development documentation

Summary

Introduction

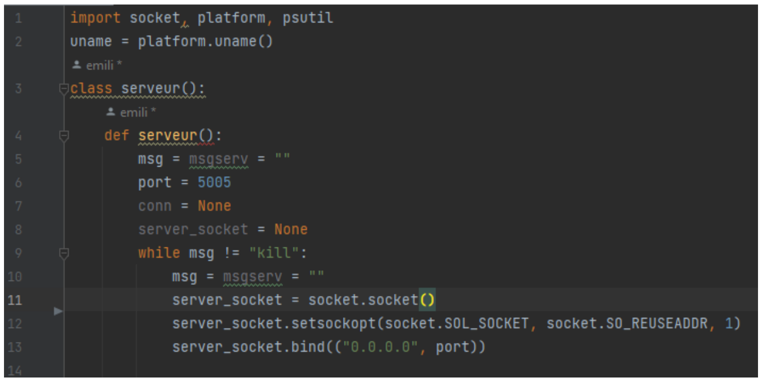
In this document you will found the explanation of the python files for the SAE 302.

For this project I only used 3 python files, one for the configuration of the server, one for the client configuration and the graphique interface, the last file complete the client file for the graphique interface.

To explain every file I will devide the codes in diffrent part and insert pictures of it.

First I will explain how the code of the server works then I will do the same thing with the code of the client and the graphic interface.

Serveur.py



```
1   import socket, platform, psutil
2   uname = platform.uname()
    emili *
3   class serveur():
        emili *
4       def serveur():
5           msg = msgserv = ""
6           port = 5005
7           conn = None
8           server_socket = None
9           while msg != "kill":
10              msg = msgserv = ""
11              server_socket = socket.socket()
12              server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
13              server_socket.bind(("0.0.0.0", port))
14
```

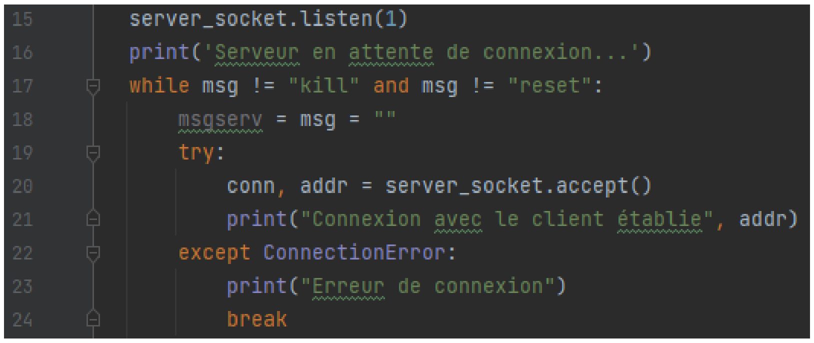
Figure 1 - Beginning of the code

Lines 1&2: Import of the needed modules, socket is usefull for the chat between the client and the server. platform and psutil helps to give information like the cpu.

I decide to made the server a fonction in a class to be proper and easier to module if I needed it.

Lines 5 to 8: I declare some variable that are needed like "msg" and "msgserv" that represent the messages of the client and the server. "Port" help for the connection.

From line 9 I told the server that while the message of the client isn't kill he needs to open the connection and wait for a client with any ip address and the port number 5005 (line 13).



