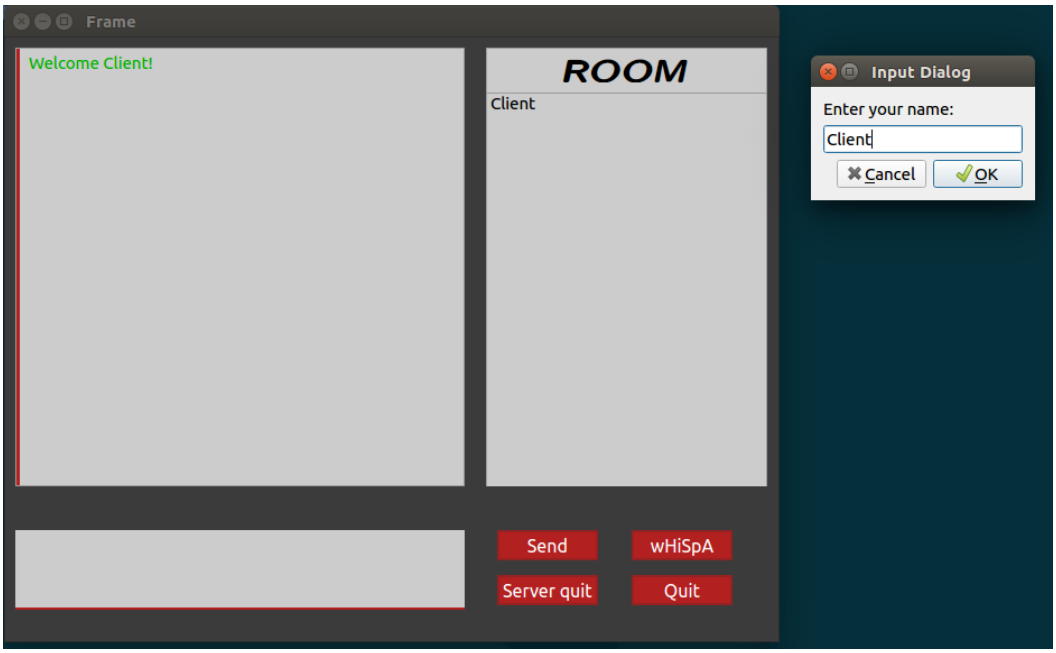

Figure 4.1: Second client trying to connect

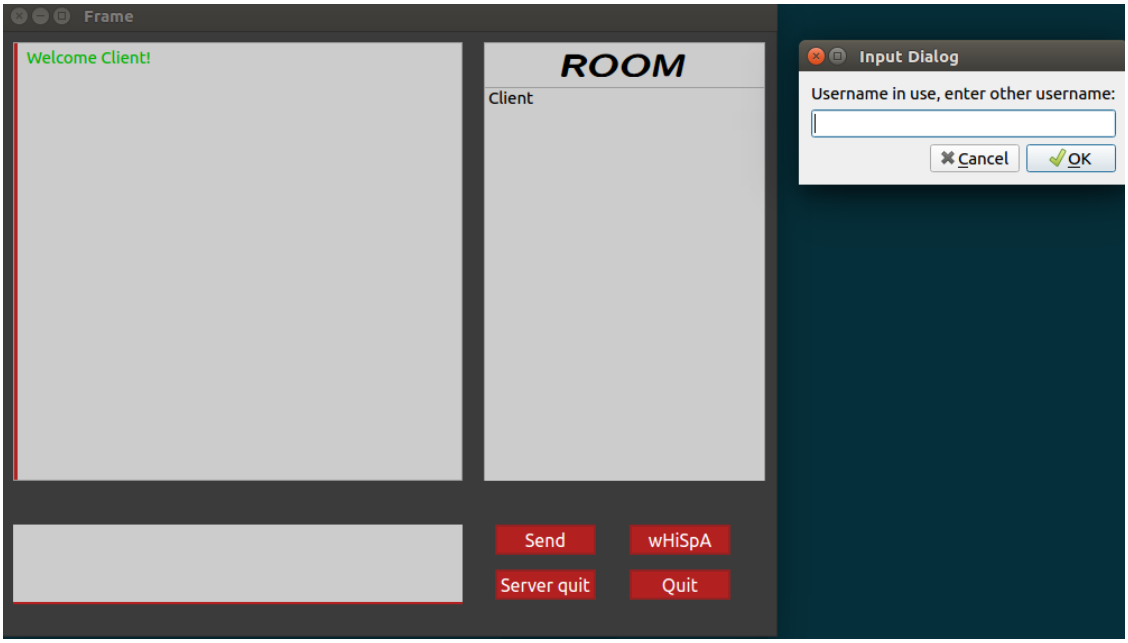

Figure 4.2: Second client prompted that username is taken

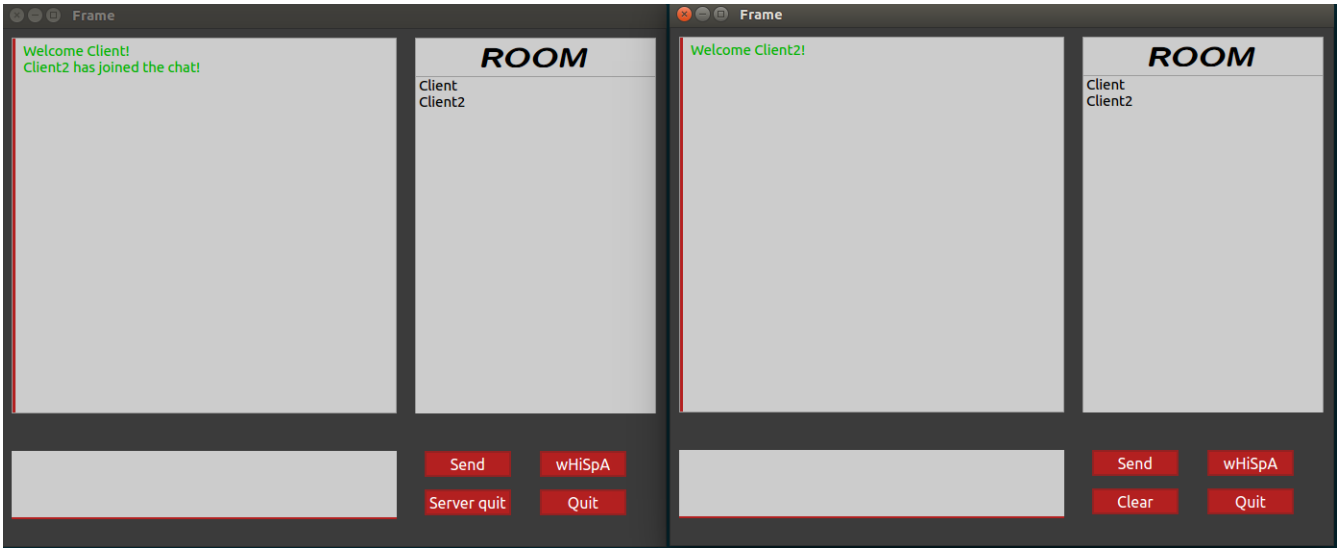

Figure 4.3: Second client connected with valid username

4.1.2 Client trying to connect to a non-existent Server

When a client starts a pop-up will show asking the user to enter the IP and port of the server they want to connect to. Given a wrong IP or port the client would not be able to connect to the server and an error would be thrown, we countered this error by displaying the message "FAILED TO CONNECT" in the client terminal and displayed the same pop-up until the correct IP and port was given. The user could then successfully connect to the server and continue as per usual. We set the port to a default value of 5000.

```
21261687@localhost:~/Documents/CS354/BraamfonteinPeanutButter$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr d8:9e:f3:86:10:09
          inet addr:146.232.50.156  Bcast:146.232.51.255  Mask:255.255.252.0
