1. Introduction

The module is 100% coursework based, and has a single component , weighted at 100% of the module marks. In summary:

- You will select [redacted] of customised board game, following the style of the lab exercises.
  - You ar[redacted] Tic-tac-toe, but you should implement some custom extens[redacted]
  - You m[redacted] similar grid-based board game (battleships, draughts, connec[redacted]is, sudoku, etc.
  - Your task should be customised by implementing 2 or more bespoke features. These should be **simple** features and overly complex features will not be rewarded. Examples may include accessibility features, improved UI, additional game-mechanics, or a simplistic rules-based AI player.
  - You should write a brief text description of your task considering the features that are required to be implemented.
- You will write **three** solutions to your task, using different programming languages and paradigms.
  - Your solutions must each exhibit a range of paradigmatic features
  - Your solutions must each use a unique programming language
- For each solution, you will provide:
  - (a) the name of the language and paradigm you have used.
  - (b) Screenshots of your design process, demonstrating how you created the code and the validation process you used to ensure that it was suitable for your task
  - (c) a short description of your programme, explaining how the code you have written completes the task and how your programme fits the named paradigm.
- The deliverables are:
  - Your uniquely proposed task description (unmarked)
  - A folder containing 3 subdirectories, one per language. Each subfolder should contain a text file with your code.
  - A document with the following specified headers (template available on Moodle):
    - Code Design
    - Language 1 – Paradigm Description
    - Language 2 – Paradigm Description
    - Language 3 – Paradigm Description
    - Comparison

a) Propose a novel task

You should write your own task description. As a result of your customised features, there should not be solutions readily available via web search. You should base your task on an existing grid-based board game in a similar style to the lab exercises.

For example, you may start with the task of developing a tic-tac-toe game. To make this unique you could implement some additional rule, or rules to the game, such as making the board larger (e.g., 7x7, or NxM) or implement some extra gameplay rule (e.g., you can choose to remove one of the opponents O's or X's every 3rd turn. You should be imaginative in creating your task to ensure it is unique to you and different from the rest of the class. You must incorporate a minimum of 2 customisations to differentiate from others in the group.

To further personalise your solution, you should incorporate either your student ID number or your name into the task description. This helps for future plagiarism detection. For example, you may seed a random number generator with your student ID, or if your task includes some text element then you may use your name (e.g., a cryptograph). Failing that you can include this information in the comments of your code.

I am not making a limit to the difficulty or ease of the task. You should choose a difficulty level that you feel is appropriate to your coding ability and that will set you an appropriate challenge. Solutions to more difficult problems will likely expose more interesting features of the languages and paradigms, leading to the opportunity to score more highly. As a rough guide, your task should be more difficult than a typical lab-exercise (e.g., fizzbuzz with the numbers changed is probably too simple), but less difficult than a typical end of year assignment (e.g., you should not propose to implement a full-fledged mobile app or web front end).

b) Choose three different languages

You should select three different languages to use to solve your task. These languages **must** be selected from those taught during the module. You can refer to Moodle for a full list of languages that have been covered. The languages you choose should allow you to solve the task in a variety of programming styles. **You must use a different programming paradigm for each solution** and your choice of language should reflect this. The five programming paradigms we cover are as follows: Imperative, Procedural, Object-Oriented, Functional, Logic. Solutions which only cover languages in the Imperative style (i.e., selecting imperative, procedural and OO) will find it difficult to give an appropriate level of comparison and may be self-limiting. Students wishing to achieve grades of a first or higher should incorporate at least one solution making use of a declarative paradigm language (e.g., functional or logic).

c) Create Solutions

You should write a bespoke solution in each language, conforming to a paradigm.

Please ensure that your code is appropriately indented, well commented, and conforms to appropriate standards for the language you are coding in (e.g., variable naming conventions, etc.). You are welcome to use the same approach to solve the task you have designed across your three solutions, however you should design your solutions in such a way that the specific paradigmatic features of each language you have used may be properly showcased.

d) Document Solutions

You must provide your documentation using the template on Moodle. The template will ask you to write the following 5 elements of documentation, each of which will be marked according to the mark scheme (see Section 3).

