Foundation Degree in Digital Innovation

Module Name	**Advanced Programming**
Year of Study	Year 2
Assessment	**Module Assignment 2 of 2 - Individual Assignment**
Name of Student	……
Name of Module Lecturer	Anand Veeraswamy
Assignment Weighting	70%
Release Date	19th June 2023
Submission Date	12th July 2023 *(3.5 weeks)*

PLAGIARISM: please refer to the student handbook for the college policy on plagiarism and procedures for correct referencing. Material should not be copied directly from the web or a publication without a correct acknowledgement at the place of insertion of the material. It is not sufficient to include a bibliography reference only. It is also an offence to copy work from another student.
Statement (to be signed by the student before hand-in):
"I confirm that the submitted coursework is my own work and that all material attributed to others (whether published or unpublished) has been clearly identified and fully acknowledged and referred to original sources. I agree that the College has the right to submit my work to the plagiarism detection service Unicheck for originality checks."

Signed……..

- Demonstrate an understanding of the advanced principles and concepts of Object-Oriented Programming.
- Analyse a problem and determine what problem elements to represent as functions or objects.
- Apply object-oriented techniques and create object-oriented models of reasonable complexity.
- Effectively use inheritance to promote reuse and develop programs.
- Appreciate the need for continuing professional development in recognition of the need for lifelong learning, professionalism and employability.

Where coursework is submitted late and there are no accepted extenuating circumstances it will be penalised in line with the following tariff:

- Submission within 6 working days: a 10% reduction for each working day late down to the 40% pass mark and no further.
- Submission that is late by 7 or more working days: submission refused, mark of 0.

1st Marker Signature……Mark……Date……

Mark after late submission penalty (if applicable)

2nd Marker Signature……..Mark ……Date …… (if applicable)

Assignment Description

Use a structured, object-oriented design approach for the concept, design, development, testing and potential post completion and further improvements of AdaShip (a REPL.IT C++ computer console version/adaptation of the classic paper-based Battleships game). Optionally, if you would like to design something else - use the same approach and level of problem complexity for the creation of your own, approved, individual advanced computer system, application, or game.

If useful, the option to work in a small (ideally 2, max 3) critically supportive team is available. However, regardless of this - you are individually expected to adopt an agile and iterated development method supported with the principles of decomposition and abstraction and ideally combined with a test early and fail fast approach to demonstrate your effective use of inheritance and code reuse. In addition, both as a team and individually try to ensure that each MVP release/iteration/version or evolution of your design and/or codebase incorporates some type of innovation and/or improvement.

Suggested Guidance Re Possible Planning and Preparing:

Although you are free to explore and approach this challenge in any preferred method the checklist below has been provided for your review, consideration and reflection; please feel free to skip, adapt, utilise, blend any or all of the following aspects as part of your planning and preparation:

- Problem Outline and Analysis.
- Task or story backlog with clear definitions of done.
- Evaluated Requirements reviewed and linked to required features and functions.
- Use and incorporation of networking capability.
- UML overall logic and/or data diagram.
- Code Smells, reuse / reusable.
- Fundamental OO: classes, objects, instantiation.
- Advanced OO: inheritance, composition, encapsulation, polymorphism, abstraction.
- Team supported code reviews.
- Code refactoring and/or use of design patterns.
- Use of multithreading and/or asynchronous operations.
- Designed and Implemented Security.
- 15 Point Good Quality Standards Referenced Review.
- Approach to design and development review.
- Evidence of testing.
- Further development / improvements Review.
- Self-reflection and review on skills, experience and insights utilised and improved.

Pre-Design Steps:

- Outline the system's goal/vision and define the needed functionality.
- Determine how your system will work.
- Outline the features & functions needed and wanted.
- Consider the user's journey and workflow for example, usability & cognitive load.

Design Steps

- User Interface Design – input, feedback loops, validation, etc.
- Application Mechanics and/or logic and/or data flow.

Development Steps

- Testing and QA – testing, features, functions, mechanics, logic and data.
- Iterated and/or Continuous Development – completion of a MVPs / release candidates.
- Demo planning & feature review documentation and evaluation & source code submission.

AdaShip Concept - How AdaShip Is Played

Essentially, AdaShip is a clone of the classic 'Battleship' game – as a default, AdaShip is a two-player, turn based game of sea warfare. You and an opponent each place a number of ships on your own board, and you then alternate turns "firing" torpedoes at each other's ships. The game is won when one player has destroyed/sunk all of the other player's ships.

Setup:

By default, each player uses two 10x10 boards (a shipboard and a targetboard). Ship boards are used to initially position and hold a record of the location of your ships and any hits made by opponents. The target board is used to keep track of where you have fired (i.e., guessed) and the outcome; for example – hit or miss. Board locations are referenced using a single notation type coordinate system ('A1', 'F4', etc). The coordinate system must be designed to use alphabetic letters (A, B, …) to represent columns and numbers (1, 2, …) to represent rows. For example (illustrated below), the player's Carrier is located at B2, B3, B4, B5 and B6.

