TCP Client-Server Game

Game

1. The game is a simple number guessing game.
2. User specifies the range of numbers.
3. The game generates a random number in this range.
4. User has 5 attempts to guess the number.
5. On each user's input game responds with a message.
6. When user guesses the right number or the number of attempts is exceeded, the game responds with "You win!" or "You lose" message respectively and the game ends.

Server

Server application can serve up to **two** clients at the same time.

- Server application has one command line argument:

    1. server port

  Example: `python ./server.py 65656`

  If the argument is not provided, the client outputs the message: `Usage example: python ./server.py <port>` and **exits**.

  The ip address is always `127.0.0.1`.

- At the start, server tries to bind to the provided port and localhost address.

    1. If it is not successfull, server outputs `Error while binding to the specified port` and **exits**.

    2. If it is successfull, server outputs `Starting the server on <ip>:<port>`

       Example: `Starting the server on 127.0.0.1:65656`

- Now server outputs `Waiting for a connection` message. and is waiting for incoming connections, until the `KeyboardInterrupt` is triggered.

- When server accepts an incoming connection from a client:

    1. If there is no room for a new client (two clients are already connected), server sends message to the client `"The server is full"` and closes the connection.
    2. Otherwise, outputs `Client connected` and continues.
    3. The server creates a new **thread** (not a process) in which the **game logic** will run.
    4. This thread binds to a new socket. IP address is the same.
    5. Server sends the port number of the new thread to the client and closes the connection.

- Server is waiting for a new client.

Game logic

1. Server is waiting for a client connection for 5s (ends if no connection)
2. Server sends a **Welcome message** to the client.
3. Server accepts a range of numbers from the client and generates a random number in this range (including boundaries).
4. Server starts the game and waits for client response (the guess).
5. Client has 5 attempts to guess the number.
6. If the client guesses the number, server sends a `You win!` message. The game ends.
7. If the limit of attempts is exceeded, server sends a `You lose` message. The game ends.
8. If the guess is less than the number, server sends a `Greater` messsage.
9. If the guess is greater than the number, server sends a `Less` messsage.
10. Before every guess server sends the number of attempts remaining to the client in the form: `You have N attempts`, where N is the number of attempts remaining.
11. If the connection is lost during the game, game ends with no output from the server.

Welcome message is a message `Welcome to the number guessing game!\nEnter the range:`

Client

Client application for user interaction.

- Client application has two command line arguments:

    1. server ip address
    2. server port

    Example: `python ./client.py 127.0.0.1 65656`

    If at least one argument is not provided, the client outputs the message: `Usage example: python ./client.py <address> <port>` and **exits**.

- **Step 1**. At the start, client connects to the server on provided address and port using **TCP socket**.

    If server is unavailable (not running or there is no connection), client outputs `Server is unavailable` and **exits**.

- After connecting to the server, client waits to the server's respond with the new port number, and the connection is closed by the server.

- If client recieves a **Server is full** message, it **exits** with the corresponding output.

- **Step 2** Client connects to a new port of the server and the game starts.

- Client waits for a **welcome message** from the server.

- Client sends a range of numbers specified by the user (two positive integers devided by space, where the second one is greater). Example: `0 42`

    If range input is incorrect, client prints `Enter the range:` again and waits for the input.

- **Step 3** Client waits for the message with left amount of attempts.

- Client sends a guess number
- Client waits for the message with correctness of guess. Possible messages: `Less`, `Greater`, `You win!`, `You lose`
- If the message was `Less` or `Greater`, the game continues from **Step 3**
- If the message was `You win!` or `You lose`, the game ends.

If the connection was lost during any step of client execution, client outputs `Connection lost` and **exits**.

If there is the `KeyboardInterrupt` during any step of client execution, client must **exit**

Important information

1. Both server and client must use TCP sockets.
2. The protocol (message format) for communications between client and server is not defined, so it is up to you, but (2)
3. The input and output of the program **must strictly** correspond to that specified in this file.
4. Your submission should contain at least two files: `server.py` and `client.py`. You may submit other files in addition to those if your implementation requires it.
5. If you want to submit an archive, please submit it in a `.zip` format

Server example

```
> python ./server.py 65656
Starting the server on 127.0.0.1:65656
Waiting for a connection
Client connected
Waiting for a connection
Client connected
Waiting for a connection
The server is full
Waiting for a connection
```

Client example

```
> python ./client.py 127.0.0.1 65656
Welcome to the number guessing game!
Enter the range:
> 0 42
You have 5 attempts
> 5
Greater
You have 4 attempts
```

```
> 40
Less
You have 3 attempts
> 24
Greater
You have 2 attempts
> 33
You win!
```