- Code Design
- Language 1 – P
- Language 2 – P
- Language 3 – P
- Comparison

The first document (**Code Design**) should show your design and development process for your solutions across the three languages and paradigms. This document will be used alongside your submitted code to judge your understanding of the language you have implemented. You should include any sketches, UML diagrams, class diagrams, pseudocode or wireframes that you create. There may be some crossover in your design work between languages, but you should still document this for each language. As part of your code design document you should also capture the development process that you have undertook including testing and bug-fixing. You should include intermediary screenshots of your design process. If you use a large language model or copilot as part of your programming, you should include screenshots of all interactions, as well as some indication as to how you used this information and how you validated the results. This document may be presented in a 'scrapbook' format, and should be made mostly of images or figures (with short connecting texts) collected during your design and implementation process.

The next three documents **(Paradigm Description)** consist of a short paragraph for each solution (max 300 words per solution) describing the language features that you have used to solve the solution, and explaining how the solution conforms to the stated paradigm. A typical solution might spend 200 words on the former and 100 words on the latter, although this will vary from one language to another.

You must not use a language model to produce either of these documents. They will typically produce hallucinated documentation or false reasoning for this type of task which will impede your marks.

The final document (**Comparison**) should be no more than 1000 words and should highlight similarities and differences between your solutions in each pair of languages, especially considering the paradigms that you have conformed to. A typical solution might spend around 150 words introducing the three languages and paradigms, then 250 words per language pair highlighting similarities and differences in approach, with 100 words reserved for a summary conclusion. You may assume that the marker is aware of your task and has read your design and explanation documents.

Again, you must not use a language model to produce your comparison document as they are not suitable for this task and are likely to give incorrect answers, harming your chance to succeed

2. The Submission

Your submission is via Moodle. You must submit a zip file containing a folder. The folder should have your ID number as its n you should place:

(a) a text file containi you have written.
(b) A word document completed template with the 5 required elements:
    a. Code Desi
    b. Language on
    c. Language on
    d. Language 3 – Paradigm Description
    e. Comparison
(c) 3 sub-folders. Each sub-folder should have the name of the programming language that you have used for that solution. Inside each sub-folder you must have a text file containing the code you used named **LANG_code.txt**, where LANG is replaced with the language you have used. I will accept files with a language specific extension (e.g., `.c' for a c solution), but `.txt' is preferred.

A sample file hierarchy is given below. Note, you are free to choose any 3 languages from the course:

- 99999999
    - Task.txt
    - Documentation.docx
    - Python
        - Python_code.txt
    - Prolog
        - Prolog_code.txt
    - GO
        - GO_code.txt

3. Mark Scheme

Marks will be apportioned as follows:

- 15% Code Design
- 15% Language 1 – Paradigm Description
- 15% Language 2 – Paradigm Description
- 15% Language 3 – Paradigm Description
- 40% Comparison

Individual mark schemes for each section are given below:

Design

0 marks: No code or documentation thereof.

1-5 marks: Poor code quality, with little design and implementation documentation work, or design and implementation documentation is incoherent and unrelated to submitted code.

6-10 marks: Adequate code quality and design design and implementation documentation work. Code evidences appropriate knowledge of language features. Design and implementation documentation is related to submitted c

11-15 marks: Excellent sive design and implementation documentation work. Code evidences advanc age features. Documentation goes beyond usual expectations for a final udent.

Paradigm

0 marks: No documentation.

1-5 marks: Paradigm is incorrectly identified. Poor description of features, with little relationship to the code.

6-10 marks: A paradigm is stated with appropriate reasoning. Most features are correctly described, with few to no errors.

11-15 marks: Paradigm is correctly identified. Outstanding description of features, showing exceptional understanding of how the given paradigm is used.

Comparison

0 marks: No Comparison.

1-10 marks: An inadequate comparison, covering an incomplete set of paradigms. Little or no criticality in evaluation.

11-20 marks: An adequate level of comparison. At least two paradigms are correctly compared. Some appropriate features are identified and equivalencies are demonstrated in solutions with little or no errors.