          inet6 addr: fe80::7183:526b:81bd:704c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7111669 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5984526 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:7004421627 (7.0 GB)  TX bytes:7203786790 (7.2 GB)
          Interrupt:20 Memory:f7100000-f7120000
```

Figure 4.4: Here we see the IP is: 146.232.50.156

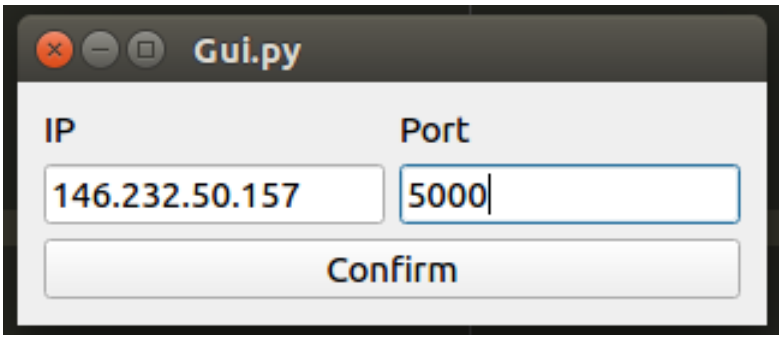

Figure 4.5: User trying to connect to the IP 146.232.50.157:5000

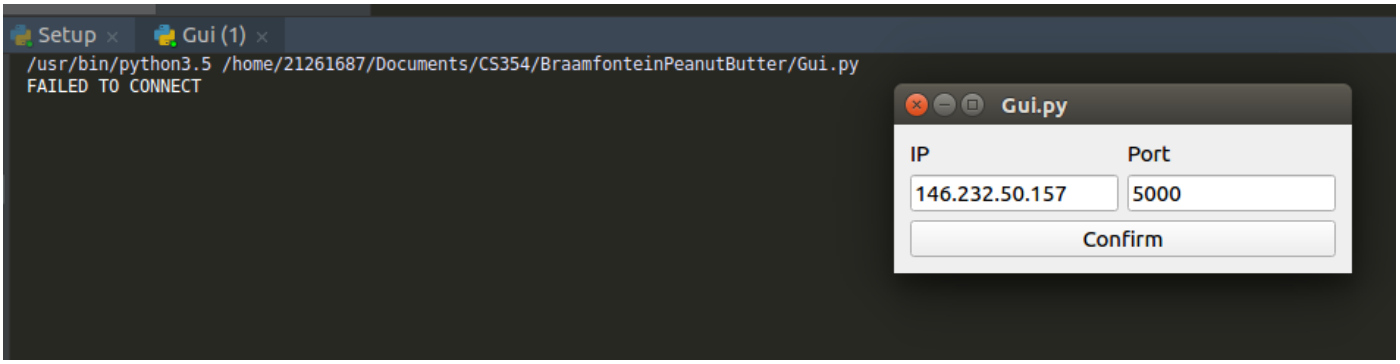

Figure 4.6: User trying to connect to the IP 146.232.50.157:5000

As seen from the above images the experiment successfully tested the functionality explained above.

4.1.3 Client disconnects without disrupting the Server

When a client disconnects from a server the client sends a message to the server which identifies the client who wants to disconnect and then deletes the user from the users dictionary, the server then closes the client's socket and then sends a message to the client to tell the application to close without giving an error or disrupting the server.

