CLIENT-SERVER GAME TASK

Version 2.0.1
BSc Computing for Games
COMP260

Dr Michael Scott

Introduction

"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

For this assignment, you implement a multiplayer online text-based adventure game using a client-server architecture. The server will be written in C#. Though, you may implement the client in a suitable language of your choice.

Knowledge of computer networking is essential. You will need to work with highly sophisticated network technologies to satisfy increasing demand for social experiences in games. Experience working with client-server architectures, in particular, is highly desired by employers even outside games.

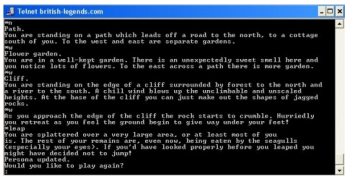
This assignment is formed of several parts:

(A) **Extend** your client-server MUD such that:
  i. the server **operates** from a DigitalOcean Ubuntu Droplet;
  ii. and **supports** multiple simultaneous clients;
  iii. clients **operate** from local machines as a thin clients;
  iv. the system **incorporates** network communication between a local client **and** remote server over the Internet;
  v. the system and network communication is **secured and encrypted**;
  vi. the server **manages** all major game content (e.g., room topologies);
  vii. while **leveraging** a distributed data infrastructure (e.g., for trade);
  viii. as well as player log in **and** session recovery functionality;
  ix. server data **is managed** within appropriate databases;
  x. and databases **discriminate** between session and persistent data.

(B) **Implement** a fnal prototype of the game that will:
  i. **respond** to security concerns raised in the security wiki;
  ii. and **revise** any other issues raised by your tutor and/or your peers.

(C) **Present** a live demonstration of the MUD that will:
  i. **show** your academic integrity **and** technical communication skills.

Please Note: Falmouth University Games Academy have partnered with DitigalOcean to provide **EACH** of you with your own remote server on their cloud platform. You will receive login details after completing the first coursework task. You **MUST** act in an ethical and professional manner when using this platform. If you encounter technical difficulties, please contact `games.techs@falmouth.ac.uk` in the first instance.



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.

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

Part A is a **single formative submission**. This work is **individual** and will be assessed on a **threshold** basis. The following criteria are used to determine a pass or fail:

(a) Submission is timely;
(b) Enough work is available to conduct a meaningful review;
(c) A broadly appropriate review of a peer's work is submitted.

To complete Part A, prepare draft versions of the server and client programs. Deploy these to the DigitalOcean server. Also ensure that the source code and related assets for both client and server are pushed to GitHub. These should be made available for review prior to the scheduled peer-review session. Then, attend the scheduled peer-review session.

Important: It is your individual responsibility to ensure the server code has been deployed and is running ahead of the peer review. Tinkering and last minute bug fixing should be done **before** the workshop. Though do not worry if, at this stage, your work is not feature complete and the game still suffers from a few bugs. This is an opportunity to get feedback and advice on your work-in-progress.

You will receive immediate **informal feedback** from your **peers**.

Part B

Part B is a **single summative submission**. This work is **individual** and will be assessed on a **criterion-referenced** basis. Please refer to the marking rubric at the end of this document for further detail.

To complete Part B, revise both the client and server programs based on the feedback you have received. Then, upload both to the LearningSpace. Please note, the LearningSpace will only accept a single `.zip` file.

You will receive **formal feedback** from your **tutor** three weeks after the final submission deadline.

Part C

Part C is a **single summative submission**. This work is **individual** and will be assessed on a **threshold** basis. The following criteria are used to determine a pass or fail:

(a) Enough work is available to hold a meaningful discussion;
(b) Clear evidence of programming knowledge **and** communication skills;
(c) No breaches of academic integrity.

To complete Part C, prepare a practical demonstration of the MUD. Ensure that the source code for both client and source code, in addition to any related assets, are pushed to GitHub and made available for review prior to the scheduled viva session. Then, attend the scheduled viva session.

Important: It is your individual responsibility to ensure the server is running and stable ahead of the viva. Tinkering and last minute bug fixing should be done **before** the alloted time-slot. No extra time will be given. You must also ensure that a network connection can be obtained using your computer. Test the wired and/or wireless network connection at the venue ahead of the viva.

