Project 1: Chat Room

EGP405: Networking for Online Games (Fall 2017)

Assigned: **Thursday, Sept. 7**

Due date: **Thursday, Sept. 21**

Grade weight: **10%, out of 20 points**

Introduction:

In recent classes, we have explored RakNet's packet preparation and used it to send greetings between multiple instances of a networked application. In this project, we will build upon this experience by implementing a console-based chat room application.

Goals:

This project requires you to do the following:

1. Design a networked chat room system using RakNet's peer-to-peer architecture.
2. Implement the system in a console application.
3. Justify design choices using diagrams and descriptions.

Instructions:

Design and implement your chat room with the following major and minor features:

1. **Lobby state**: On start-up or leaving the chat room, choose to start a server or client. This is a different "state" from the chat room itself, pre- or post-connection.
2. **Chat room client state**: Implement a message loop for a chat room "participant" with the following specifications:
    a. Client peer: the peer cannot receive connections, and can only connect to a chat room server using a known IP address and port.
    b. Sends messages to server only with participant information prepended.
    c. Receives pertinent messages from server.
    d. Can send and receive public messages (broadcast to all via server) or private messages (for a specific participant).
3. **Chat room server state**: Implement a message loop for the chat room's "administrator" with the following specifications:
    a. Server-on-top-of-client peer: the peer receives connections.
    b. Sets up properties and rules for chat room.
    c. Maintains user names and IP addresses of multiple connected users.
    d. Relays and redirects all messages and events from all participants.
    e. Broadcast messages to all or selected clients, public or private (e.g. "JimBob123 has joined the chat room."; "hi JimBob123").
    f. Displays all inbound and outbound messages, public or private.
    g. Print user names and IP addresses of all connected users to the server console or to a file with a time stamp.

4. **User interface**:
    a. Console clearly displays keyboard commands or shortcuts for user to do different tasks (e.g. send public message, send private message, join chat); commands should do exactly what they say they do.
    b. Console should refresh as needed; e.g., it does not need to show the entire message history, but this may be an option.

Additional Requirements:
Complete the project in teams of ***up to three members*** with only one submission required per team. You must complete an independent peer review (see Canvas assignment) to receive a grade for your contributions. Be sure to ***test*** your project on multiple computers simultaneously before submitting.
Note: Expectations are proportional to team size!

Your team must also submit a ***brief justification document*** (1 – 2 pages) that outlines the system design. Provide a UML diagram of data structures used, and a diagram of the chat system's architecture (i.e. logical flow). This is not an essay or full design document; its purpose is to help me navigate your code. Provide team member contributions and point-form descriptions of where to find your features in the code.

To achieve a perfect score on this project, consider incorporating ***delighters*** in your system design. Implement at least one complementary feature not stated as a core requirement as above, and justify it in terms of the system design. This is the creative aspect of the project, so do things that are fun and unique.

Submission Guidelines:
1) Include the following header information at the top of all source files ***modified***:
    a. "This file was modified by <*names*> with permission from author."
2) Include the following header information at the top of all source files ***created***:
    a. Team member names and student IDs.
    b. Course code, section, project name and date.
    c. Certificate of Authenticity (standard practice): "*We certify that this work is entirely our own. The assessor of this project may reproduce this project and provide copies to other academic staff, and/or communicate a copy of this project to a plagiarism-checking service, which may retain a copy of the project on its database*."
3) All sections of code modified or written by your team should be commented, explicitly stating ***who*** made the change or wrote code and ***why***, and what the code ***does*** in the context of the program.

4) ***When you are finished***, delete all garbage files. ***Do not*** delete Visual Studio project or solution files, or your source files. ***Do not*** submit developer SDKs with your project (I gave them to you; I already have them). **Delete the following files**:
    a. ***All*** build and intermediate files (Debug, Release).
    b. Intellisense database files (including but not limited to *.db, *.sdf).
5) Zip up your project's root and rename using this convention: "**EGP-405-F2017-Net-Project1-<insert submitter's name here>**".
6) In the Canvas submission, add:
    a. Your design justification document as a PDF file.
    b. The above-mentioned zip file.
    Note: If the file size is larger than 1MB, you have not cleaned it correctly!
    c. A link to your public repository.
    Note: Grades will not be discussed or released through public repositories, for any reason, in compliance with FERPA.

Grading:

Grading for this project is broken into categories, with specific objective expectations listed in the table/rubric below.

Points **Category**	**0** **N/A**	**1**	**2** **Fair**	**3**	**4** **Meets Expectations**	**5** **Exceeds Expectations**
Functionality	Application does not serve the purpose or goals of the project.		Application serves its purpose with some bugs or issues.		Application serves its purpose with no bugs.	Application serves its purpose with "contingency plans" for user errors.
Features	Several required features not implemented.		Some of the required features implemented.		Most of the core features implemented; one or more delighters.	All required features and multiple delighters implemented.
Architecture	Code is not coherent, organized or modular.		Code has some organization and modularity.		Code is clean, coherent and organized into functions and modules where applicable.	Project makes use of appropriate file structure with full modularity.
Design	Design justification not provided.		Diagrams and/or descriptions provided but do not fully supplement the system implemented.		Diagrams and descriptions provided; adequately summarize features.	Diagrams and descriptions adequately summarize features, and help with navigating the code.

Specific penalties:

- **–10 points**: Submission does not build and run *correctly* and *immediately* out of the box (test using both debug and release modes).
- **–10 points**: Code not documented and commented, including contributions, explanations, certificate of authenticity, etc.
- **–10 points**: Lacking or weak evidence of having used version control to maintain project. Please ask for help if needed.
- **–10 points**: Submission contains junk files. Follow instructions above to ensure you have removed the correct files, and ask for help if needed.

Good luck, learn lots and have fun! ☺