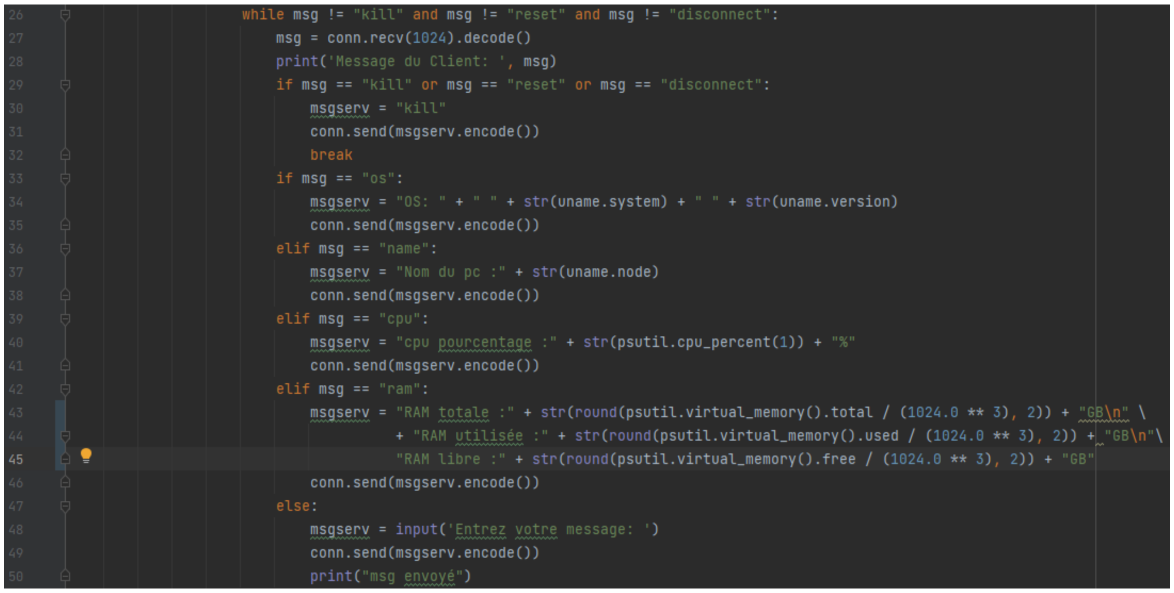
```
15    server_socket.listen(1)
16    print('Serveur en attente de connexion...')
17    while msg != "kill" and msg != "reset":
18        msgserv = msg = ""
19        try:
20            conn, addr = server_socket.accept()
21            print("Connexion avec le client établie", addr)
22        except ConnectionError:
23            print("Erreur de connexion")
24            break
```

Figure 2- Connection with the client

Lines 15&16: the server is waiting for the connection of the client and writes on his terminal that is waiting.

From line 17 I told the server that while the message of the client isn't kill or reset he needs to accept the connection of the client and writes that the connection is successful.

The " try , except" manage the possible error of connection.



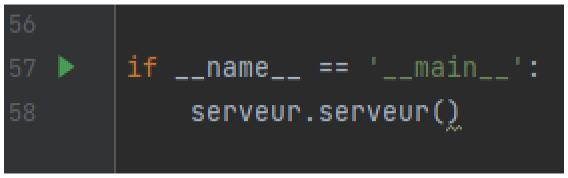
```
        while msg != "kill" and msg != "reset" and msg != "disconnect":
            msg = conn.recv(1024).decode()
            print('Message du Client: ', msg)
            if msg == "kill" or msg == "reset" or msg == "disconnect":
                msgserv = "kill"
                conn.send(msgserv.encode())
                break
            if msg == "os":
                msgserv = "OS: " + " " + str(uname.system) + " " + str(uname.version)
                conn.send(msgserv.encode())
            elif msg == "name":
                msgserv = "Nom du pc :" + str(uname.node)
                conn.send(msgserv.encode())
            elif msg == "cpu":
                msgserv = "cpu pourcentage :" + str(psutil.cpu_percent(1)) + "%"
                conn.send(msgserv.encode())
            elif msg == "ram":
                msgserv = "RAM totale :" + str(round(psutil.virtual_memory().total / (1024.0 ** 3), 2)) + "GB\n" \
                          + "RAM utilisée :" + str(round(psutil.virtual_memory().used / (1024.0 ** 3), 2)) + "GB\n"\
                          "RAM libre :" + str(round(psutil.virtual_memory().free / (1024.0 ** 3), 2)) + "GB"
                conn.send(msgserv.encode())
            else:
                msgserv = input('Entrez votre message: ')
                conn.send(msgserv.encode())
                print("msg envoyé")
```

Figure 3 - conversation with the client

While the message of the client isn't kill, reset or disconnect, the server needs to receive the message of the client, decode it and write it (line 27 and 28). At the three following lines, if the message of the client is kill, reset or disconnect the server have to send the message kill to the client.