You will receive immediate **informal feedback** from your **tutor**.

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. The live deployment, in this assignment, is an added dimension. Aim to complete your client and server a week early so you have sufficient time to troubleshoot your DigitalOcean instance, your server stability, and the network protocols you are using.

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.

Do not underestimate matters of security. You **MUST** consider the security of your solution and incorporate appropriate safeguards in order to ontain a passing grade on the distributed infrastructure criterion. Start the research journal early, and incorporate both your own findings as well as the findings of your peers into your work. Act as a research community and develop a discourse that raises awareness of key issues.

On a related note, you MUST also 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 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. Provided a meaningful review of a peer's work. 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.				
Functional Coherence	10%	Few or no working multiplayer features. The source code does not compile or there are serious bugs that prevent play.	Some multiplayer functionality. Player chat is working.	Modest multiplayer functionality. Login, authentication, and player chat are working. Only a modest number of obvious logical errors.	Much multiplayer functionality. Login, authentication, player chat, and session recovery are working. Only some obvious logical errors.	Considerable multiplayer functionality. Login, authentication, player chat, session recovery, and item gathering are working. Only a few obvious logical errors, which if present are largely cosmetic and/or superficial in nature.	Significant multiplayer functionality. Login, authentication, player chat, session recovery, item gathering, and trade are working. Almost no obvious logical errors, which if present are largely cosmetic and/or superficial in nature.
Distributed Systems Architecture	30%	Little to no merit to networking. Little to no ability to articulate concepts associated with computer networking and/or databases and/or client-server architecture.	Some merit to networking. Database incorporated into server. Some ability to articulate concepts associated with computer networking, databases, and client-server architecture.	Modest merit to networking. Databases sufficiently incorporated into both server and client. Security robust against naiive attacks. Some ability to articulate design decisions associated with concurrency and distribution.	Much merit to networking. Databases adequetely incorporated into both server and client. Security robust against basic attacks. Modest ability to articulate design decisions associated with concurrency and distribution.	Considerable merit to networking. Distributed data infrastructure is appropriate, leveraging well-known design patterns. Much ability to articulate design decisions associated with concurrency and distribution. Security robust against competent attacks.	Significantly merit to networking. Distributed data infrastructure is highly appropriate, leveraging well-known design patterns. Security robust against somewhat sophisticated attacks. Considerable ability to design decisions associated with concurrency and distribution.
Programming Quality	20%	There are no comments in the source code, or comments are misleading. Most variable names are unclear or inappropriate. Code formatting hinders readability. No insight into the appropriate use of programming constructs is evident from the source code. No attempt to structure the program (e.g. one monolithic function).	The source code is only sporadically commented, or comments are unclear. Some identifier names are unclear or inappropriate. Code formatting is inconsistent or does not aid readability. Some insight into the appropriate use of programming constructs is evident from the source code. The program structure is somewhat adequete.	The source code is somewhat well commented, with doc-strings. Some identifier names are descriptive and appropriate. An attempt has been made to adhere to an appropriate formatting style. There is little obvious duplication of code or of literal values. Modest insight into the appropriate use of programming constructs is evident from the source code. The program structure is adequate.	The source code is reasonably well commented, with doc-strings. Most identifier names are descriptive and appropriate. Most code adheres to an appropriate formatting style. There is almost no obvious duplication of code or of literal values. Much insight into the appropriate use of programming constructs is evident from the source code. The program structure is appropriate.	The source code is reasonably well commented, with doc-strings. Almost all identifier names are descriptive and appropriate. Almost all code adheres to an appropriate formatting style. There is no obvious duplication of code or of literal values. Some literal values can be easily "tinkered" in the source code. Considerable insight into the appropriate use of programming constructs is evident from the source code. The program structure is highly appropriate. There is high cohesion and low coupling.	The source code is very well commented, with doc-strings. All identifier names are descriptive and appropriate. All source code adheres to an appropriate formatting style. There is no obvious duplication of code or of literal values. Most literal values are, where appropriate, easily "tinkered" outside of the source code. Significant insight into the appropriate use of programming constructs is evident from the source code. The program structure is very effective. There is high cohesion and low coupling.