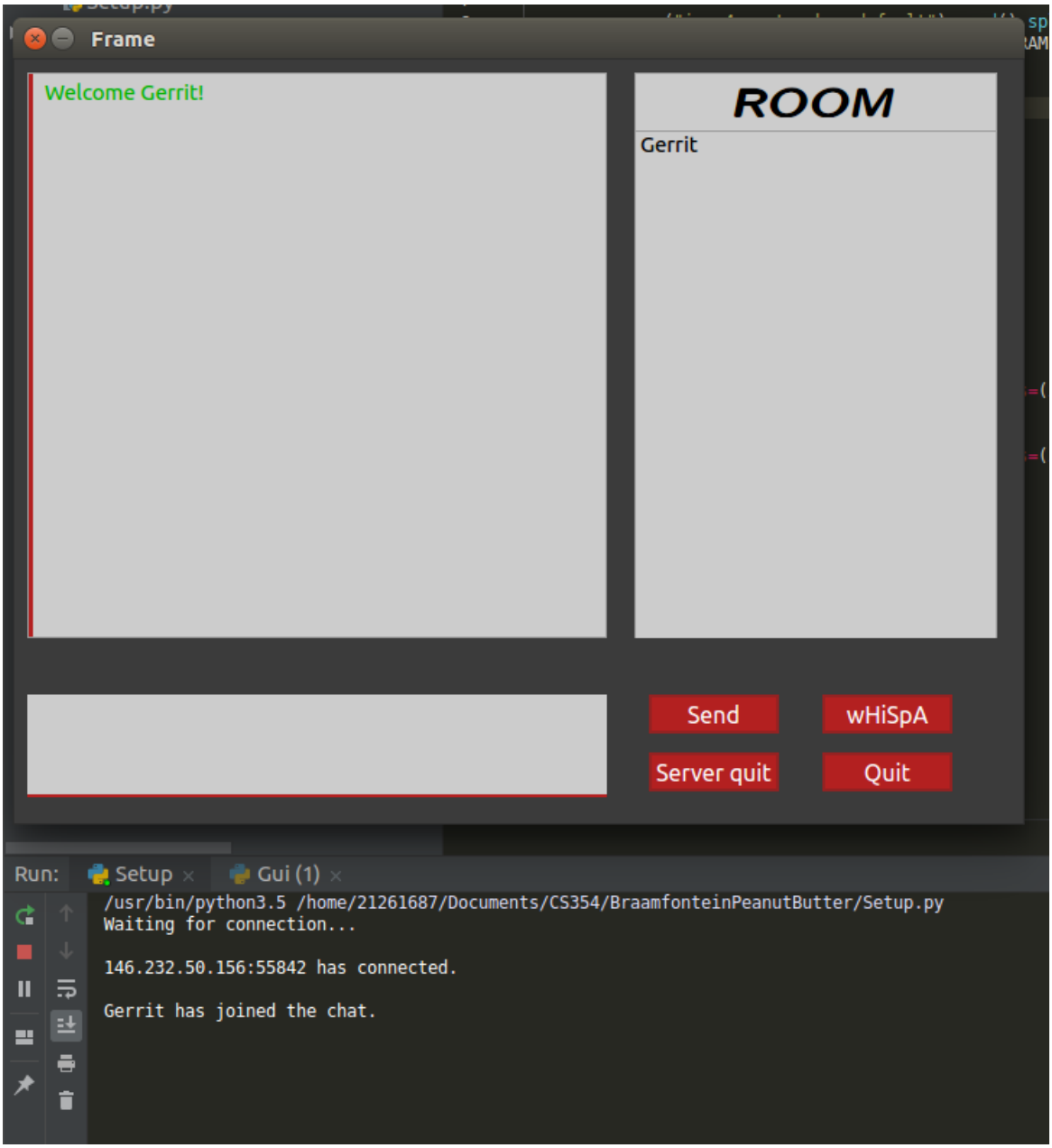

Figure 4.7: User connects to the server

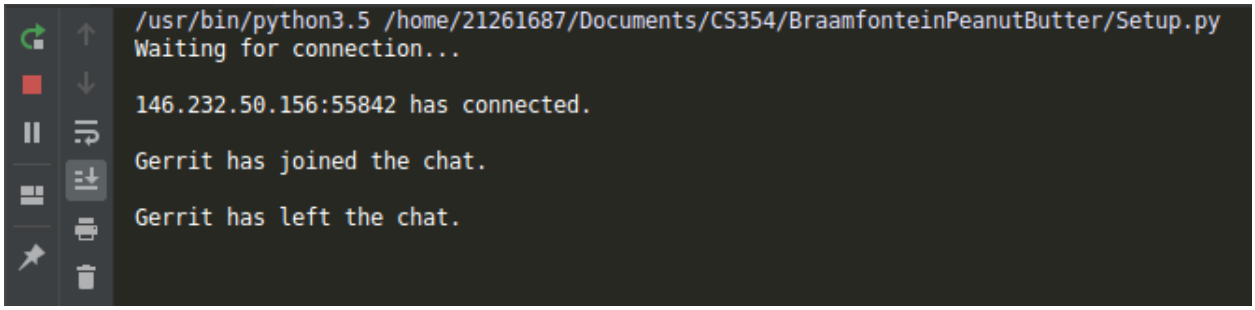

Figure 4.8: User disconnected

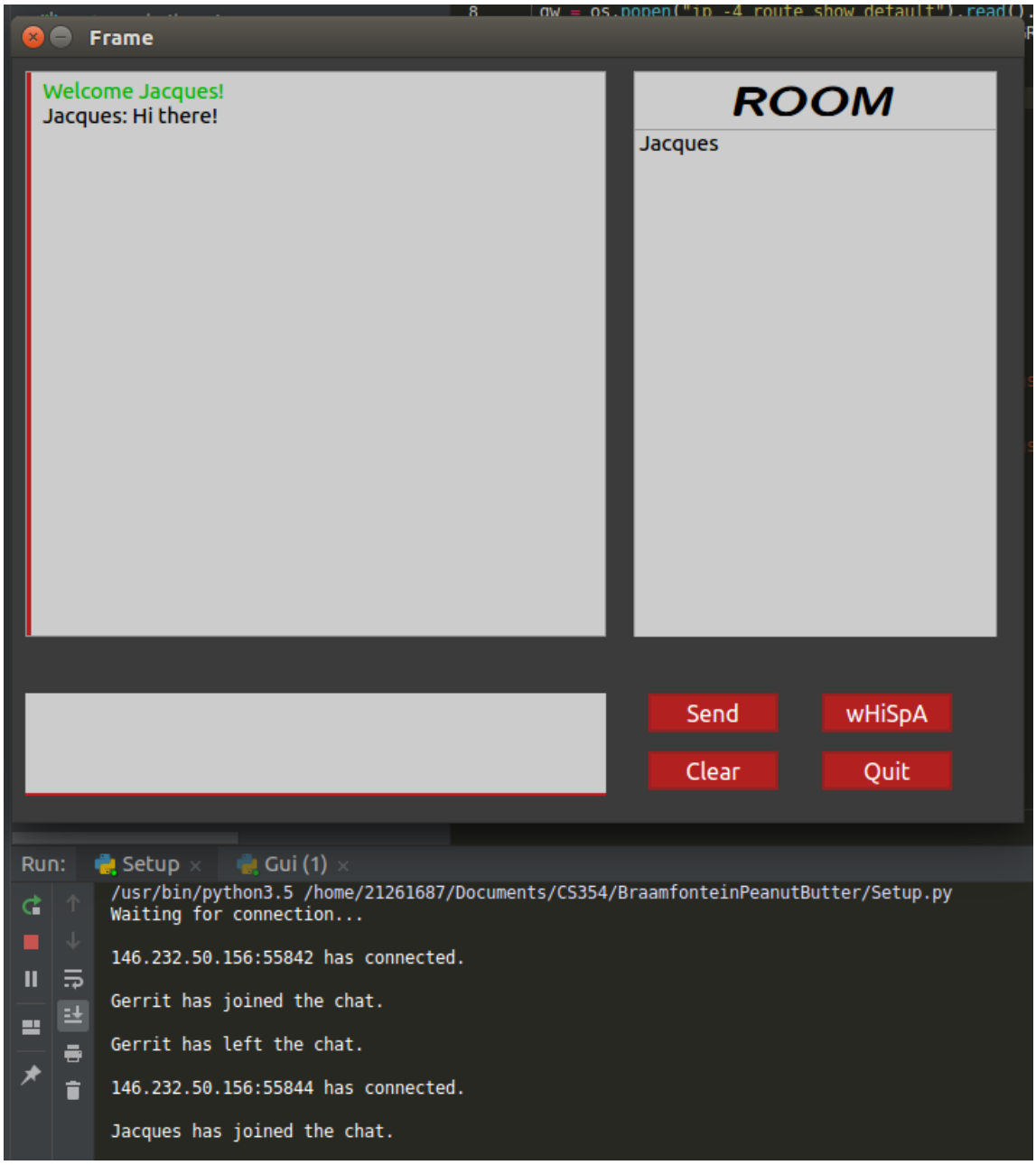

Figure 4.9: User connects after the previous user disconnected

Here the user Gerrit has disconnected from the server and there after the user Jacques connected and sent a message without any problems and thus we can see the users can disconnect without disrupting the server.

4.1.4 Multiple Clients trying to connect

For this experiment we connected 15 clients and sent messages to see if our server and clients were still fully functioning. We sent multiple messages from multiple clients and disconnected some clients to see if our messages still function between all of the other server traffic.