After that if the message is os he need to send his own information about his os to the client and it's the same with the name of the server, the cpu and the ram. And if the message of the client is none of it, at line 47, the server ask you what message you want to send to the client and send it.

To finish the first while boucle at line 52 until lin 54 if the message of the client is kill the server close the connection then stop waiting for the client connection and write it in his terminal.

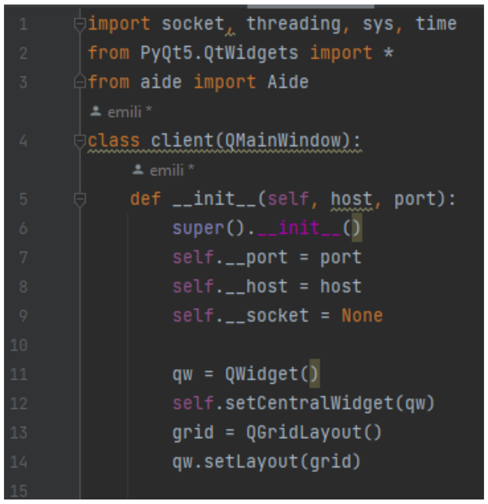


```python

if __name__ == '__main__':
    serveur.serveur()
```

Figure 4 - Running of the fonction

Here I ask the file to run the fonction server() of the class server.

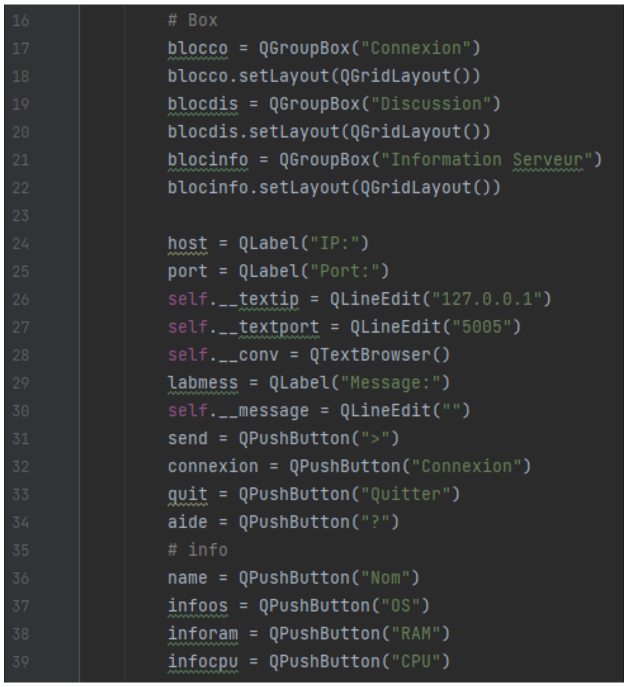
Client.py



```python
import socket, threading, sys, time
from PyQt5.QtWidgets import *
from aide import Aide

class client(QMainWindow):

    def __init__(self, host, port):
        super().__init__()
        self.__port = port
        self.__host = host
        self.__socket = None

        qw = QWidget()
        self.setCentralWidget(qw)
        grid = QGridLayout()
        qw.setLayout(grid)

