DISTRIBUTED PROCESSING TASK

Version 2.0
BSc Computing for Games
COMP260

Gareth Lewis

Introduction

In this assignment, you are required to design and implement algorithms that process data in a *distributed* manner by developing a prototype multiplayer dungeon, implemented as client and server applications in Python.

Games are resource intensive. Compounding this issue, players are sensitive to performance issues. It is critical, then, to leverage available resources to ensure adequate performance. Distributed processing is one solution. Apply the principles of coordination and agreement, and you will be successful.

"...a folk definition of insanity is to do the same thing over and over again and to expect the results to be different. By this definition, we in fact require that programmers of multithreaded systems be insane. Were they sane, they could not understand their programs."

— Edward A. Lee

This assignment is formed of several parts:

(A) **Design** a distributed processing architecture in UML for a MUD that will:

i. **Support** multiple client instances on a single computer
ii. **Enable** players to navigate multiple locations in a virtual dungeon
iii. **Allow** players to be aware of other players in the same location
iv. **Permit** players in the same room to communicate.
v. **Robustness** that allows the server to continue operation when a client is lost
vi. **Robustness** that allows a client to continue (limited) operation when server is lost
vii. **Create** a suitable wireframe mock-up of the client UI.
viii. **Use** appropriate UML techniques to capture:
   (a) The class hierarchy form of the client and server applications
   (b) The state-based function of the client and server applications
   (c) The data transmitted between client and server applications

"No one can write correct programs in a language where a=a+1 is not deterministic."

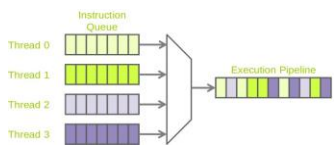
—Luiz Henrique de Figueiredo

"Frameworks don't solve scalability problems, design solves scalability problems."

— Ryan Tomayko

(B) **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



Multi-threading is commonly used to improve performance in games.

(C) **Implement** a more refined design and MUD prototype that will:

i. Revise any issues raised by your tutor and/or your peers.

(D) **Present** a practical demonstration of the MUD prototype that will:

i. Show academic integrity and technical communication skills.

cont...

The resulting client-server application will need to be capable dealing with the following use-cases which will be demonstrated during the peer review (part B) and the viva presentation (part D).

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 repository at:

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

Use the existing directory structure and, as required, extend this structure with sub-directories. Ensure that you maintain the readme.md file.

Modify the .gitignore to the defaults for **Python**. Please, also ensure that you add editor-specific files and folders to .gitignore.

Part A

Part A consists of a **single formative submission**. This work is **individual** and will be assessed on a **threshold** basis. This deliverable is not assessed and is intended to be advisory at this stage.

To complete Part A, incorporate the design, using UML, into the readme.md document. Show this to your tutor in-class. If acceptable, it will be signed-off. You will receive immediate **informal feedback** from your **tutor**.

Part B

Part B 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 B, prepare a draft version of the MUD. Please ensure that the source code and related assets are pushed to GitHub and are made available prior to the scheduled peer-review workshop. Then, attend the scheduled session.

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

cont...

Part C

Part C 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 C, revise the MUD based on the feedback you have received. Then, upload it to the LearningSpace. Ensure that you include the readme.md document containing the design that you developed in Part A. Please note, the LearningSpace will only accept a single .zipfile.
You will receive **formal feedback** from your **tutor** three weeks after the final submission deadline.

Part D

Part D 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 D, prepare a practical demonstration of the computer programs. Ensure that the source code and related assets are pushed to GitHub and a pull request is made prior to the scheduled viva session. Then, attend the scheduled viva session.

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

Additional Guidance

A common pitfall is poor planning or time management. Many underestimate the work involved in designing and implementing multiplayer games. It simply cannot be crammed into a last minute deluge just before a deadline. There is a critical and time-consuming phase of testing! It is, therefore, very important that you begin work early and sustain a consistent pace: little and often.

The first deadline is close to the start of the module and not much material will have been covered by this point. Please rest assured, this first formative submission is supposed to be a simple analysis of design. It is advisory to kick start the project such that you receive early feedback to give you some direction and to encourage you to practice your programming skills.

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.

cont...

Please raise an issue and comment accordingly.

Additional Resources

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

  https://resourcelists.falmouth.ac.uk/

cont...

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.	Parts A—D are complete and timely. Enough work is available to hold a meaningful discussion. Provided a meaningful review of a peer's work. Submission of client and server applications in Python. Clear evidence of programming knowledge and communication skills. Appropriate use of GitHub for version control. No breaches of academic integrity				
MUD Design	20%	Little or no design work. Design does not incorporate concurrency in either client or server No UML of client / server form No UML of client / server function No UML of client / server communications No client wireframes	Design is captured within class, state and sequence diagrams. Diagrams are laid out in a way where they are generally hard to follow and interpret. Little consideration is given to use cases. Wireframe shows basic layout of client UI	Design is captured within class, state and sequence diagrams. Diagrams make some sense. Some consideration is given to use cases. Wireframe shows basic layout of client UI	Design is captured within class, state and sequence diagrams. Diagrams make sense and are relatively easy to interpret. Some consideration is given to use cases and socket parallel processing. Wireframe shows basic layout of client UI	Design is captured within class, state and sequence diagrams. Diagrams make sense and are relatively easy to interpret. Much consideration is given to use cases and socket parallel processing. Wireframe shows basic layout of client UI	Design is captured within class, state and sequence diagrams. Diagrams make sense and are straightforward to interpret. Significant consideration is given to use cases and socket parallel processing. Wireframe shows basic layout of client UI
MUD Implementation	10%	Absence of threading Client has not been developed in Python Server has not been developed in Python	Class hierarchy bears little or no relationship to UML Functionality bears little or no relationship to UML Variables, functions, class names and comments make code hard to follow	Some coherence between UML and code for class hierarchy and functionality Some, but not all variables, functions and classes are well named and easy to follow Some comments make sense	Reasonable coherence between UML and code for class hierarchy and functionality Most variables, functions and classes are well named and easy to follow Comments add some value to code base	Strong coherence between UML and code for class hierarchy and functionality for most of the applications Variables, functions and classes are generally well named and easy to follow Comments add a lot of value to code base	Strong coherence between UML and code for class hierarchy and functionality Variables, functions and classes are well named and easy to follow Comments add significant value to code base
Demo	30%	Client and/or server applications do not run Client and server cannot connect to each other	Demo successfully handles a small set of use-cases Problematic to maintain multiple clients with server	Demo successfully handles a reasonable set of use-cases Client/server operation is solid and stable with few, if any issues.	Demo successfully handles most, but not all use-cases. Client/server operation is solid and stable with few, if any issues.	Demo successfully handles all the use-case defined Client/server operation does not fail during demo	Demo successfully handles all the use-case defined Client/server operation does not fail during demo Presentation / applications are particularly slick or impressive in some way