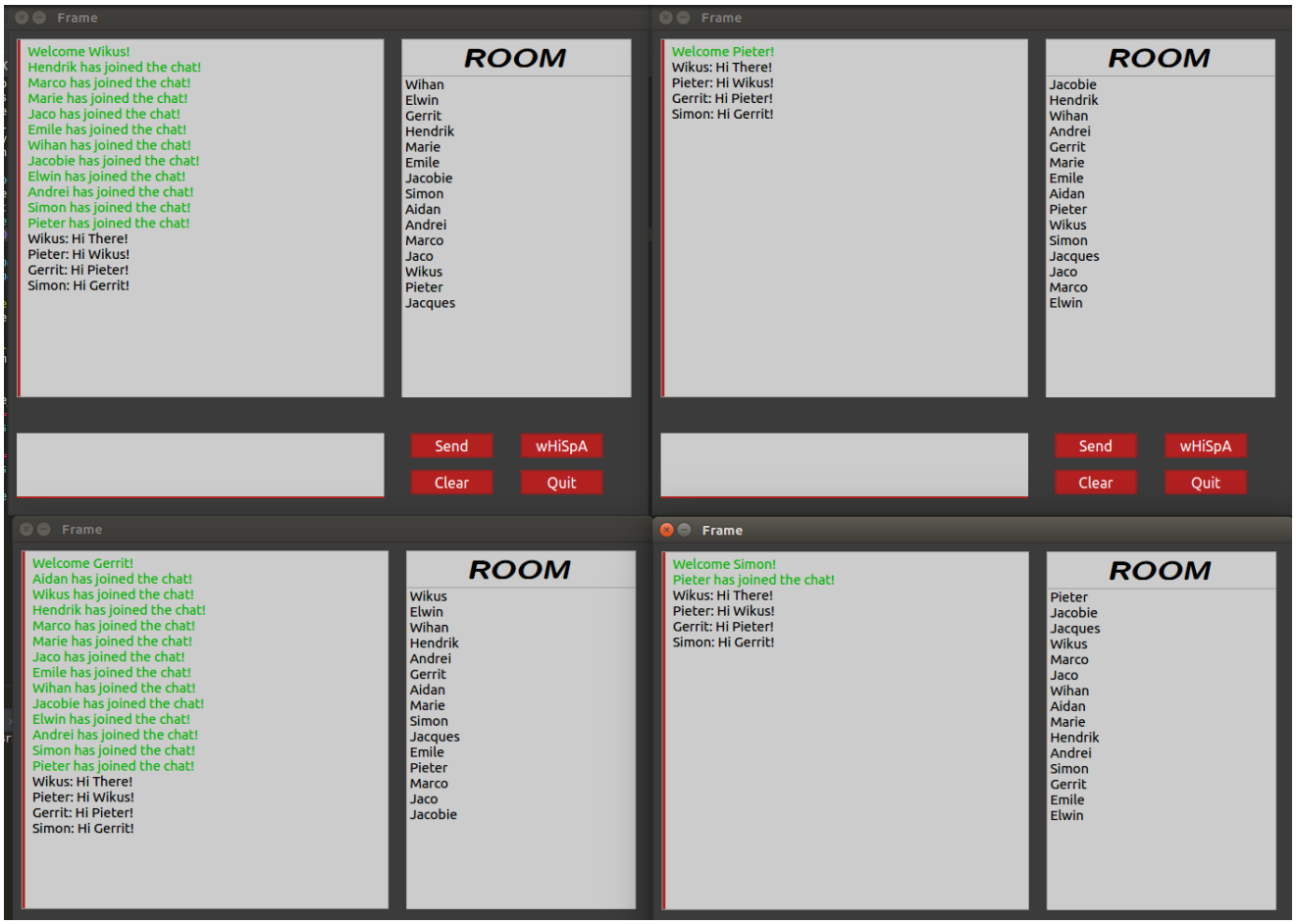

Figure 4.10: Multiple users connected and communicating

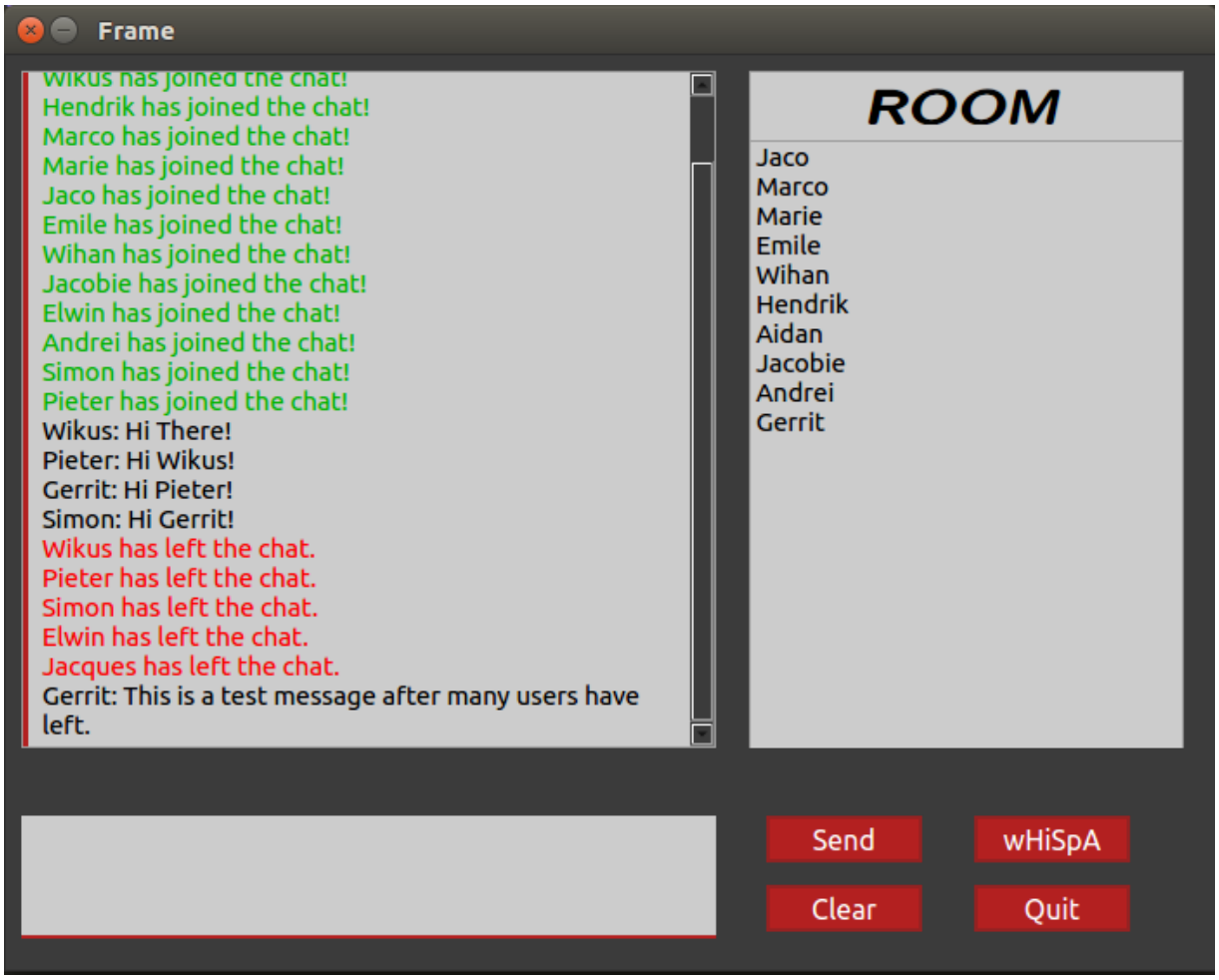

Figure 4.11: Messages after users disconnected and multiple still connected

4.1.5 Message handling after the Server terminates

After the server terminates the client application is still stable. In this experiment we closed the server and then continued to send messages and whispers to see what would happen. After sending the messages no errors were encountered and the client is presented with the message "Server disconnected."