```

Figure 5- Beginning of the client code

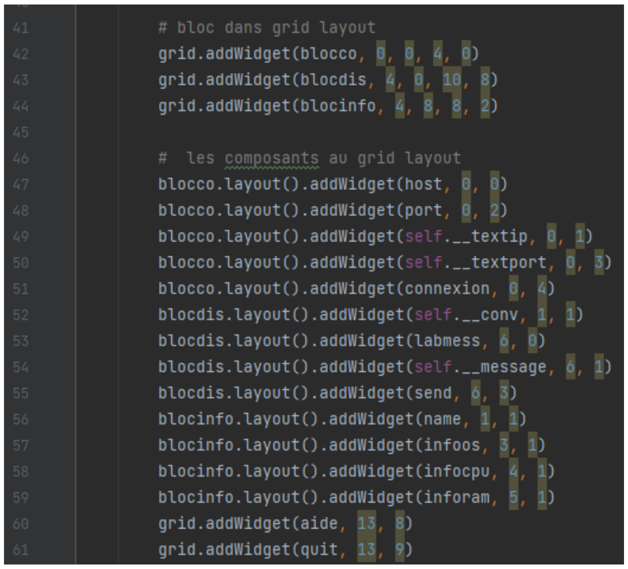
Here it is the same as the server code, I start by importing the modules I need then I made the client a class if the argument 'QMainWindow' for the graphic interface. After that I declare the variables that was needed for the client to work.



```python
        # Box
        blocco = QGroupBox("Connexion")
        blocco.setLayout(QGridLayout())
        blocdis = QGroupBox("Discussion")
        blocdis.setLayout(QGridLayout())
        blocinfo = QGroupBox("Information Serveur")
        blocinfo.setLayout(QGridLayout())

        host = QLabel("IP:")
        port = QLabel("Port:")
        self.__textip = QLineEdit("127.0.0.1")
        self.__textport = QLineEdit("5005")
        self.__conv = QTextBrowser()
        labmess = QLabel("Message:")
        self.__message = QLineEdit("")
        send = QPushButton(">")
        connexion = QPushButton("Connexion")
        quit = QPushButton("Quitter")
        aide = QPushButton("?")
        # info
        name = QPushButton("Nom")
        infoos = QPushButton("OS")
        inforam = QPushButton("RAM")
        infocpu = QPushButton("CPU")
```

Figure 6- Declaration of the interface's variables

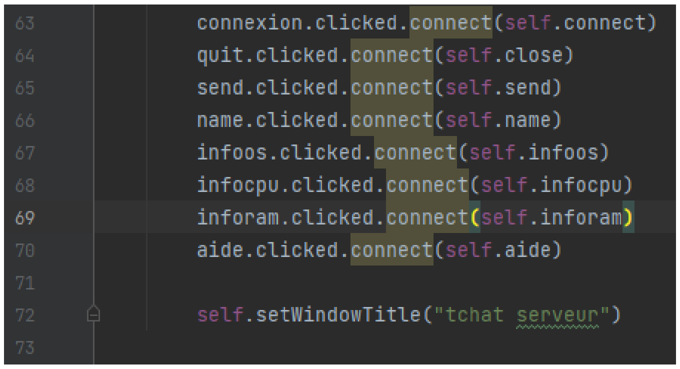
Here I declare all the variables for the interface starting by doing three blocs to organize the others variables.



```python
        # bloc dans grid layout
        grid.addWidget(blocco, 0, 0, 4, 0)
        grid.addWidget(blocdis, 4, 0, 10, 8)
        grid.addWidget(blocinfo, 4, 8, 8, 2)

        #  les composants au grid layout
        blocco.layout().addWidget(host, 0, 0)
        blocco.layout().addWidget(port, 0, 2)
        blocco.layout().addWidget(self.__textip, 0, 1)
        blocco.layout().addWidget(self.__textport, 0, 3)
        blocco.layout().addWidget(connexion, 0, 4)
        blocdis.layout().addWidget(self.__conv, 1, 1)
        blocdis.layout().addWidget(labmess, 6, 0)
        blocdis.layout().addWidget(self.__message, 6, 1)
        blocdis.layout().addWidget(send, 6, 3)
        blocinfo.layout().addWidget(name, 1, 1)
        blocinfo.layout().addWidget(infoos, 3, 1)
        blocinfo.layout().addWidget(infocpu, 4, 1)
        blocinfo.layout().addWidget(inforam, 5, 1)
        grid.addWidget(aide, 13, 8)
        grid.addWidget(quit, 13, 9)
```

