Gareth Lewis
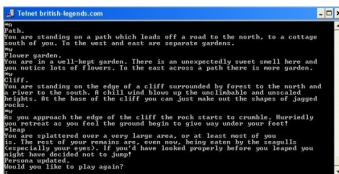
*"Essentially everyone, when they first build a distributed application, makes the following eight assumptions:*

*- The network is reliable;*

*- Latency is zero;*

*- Bandwidth is infinite;*

*- The network is secure;*

*- Topology doesn't change;*

*- There is one administrator;*

*- Transport cost is zero;*

*- The network is homogeneous.*

*All prove to be false in the long run and all cause big trouble and painful learning experiences."*

*— Peter Deutsch*



*MUD1*, written by Roy Trubshaw and Richard Bartle—now, Prof. Bartle—was the first virtual world to run multiplayer over a network. It was written in 1978, pre-dating the Internet.

## Introduction

For this assignment, you will implement the client/server MUD application that you designed in assignment 1.

This assignment is formed of several parts:
- (A) **Implement** a MUD prototype as client and server applications that will:
  - i.    **Support** multiple clients using socket-based networking
  - ii.   **Incorporate** distributed processing using threads for both client and server applications
  - iii.  Be **realised** as fault-tolerant client-server architecture.
  - iv.   Server is **implemented** in Python
  - v.    Client is **implemented** in Python

- (B) Implement **persistent** data by maintaining player data in a relational database (sqlite), such that:
  - i.    The MUD will support named players
  - ii.   Player data (location, status etc) can be saved between sessions

The resulting client-server application will need to be capable dealing with the following use-cases :
1. When launched, server will not fail if no clients are running / available
2. When launched, client(s) will not fail if server is not running / available
3. A client can connect to server without failure of client, currently connected clients or server
4. A client can disconnect from server without failure to client, currently connected clients or server
5. Server can support multiple clients
6. A player using a client to interact with the game world will not adversely impact other clients or the server
7. When a player enters a room that other players are in, all players in that room will be made aware of the player's entry
8. When a player leaves a room that other players are in, all players still in that room will be made aware of the player's exit
9. A player can communicate with other players in the same room, but not in the entire dungeon

## Assignment Setup

This assignment is a programming task. Fork the GitHub repositories at:

https://github.com/Falmouth-Games-Academy/comp260-client
https://github.com/Falmouth-Games-Academy/comp260-server

Ensure that you maintain the readme.md file. Modify the .gitignore to the defaults for the programming languages and integrated development environments you will be using, as appropriate.

## Part A

To complete Part A, refactor the existing SUD and Chat Service applications, as per your assignment 1 design, to create the MUD service.

## Part B

To complete Part B, refactor your MUD service such that player related data, referred to `currentRoom` in Dungeon.py, is managed through an sqlite database.

Take your client and server applications, zip them up and submit them to Learning Space. Then arrange to have a meeting with the module leader to demonstrate your MUD service. You will receive informal feedback during the review, and you will receive formal feedback for the assignment within three weeks.

## Additional Guidance

A common pitfall is poor planning or time management. Often, students underestimate how much work is involved in first learning programming concepts and then actually applying them. Programming is quite unlike other subjects in that it cannot be crammed into a last minute deluge just before a deadline. It is, therefore, very important that you begin work early and sustain a consistent pace: little and often.

It is very easy for the deadline for this assignment to sneak up on students. Although a MUD looks quite simplistic in comparison to the games you have been making using game engines, it requires a considerable effort to get an application operating robustly over a computer network.
Be wary! Start early, and dive into research on these topics.

On a related note, you **MUST** incorporate a database into your solution. This is an absolute requirement. If you didn't master the material on SQLite and mySQL databases that were introduced to you in the first year, then you should revisit these topics. A great resource to do so is W3Schools:

https://www.w3schools.com/sql/default.asp

Likewise, the material on UNIX-based servers and commands will not be formally covered again. Here is a resource to remind yourself:

http://mally.stanford.edu/~sr/computing/basic-unix.html

Consider carefully how the MUD service should cope with session and persistent data. Persistent data will likely be user and player-centric, like

username, password, experience, room, inventory, and so on. Session data is likely to be things like current chat log. Things that developers would expect to be discarded when the player finishes playing.

**FAQ**

- **What is the deadline for this assignment?**
  Falmouth University policy states that deadlines must only be specified on the MyFalmouth system.

- **What should I do to seek help?**
  You can email your tutor for informal clarifications. For informal feedback, make a pull request on GitHub.

- **Is this a mistake?**
  If you have discovered an issue with the brief itself, the source files are available at:

  https://github.com/Falmouth-Games-Academy/bsc-assignment-briefs.

  Please raise an issue and comment accordingly.

## Additional Resources

Additional resources have been migrated to the Talis Aspire system, which is available at:

http://resourcelists.falmouth.ac.uk/.

# Marking Rubric

| Criterion | Weight | Refer for Resubmission | Basic Proficiency | Novice Competency | Novice Proficiency | Professional Competency | Professional Proficiency |
|---|---|---|---|---|---|---|---|
| Threshold | 40% | At least one part is missing or is unsatisfactory. | Submission is timely. Enough work is available to hold a meaningful discussion. Clear evidence of programming knowledge and communication skills. Clear evidence of use of appropriate version control techniques, including regular commits and some use of branching. No breaches of academic integrity. Server has been developed in Python | | | | |
| MUD Implementation | 20% | Absence of threading<br><br>Client has not been developed in Python<br><br>Server has not been developed in Python | Class hierarchy bears some relationship to UML<br><br>Functionality bears some relationship to UML<br><br>Demo successfully handles a reasonable set of use-cases | | | | |
| Remote Service | 20% | Client and server cannot communicate | Client/server operation is solid and stable with few, if any issues. | | | | |
| Data Persistence | 20% | Server does not use relational database to manage persistent game data or client data<br><br>On returning to the game, players restart at the 'beginning' of the dungeon rather than where they last were | Clear evidence that data is stored in server-side SQL database | | | | |