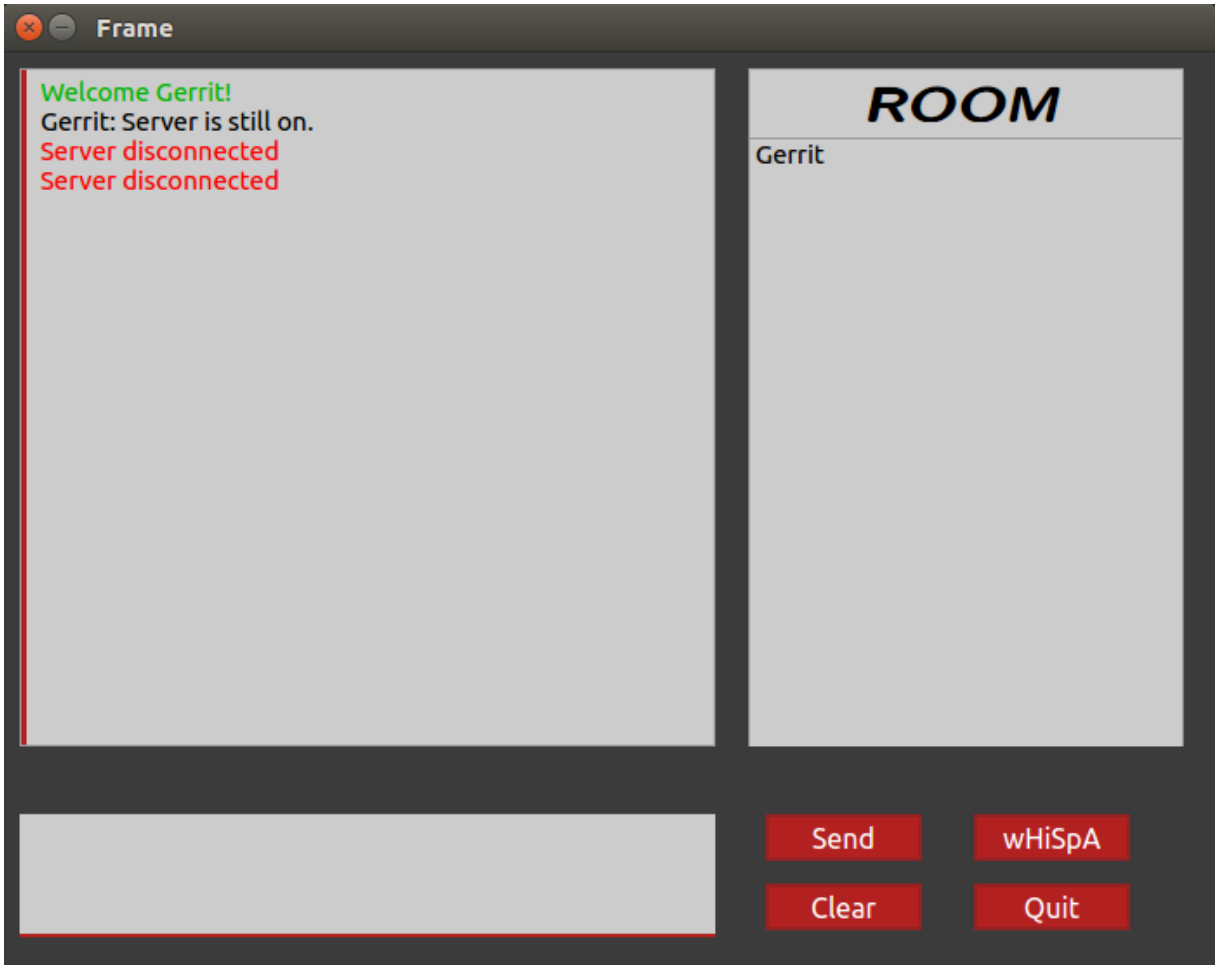

Figure 4.12: Message after server is off

4.1.6 Whispering to non-existent Clients

Users can only whisper to online users since a user has to select another user from the online users panel and then whisper to them by typing the message and clicking the whisper button.

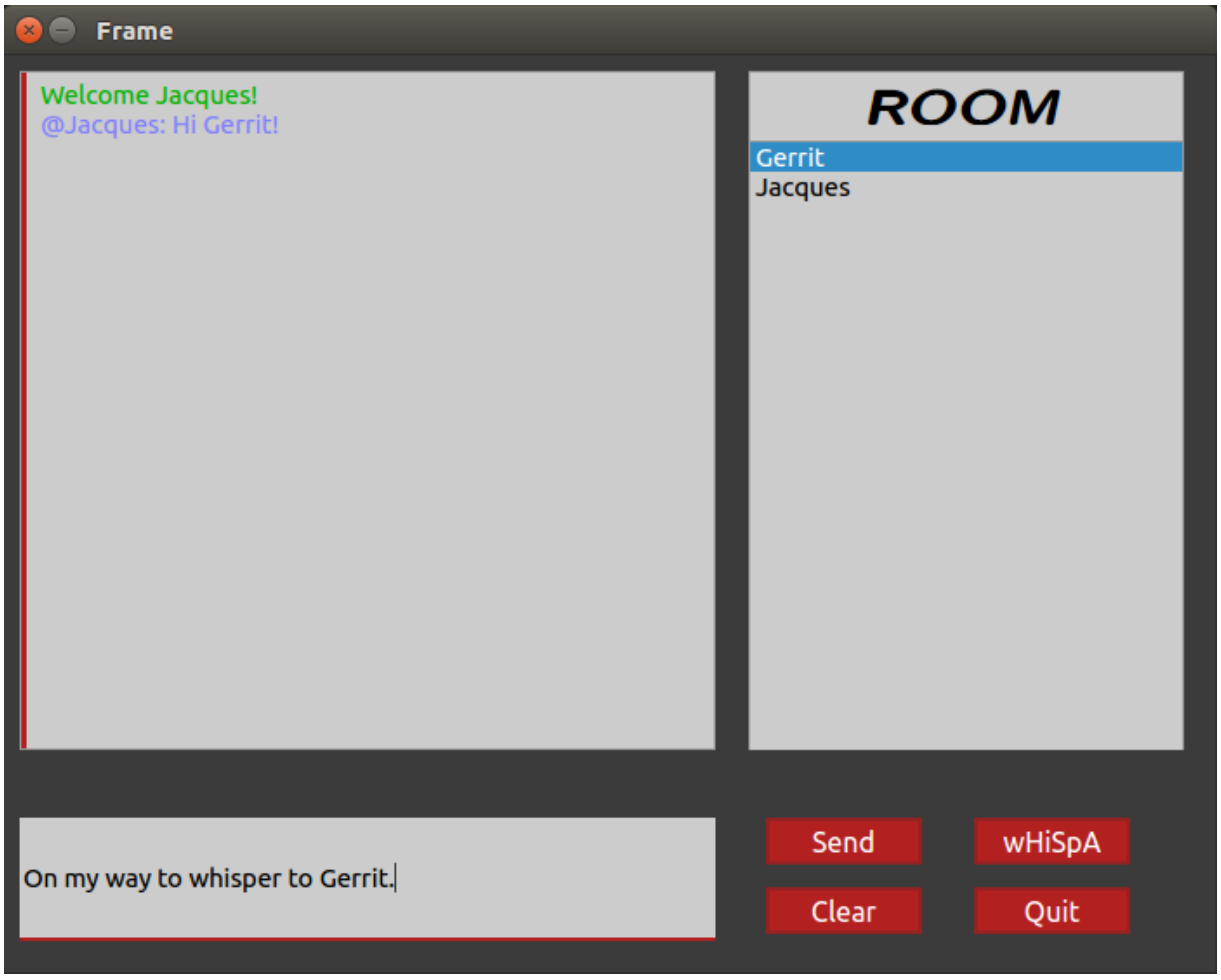

Figure 4.13: Whispering to another user

4.1.7 Receiving messages while typing

For this test we continuously sent each other messages while typing to each other to test whether we could successfully send and receive messages concurrently.