Figure 7- Placement on the grid

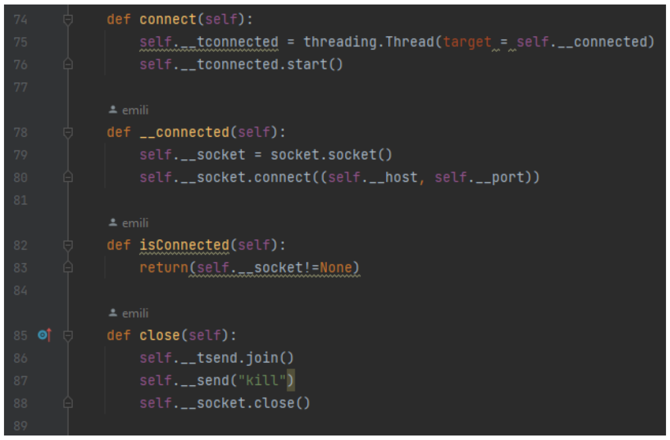
Again I start by placing the bloc on the interface then I place every element in a bloc and a specific place.



```python
        connexion.clicked.connect(self.connect)
        quit.clicked.connect(self.close)
        send.clicked.connect(self.send)
        name.clicked.connect(self.name)
        infoos.clicked.connect(self.infoos)
        infocpu.clicked.connect(self.infocpu)
        inforam.clicked.connect(self.inforam)
        aide.clicked.connect(self.aide)

        self.setWindowTitle("tchat serveur")
```

Figure 8- Bouton redirection

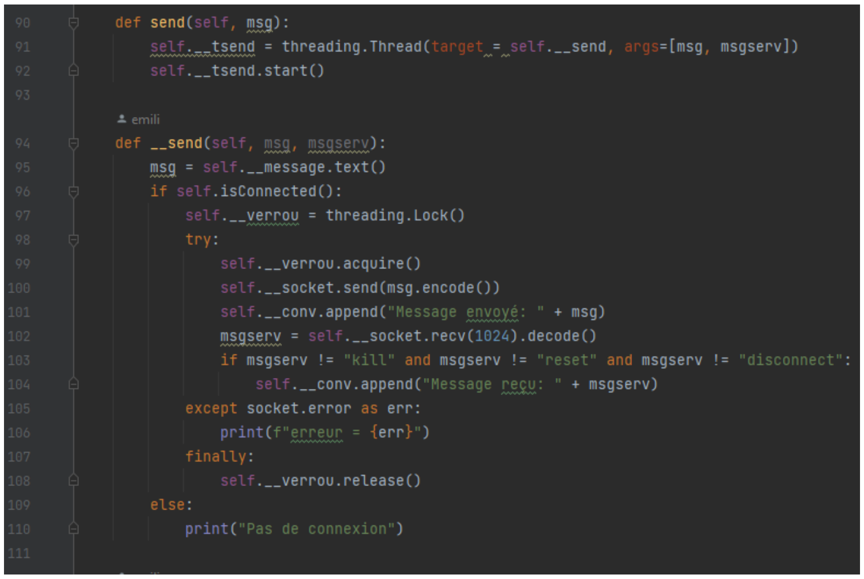
Here I connect every bouton to an action and I name the window "tchat serveur".



```python
    def connect(self):
        self.__tconnected = threading.Thread(target = self.__connected)
        self.__tconnected.start()

    def __connected(self):
        self.__socket = socket.socket()
        self.__socket.connect((self.__host, self.__port))

    def isConnected(self):
        return(self.__socket!=None)

    def close(self):
        self.__tsend.join()
        self.__send("kill")
        self.__socket.close()
```