21-30 marks: A good degree of comparison. All paradigms are compared appropriately. A complete set of features is identified with no errors made. High level of criticality and understanding of programming paradigms.

31-40 marks: An excellent degree of comparison, above and beyond the reasonable expectations for the final year of study. Each paradigm is compared to the other two paradigms. Highly coherent analysis of features used.

Appendix A – Example Design Document

I decided to implement [illegible] programme using Haskell, Python and Prolog. My high-level pseudocode for th[illegible]s follows:

1. Represent boa[illegible]
2. Implement rec[illegible]rough a single list determining if there is a win (rows)
3. Implement rec[illegible]rough a list of lists determining if there is a win (cols)
4. Implement rec[illegible]rough a list of lists determining if there is a win (diagonal)
5. Implement fun[illegible]' in a row-col position. Signature: char, [[Integer]] -> [[Integer]]
6. Implement function to remove a 'M' or 'S'. (reuse above function?)
7. Implement function to govern game logic
    a. M goes first, then S
    b. At each iteration get a number (1-49) indicating cell to play in
    c. Every 3rd turn players can remove a cell

I adapted this for my Prolog solution as follows:

[[PSEUDOCODE 2]

I have provided screenshots of the pseudocode that I wrote on my whiteboard for each function in each language below:

Recursive function: [SCREENSHOT 1]

Add/remove char to board: [SCREENSHOT 2]

Game Loop: [SCREENSHOT 3]

During my implementation process, I wrote the following code as a first iteration:

[SCREENSHOTS OF CODE]

This allowed me to identify the following errors in my approach, which led me to redesign my system as follows:

[SCREENSHOTS OF ERRORS AND UPDATED CODE]

Once I had a working system for each language, I decided to test it. The tests that I ran are as follows:

1) Run to the end with M player winning
2) Run to the end with S player winning
3) Run to the end with a draw.

[SCREENSHOTS OF TESTING]

Appendix F – Feedback Sheet

Marker Name: John

Student Name: Exampl

Student ID: 99999999

Code Design (15)	Solutic… Explana… (15)	…ion 2 …nation …6)	Solution 3 Explanation (15)	Comparison (40)	Total (100)
12	7	3	10	30	62

This submission contains a tic-tac-toe game with a modified board design and an additional rule to allow players to remove their opponents tiles. Solutions 1 and 3 (Haskell and C++) were well implemented in the functional and OO paradigms. The OO structure in C++ was exceptionally well designed and led to efficient code. Solution 2 failed to use Rust correctly and did not state the paradigm that was being used.

I have used the criteria below to mark your work. You can see a further breakdown of your marks by matching your assigned grade to the given band for each category.

Design

0 marks: No code or documentation thereof.

1-5 marks: Poor code quality, with little design and implementation documentation work, or design and implementation documentation is incoherent and unrelated to submitted code.

6-10 marks: Adequate code quality and design design and implementation documentation work. Code evidences appropriate knowledge of language features. Design and implementation documentation is related to submitted code.

11-15 marks: Excellent code quality and extensive design and implementation documentation work. Code evidences advanced knowledge of language features. Documentation goes beyond usual expectations for a final year undergraduate student.

Paradigm

0 marks: No documentation.

1-5 marks: Paradigm is incorrectly identified. Poor description of features, with little relationship to the code.

6-10 marks: A paradigm is stated with appropriate reasoning. Most features are correctly described, with few to no errors.

11-15 marks: Paradigm is correctly identified. Outstanding description of features, showing exceptional understanding of how the given paradigm is used.

Comparison

0 marks: No Comparisc

1-10 marks: An inadeq ing an incomplete set of paradigms. Little or no criticality in evaluation.

11-20 marks: An adequ n. At least two paradigms are correctly compared. Some appropriate features a alencies are demonstrated in solutions with little or no errors.

21-30 marks: A good degree of comparison. All paradigms are compared appropriately. A complete set of features is identified with no errors made. High level of criticality and understanding of programming paradigms.

31-40 marks: An excellent degree of comparison, above and beyond the reasonable expectations for the final year of study. Each paradigm is compared to the other two paradigms. Highly coherent analysis of features used.