As seen from the image below clients can successfully receive and send messages concurrently.

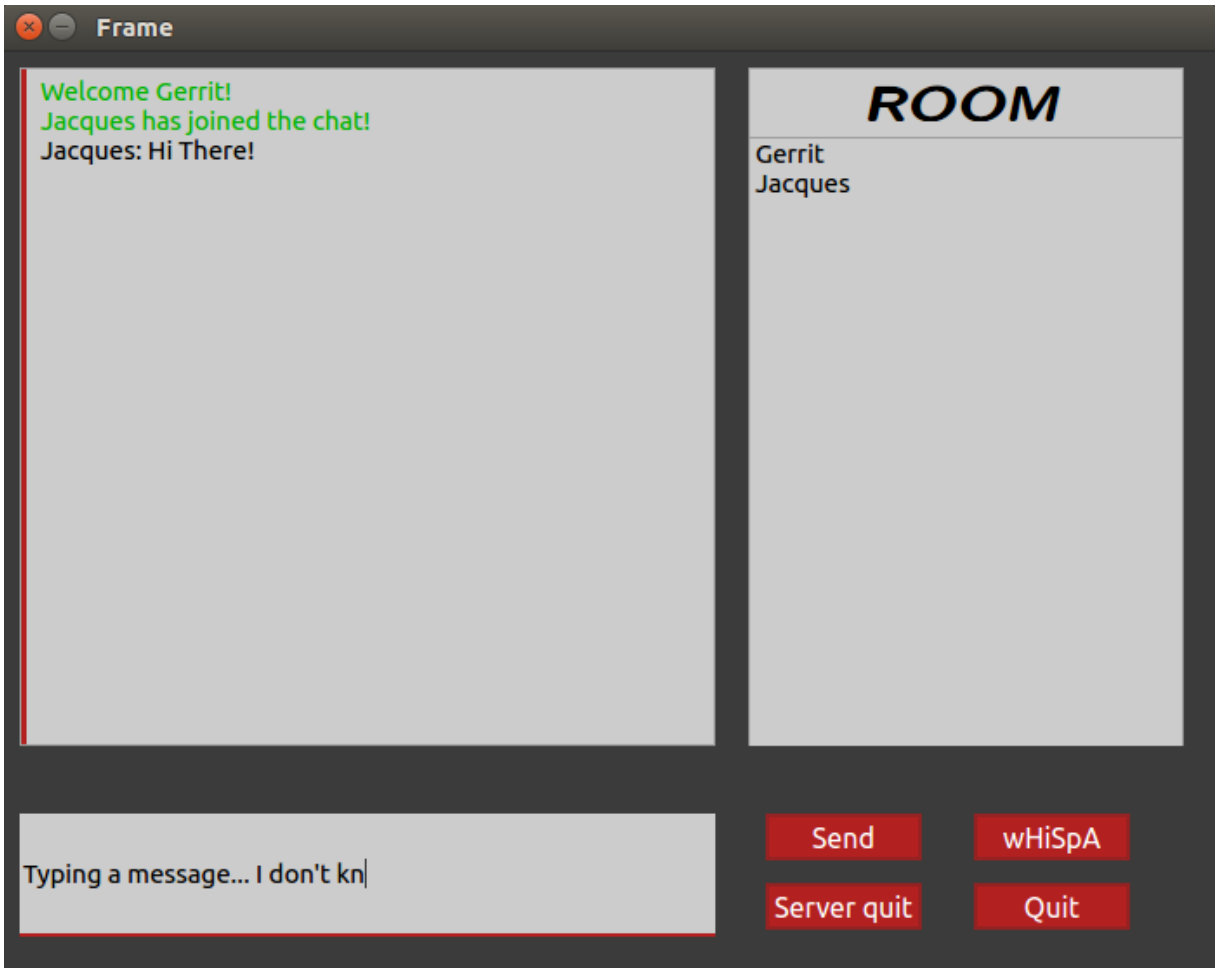

Figure 4.14: Receiving a whisper while typing a message

4.1.8 User-list updates with a lot of users connecting and disconnecting

In this experiment we connected 15 clients and checked to see that our user-list successfully updated each time a new client connected. We then disconnected some of the users and again checked that our list updated successfully.

