DISTRIBUTED PROCESSING TASK

Version 1.0
BSc Computing for Games
COMP260

Dr Michael Scott

Introduction

"...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

"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

In this assignment, you are required to implement algorithms that process data in a *distributed* manner, taking advantage of databases and concurrency.

Games are resource intensive. Compounding this issue, players are sensitive to performance issues. It is important, therefore, to leverage the available resources to ensure adequate performance. Distributed processing is one way to achieve this. It is, therefore, important that you apply the principles of concurrency, coordination, and agreement to create efficient games.

This assignment is formed of several parts:

(A) **Select**, as a **group**, **one** of the contracts provided by your tutor and:
  i. **state** which contract you will work on;
  ii. **gather** the requirements for the contract;

(B) **Implement**, collaboratively, a draft program, in a language of your choice, that will:
  i. **address** the requirements implied by the contract;

(C) **Write**, collaboratively, a final computer program that will:
  i. **revise** any issues raised by your tutor and/or your peers.

(D) **Present**, as an **individual**, a practical demo of the computer program to your tutor that will:
  i. **show** your academic integrity;
  ii. as well as **demonstrate** your **individual** programming knowledge **and** communication skills.

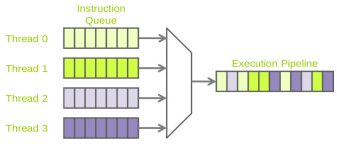
Assignment Setup

This assignment is a **collaborative 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 **Java**. Please, also ensure that you add editor-specific files and folders to `.gitignore`.



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

Part A

Part A consists of a **single formative submission**. This work is **collaborative** and will be assessed on a **threshold** basis.

To complete Part A, write about your contract in the `readme.md` document. Show this to your tutor in-class. If acceptable, this 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 **collaborative** 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 draft versions of the computer program. Ensure that the source code and related assets are pushed to GitHub and a pull request is made prior to the scheduled peer-review session. Then, attend the scheduled peer-review session.

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

Part C

Part C is a **single summative submission**. This work is **collaborative** 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 computer program based on the feedback you have received. Then, upload it 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 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

It is critically important that you do not neglect your individual roles in the development process. Programming in pairs means that you work together on the same computer—switching between driver and navigator. It is a great opportunity to develop your technical communication skills and overcome common misconceptions about programming. It should not, however, be treated as a 'free ride'—you will get to review each others' progress.

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 first deadline is quite close to the start of the course 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 requirements. We expect there to be errors. However, it is very important to make a start on this project so you recieve early feedback to give you some direction and to encourage you to practice your programming skills across the entire duration of the course. Ideally, you should be programming every day!

The peer-review component of this work does sometimes raise alarm. However, the only way to learn how to review code is by reviewing code. Your tutor will guide you through the process and provide advice. With practice, it will become clear what is satisfactory by discussing the quality of work with your peers and your tutor during the peer review sessions.

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

Criteria marked with a ‡ are shared by the group. Criteria marked with a † are weighted by individual contribution to a shared deliverable and viva performance. All other criteria are individual.

Criterion	Weight	Refer for Resubmission	Basic Competency	Basic Proficiency	Novice Competency	Novice Proficiency	Professional Competency
Functional Coherence	5% ‡	No algorithm has been implemented successfully. The source code does not compile or there are serious logical errors.	At least one algorithm has been implemented successfully. There are many obvious logical errors, more than one of which is significant.	At least two algorithms have been implemented successfully. There are several obvious logical errors, at least one of which is significant.	At least three algorithms have been implemented successfully. There are some obvious logical errors, which are not significant. The brief has been satisfied.	At least three algorithms have been implemented successfully. There are few obvious logical errors, which are cosmetic and/or superficial. The brief has been satisfied.	At least three algorithms have been implemented successfully. There are no obvious logical errors. The brief has been satisfied.
Sophistication	10% ‡	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).	Little insight into the appropriate use of programming constructs is evident from the source code. The program structure is poor.	Some insight into the appropriate use of programming constructs is evident from the source code. The program structure is adequate.	Much insight into the appropriate use of programming constructs is evident from the source code. The program structure is appropriate.	Considerable insight into the appropriate use of programming constructs is evident from the source code. The program structure is effective. There is high cohesion and low coupling.	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.
Maintainability	15% ‡	There are no comments in the source code, or comments are misleading. Most variable names are unclear or inappropriate. Code formatting hinders readability.	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.	The source code is somewhat well commented. Some identifier names are descriptive and appropriate. An attempt has been made to adhere to thhe PEP-8 formatting style. There is little obvious duplication of code or of literal values.	The source code is reasonably well commented. Most identifier names are descriptive and appropriate. Most code adheres to the PEP-8 formatting style. There is almost no obvious duplication of code or of literal values.	The source code is reasonably well commented, with Python doc-strings. Almost all identifier names are descriptive and appropriate. Almost all code adheres to the PEP-8 formatting style. There is no obvious duplication of code or of literal values. Some literal values can be easily "tinkered" in the source code.	The source code is very well commented, with Python doc-strings. All identifier names are descriptive and appropriate. All source code adheres to the PEP-8 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.
Distributed Systems Architecture	25% †	No use of distributed approaches. Unable to articulate either database, segmentation, or concurrency design decisions.	Poor performance and scalability gain. Very little ability to articulate database, segmentation, and concurrency design decisions.	Some performance and scalability gain. Little ability to articulate database, segmentation, and concurrency design decisions.	Much performance and scalability gain. Some ability to articulate database, segmentation, and concurrency design decisions.	Considerable performance and scalability gain. Much ability to articulate database, segmentation, and concurrency design decisions.	Significant performance and scalability gain. Considerable ability to articulate database, segmentation, and concurrency design decisions.
Use of Version Control	5%	GitHub has not been used.	Source code has rarely been checked into GitHub.	Source code has been checked into GitHub at least once per week. Commit messages are present. There is evidence of engagement with peers (e.g. code review).	Source code has been checked into GitHub several times per week. Commit messages are clear, concise and relevant. There is evidence of somewhat meaningful engagement with peers (e.g. code review).	Source code has been checked into GitHub several times per week. Commit messages are clear, concise and relevant. There is evidence of meaningful engagement with peers (e.g. code review).	Source code has been checked into GitHub several times per week. Commit messages are clear, concise and relevant. There is evidence of effective engagement with peers (e.g. code review).

Criterion	Weight	Refer for Resubmission	Basic Competency	Basic Proficiency	Novice Competency	Novice Proficiency	Professional Competency
Basic Competency 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 reflection on own performance and contribution. Only constructive criticism of pair-programming partner is raised. No breaches of academic integrity.				