Client.py:

```
import random, socket, struct, sys, time

class Client:
    interval_start = -1
    interval_end = -1
    guess_number = -1
    current_guess = -1
    current_response_of_server = -1
    last_try = False

    server_addr = 0
    packer = struct.Struct('ci')

    def __init__(self, interval_start, interval_end, host, port):
        try:
            self.server_addr = (host, int(port))

            if (interval_start > interval_end) | (interval_start < 0):
                raise Exception("The length of the interval cannot be less than 0!")

            self.__initialize_interval__(interval_start, interval_end)

            self.__TCP_Client__()
        except Exception:
            print("Something went wrong... :(")
            exit(-1)

    # Initializes the interval which contains the thought number
    def __initialize_interval__(self, interval_start, interval_end):
        self.interval_start = interval_start
        self.interval_end = interval_end

    def __making_a_guess__(self):
        # The guess will always be the floor of the middle number of the interval
        self.guess_number = ((self.interval_start + self.interval_end) // 2)

        # In case when client can only choose from one or two numbers, it must take an attempt
```

```
        # to guess the thought number
        if (abs(self.interval_end - self.interval_start) < 1):
            return ('='.encode(), self.guess_number)

        elif (abs(self.interval_end - self.interval_start) < 2):
            if (not self.last_try):
                self.last_try = True
                return ('>'.encode(), self.guess_number)
            else:
                return ('='.encode(), self.guess_number+1)

        # Python: 0 = False; 1 = True.
        # In this case, 2 means that client makes a guess.
        moreThanGuess = random.randint(0,1) # random.randint(0,2)
        # if (moreThanGuess == 2):
        #     return ('='.encode(), self.guess_number) # case when client takes an attempt to
guess the number
        # elif (moreThanGuess):
        if (moreThanGuess):
            return ('>'.encode(), self.guess_number)
        else:
            return ('<'.encode(), self.guess_number)

    def __BinarySearch__(self, server_answer):
        # Examining the server's answer by the client's question
        # and then narrowing down the interval
        if (self.current_guess[0] == b'>'):
            if server_answer:
                self.interval_start = self.guess_number
            else:
                self.interval_end = self.guess_number
        else:
            if server_answer:
                self.interval_end = self.guess_number
            else:
                self.interval_start = self.guess_number

    def __TCP_Client__(self):
```

```
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client:
            client.connect(self.server_addr)

            while True:
                time.sleep(5)
                self.current_guess = self.__making_a_guess__()
                packed_data = self.packer.pack(*self.current_guess)

                print(f"I'm sending this to the server: {self.current_guess}")

                client.send(packed_data)

                data = client.recv(200)
                unpacked_data = self.packer.unpack(data)
                self.current_response_of_server = unpacked_data

                print(f"I got this from the server: {self.current_response_of_server}")

                pred = self.__interprete_server_response__(client)
                self.__BinarySearch__(pred)

                print(f"My interval is: [{self.interval_start},{self.interval_end}]\n\n")


    def __interprete_server_response__(self, client):
        if (self.current_response_of_server[0] in [b'Y', b'K', b'V'] ):
            # Killing Client
            print("The game is over...")
            print("I'm leaving...")
            exit(0)
        return self.current_response_of_server[0] == b'I'


if len(sys.argv) != 3:
    print("Not enough/too much argument")
    exit(-1)
else:
    c = Client(1, 100, sys.argv[1], sys.argv[2])
```

Server.py:

```
import sys, random, socket, select, struct

class Server:
    interval_start = -1
    interval_end = -1
    thought_number = -1
    number_guessed = False
    clients_count = -1

    packer = struct.Struct('ci')
    server_addr = 0

    # Constructor: sets up the interval, thinks a number and sets up the server
    def __init__(self, interval_start, interval_end, host, port):
        try:
            self.server_addr = (host, int(port))
            self.__initialize_interval__(interval_start, interval_end)
            self.__think_a_number__()
            self.__Setup_Server__()
        except Exception:
            print("Something went wrong... :(")
            exit(-1)

    def __Setup_Server__(self):
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
            server.bind(self.server_addr)
            server.listen(4)
            print("Listening for new connections...")

            inputs = [server]
            timeout = 10

            try:
                while True:
```

```
                    readable, writeable, exceptional = select.select(inputs, inputs, inputs,
timeout)

                    if (self.clients_count == 0):
                        print("Every client's left. I'm leaving too...")
                        exit(0)

                    if not (readable or writeable or exceptional):
                        print("Data is not Readable/Writeable or the waiting time reached the
time out...")
                        exit(-1)

                    for sock in readable:
                        if sock is server: # new connection
                            client, client_addr = sock.accept()
                            inputs.append(client)
                            print("Connected: ", client_addr)

                            # Checking the number of the clients
                            self.clients_count = 1 if (self.clients_count == -1) else
self.clients_count + 1

                        else: # an existing connection is readable
                            data = sock.recv(self.packer.size)
                            if not data:
                                print("Logout: ", sock)
                                inputs.remove(sock)
                                sock.close()
                                self.clients_count -= 1
                            else:
                                # if someone has found the number, the game is over...
                                if (self.number_guessed):
                                    server_response = ('V'.encode(), self.thought_number)
                                    packer_data = self.packer.pack(*server_response)
                                    sock.send(packer_data)
                                    pass

                                # Receiving and sending back message to the client
```

```
                                client_response = self.packer.unpack(data)
                                print(f"\nI received this from a client: {client_response}")
                                server_response = self.__check_client_guess__(client_response)
                                print(f"I'm sending this to the client: {server_response}\n\n")
                                packer_data = self.packer.pack(*server_response)
                                sock.send(packer_data)

            except KeyboardInterrupt:
                print("Closing the server...")


    # Initializes the interval which contains the thought number
    def __initialize_interval__(self, interval_start, interval_end):
        self.interval_start = interval_start
        self.interval_end = interval_end

    # Generates a random number in the CLOSED interval of [self.interval_start,
self.interval_end]
    def __think_a_number__(self):
        self.thought_number = random.randint(self.interval_start, self.interval_end)

    # Checks the client's guess and responses if it is correct or not
    def __check_client_guess__(self, guess):

        print(f"My thought number is: {self.thought_number}")
        if guess[0] == b"=" : # case when a client guesses a number
            if guess[1] == self.thought_number:
                self.number_guessed = True
                return ('Y'.encode(), guess[1])

            else:
                return ('K'.encode(), guess[1])

        else: # case when a client asks a question to be decided
            if guess[0] == b">": # client asks if the thought number is more than the guess
                return ('I'.encode(), guess[1]) if guess[1] < self.thought_number else
('N'.encode(), guess[1])
            else: # client asks if the thought number is less than the guess
```

```
                return ('I'.encode(), guess[1]) if guess[1] > self.thought_number else
('N'.encode(), guess[1])



if len(sys.argv) != 3:
    print("Not enough/too much argument")
    exit(-1)
else:
    s = Server(1,100, sys.argv[1], sys.argv[2])
```