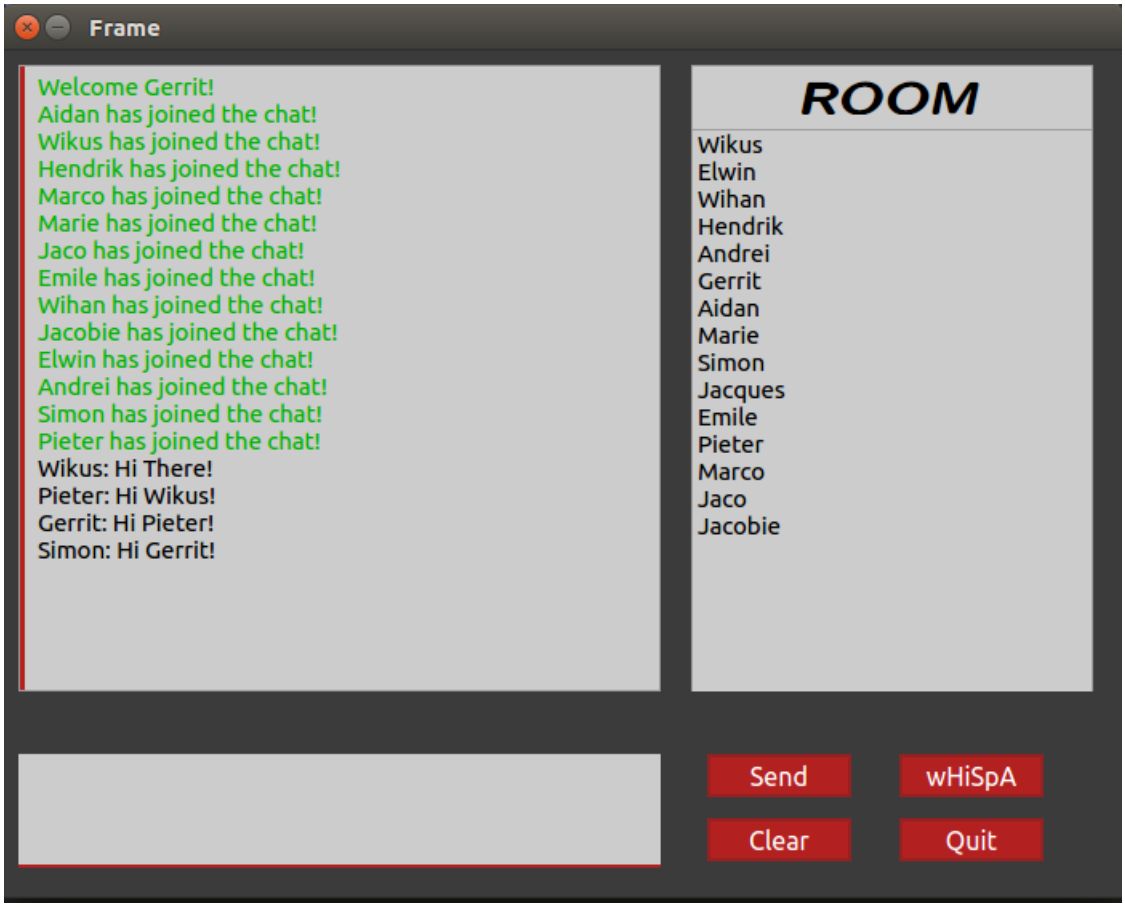

Figure 4.15: Many users connected

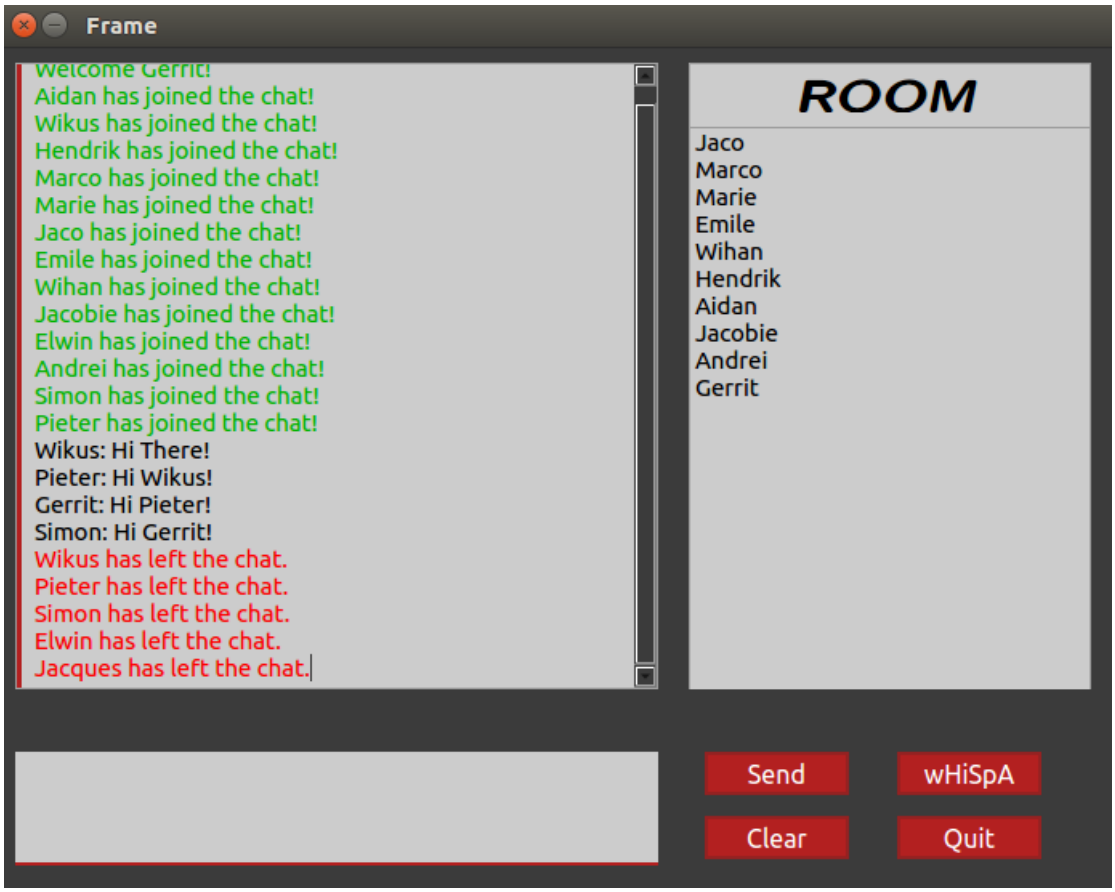

Figure 4.16: Multiple users leaving

5 Conclusion

After completion of the project we found that our program successfully passed all of our experiments and complied to all of the given requirements of the project. Our program can successfully create instances of servers and clients and create socket connections between clients and servers to send global messages and private whisper messages. We decided to use python because of its user-friendly data structures and extensive support libraries. We used the Python Socket library to handle our socket connections between clients and servers, and the PySide2 library to create our GUI. At the end of the project we can say that these libraries were easy to use and that the project was a success.