Figure 10- First capture of function

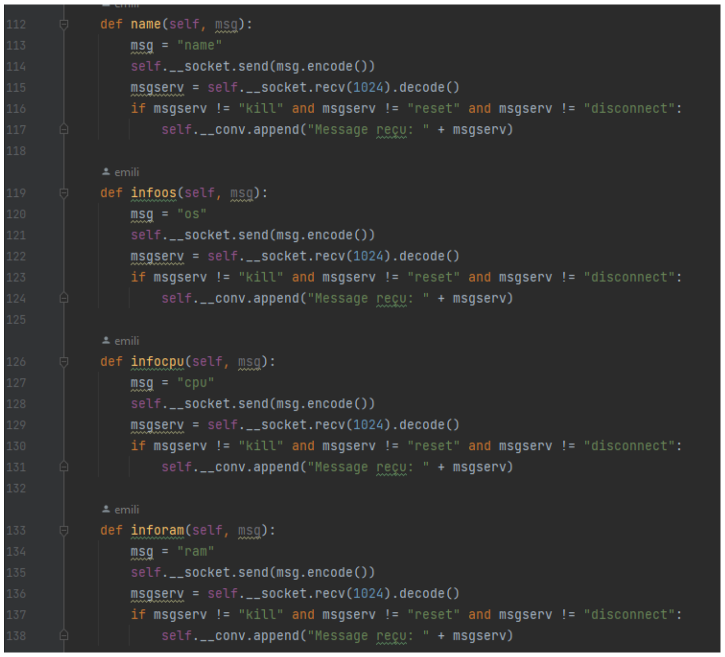
Connect is the thread (asynchrone) version of __connected. That is the fonction that connect the client to the server. isConnected is using for if boucle on the server. close is a function that close the connection.



```python
def send(self, msg):
    self.__tsend = threading.Thread(target = self.__send, args=[msg, msgserv])
    self.__tsend.start()

emili
def __send(self, msg, msgserv):
    msg = self.__message.text()
    if self.isConnected():
        self.__verrou = threading.Lock()
        try:
            self.__verrou.acquire()
            self.__socket.send(msg.encode())
            self.__conv.append("Message envoyé: " + msg)
            msgserv = self.__socket.recv(1024).decode()
            if msgserv != "kill" and msgserv != "reset" and msgserv != "disconnect":
                self.__conv.append("Message reçu: " + msgserv)
        except socket.error as err:
            print(f"erreur = {err}")
        finally:
            self.__verrou.release()
    else:
        print("Pas de connexion")
```

Figure 11- second capture of functions

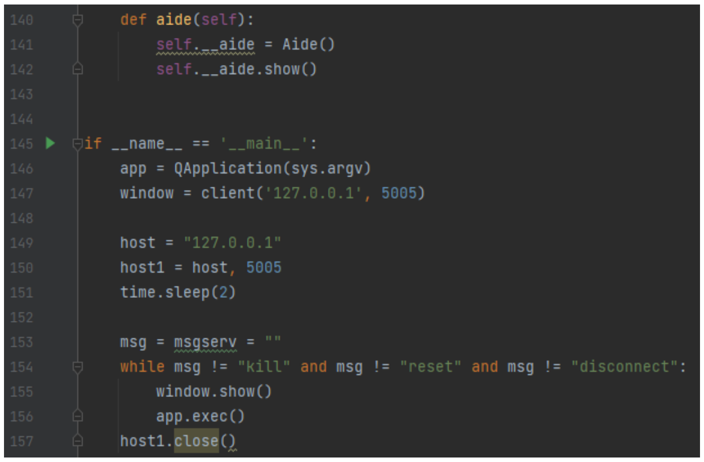
send is the threading version of __send that is the current functions that permit the discussion between the server and the client.



```python
def name(self, msg):
    msg = "name"
    self.__socket.send(msg.encode())
    msgserv = self.__socket.recv(1024).decode()
    if msgserv != "kill" and msgserv != "reset" and msgserv != "disconnect":
        self.__conv.append("Message reçu: " + msgserv)

emili
def infoos(self, msg):
    msg = "os"
    self.__socket.send(msg.encode())
    msgserv = self.__socket.recv(1024).decode()
    if msgserv != "kill" and msgserv != "reset" and msgserv != "disconnect":
        self.__conv.append("Message reçu: " + msgserv)

emili
def infocpu(self, msg):
    msg = "cpu"
    self.__socket.send(msg.encode())
    msgserv = self.__socket.recv(1024).decode()
    if msgserv != "kill" and msgserv != "reset" and msgserv != "disconnect":
        self.__conv.append("Message reçu: " + msgserv)

emili
def inforam(self, msg):
    msg = "ram"
    self.__socket.send(msg.encode())
    msgserv = self.__socket.recv(1024).decode()
    if msgserv != "kill" and msgserv != "reset" and msgserv != "disconnect":
        self.__conv.append("Message reçu: " + msgserv)
```

Figure 12- third capture of functions

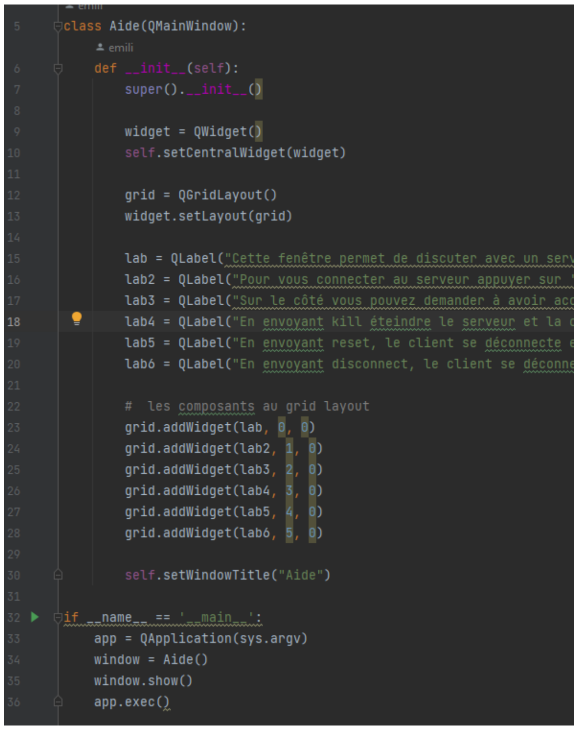
Here are all the functions for the server's informations that currently send a message to the server.



```
140     def aide(self):
141         self.__aide = Aide()
142         self.__aide.show()
143
144
145 if __name__ == '__main__':
146     app = QApplication(sys.argv)
147     window = client('127.0.0.1', 5005)
148
149     host = "127.0.0.1"
150     host1 = host, 5005
151     time.sleep(2)
152
153     msg = msgserv = ""
154     while msg != "kill" and msg != "reset" and msg != "disconnect":
155         window.show()
156         app.exec()
157     host1.close()
```

Figure 13 - last function and running of the file

The function aide return to the last file and show an other window. After the line 145 all the line permit the file to run.



```
class Aide(QMainWindow):
    def __init__(self):
        super().__init__()

        widget = QWidget()
        self.setCentralWidget(widget)

        grid = QGridLayout()
        widget.setLayout(grid)

        lab = QLabel("Cette fenêtre permet de discuter avec un serv
        lab2 = QLabel("Pour vous connecter au serveur appuyer sur
        lab3 = QLabel("Sur le côté vous pouvez demander à avoir acc
        lab4 = QLabel("En envoyant kill éteindre le serveur et la c
        lab5 = QLabel("En envoyant reset, le client se déconnecte e
        lab6 = QLabel("En envoyant disconnect, le client se déconne

        #  les composants au grid layout
        grid.addWidget(lab, 0, 0)
        grid.addWidget(lab2, 1, 0)
        grid.addWidget(lab3, 2, 0)
        grid.addWidget(lab4, 3, 0)
        grid.addWidget(lab5, 4, 0)
        grid.addWidget(lab6, 5, 0)

        self.setWindowTitle("Aide")

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = Aide()
    window.show()
    app.exec()
```

Figure 14- Aide.py

Here is the last python file that open a help window on the graphic interface.