	A	B	C	D	E	F	G	H	I	J
1										
2		C							D	
3		C							D	
4		C							D	
5		C								
6		C			B	B	B	B		
7										
8										
9			S	S	S				P	
10									P	

example of shipboard showing position and orientation of each ship

When placing ships, each player must place all their ships on their shipboard; ships have different sizes and can be either placed horizontally or vertically; a player's ships cannot be placed such that any of their ships are either partly or entirely outside their board's boundaries, across one (or occupy the same space as) a previously positioned ship.

By default, there are five types of ships:

1. Carrier - Length 5
2. Battleship - Length 4
3. Destroyer - Length 3
4. Submarine - Length 3
5. Patrol Boat - Length 2

A typical, two player game (player v computer)

In a typical two-player game, each player would set up his own shipboard. Since your default opponent for this assignment will be the computer, you should set up your own board first, and have the computer decide where to place its ships (legally of course).

Typical game play

The game is played in turns, where each player 'fires a torpedo' (by calling out a board coordinate) and the opponent indicates whether the 'torpedo' resulted in a "hit" or "miss". A "hit" indicates that the called position corresponded to a valid ship coordinate, otherwise it is a "miss". Players record their called positions using their targetboard; a record of both "hits" and "misses" should be recorded.

	A	B	C	D	E	F	G	H	I	J
1	M									
2		H								
3		H							H	
4		H								
5		H			M					
6		H			H				M	
7										
8			M							
9			H							
10										

example of targetboard showing recorded hits and misses.

Winning

Turns are repeated until all of your opponent's ships have been 'sunk/destroyed'; players must indicate as part of the response to each hit whether an entire ship has been sunk or just a single hit; for example, if our Carrier is located at B2, B3, B4, B5 and B6 and each position has been hit we need to indicate that the opponent sunk our ship.

AdaShips Features & Functions –

In addition to development, you are also required to provide a well-structured README document; details on the structure have been provided in the submission section.

MVP/Alpha Release version 1 - your game is expected to support:

1: A run-time, single, plain text file called 'adaship_config.ini' that defines the game's configuration; your file should be setup to match the following structure:

Board: 10x10
Boat: Carrier, 5
Boat: Battleship, 4
Boat: Destroyer, 3
Boat: Submarine, 3
Boat: Patrol Boat, 2

2: A usable, intuitive and highly efficient **Set-Up** interface in which a player can with minimal effort:

- Select from a game menu to start a:
  - One player v computer game
  - Quit
- Select and place any of their non-placed or placed ships (as provided via the config file) at any valid position on their shipboard.
- Clearly indicate the current non-placed and placed ships.
- See their current shipboard.
- Be robust enough to prevent any invalid behaviours, prevent or correct any illegal placements and avoid system issues or errors related to user input.
- Allow any non-placed ships to be 'auto-placed'.
- Auto-place all ships.
- Support a 'quit' game and 'reset' shipboard option.
- Support a 'continue' option if all ships have been placed and the user has confirmed they are happy with the current placements.

3: A computer-based opponent that can automatically complete a valid setup process using a random strategy (i.e., place all their ships at valid but constantly differing positions on their own shipboard); essentially this should utilise the auto-place feature outlined above.

4: A usable, intuitive, and highly efficient **Turn** based process supported with a minimal user effort interface in which game turns are played:

- Player's turn:
    - Their own, current, and up to date ship and targetboards are displayed as well aligned tables (rows and columns).
    - Players are able to 'fire' a single torpedo at a valid location (i.e., not previously targeted and within the board boundary) using the single notation coordinate (e.g., F2, etc.); if the location is invalid the player is asked to re-try.
    - Auto 'fire' option; this option targets and 'fires' at valid locations.
    - Players are clearly notified of a 'hit' or 'miss' (or win) based on the outcome of their turn.
    - Players are able to quit the game (if not already won).
    - Players are required to 'press a key' to end their turn.

- Computer's turn:
    - Its own, current and up to date ship and target boards are displayed as well aligned tables (rows and columns).
    - It uses its own up to date 'targetboard' to randomly 'fire' a torpedo at a valid location (i.e., not previously targeted and within the board boundary).
    - A clear notification of a 'hit' or 'miss' (or win) based on the outcome is shown.
    - A user based 'press a key' interaction is required to end the computer's turn.

MVP/Alpha Release version 2 – extend v1 so that your game also supports:

5: Improved customised game and configuration settings (based on changing the 'adaship_config.ini' file):

- Any valid size of board (valid ranges are: 5x5 to 80x80)

6: An extended **Set-Up** interface in which a player can with minimal effort:

- Select from a game menu to start a:
    - One player v computer game
    - Two player game; *essentially replacing the computer with a second player*
    - One player v computer (salvo) game
    - Two player game (salvo) game
    - Quit

7: Improved game play "salvo" variation:

The salvo implementation updates the basic game play by allowing the current 'player' (player or computer) to 'fire' one torpedo per their remaining ships. For example, if the 'player' has three 'non-destroyed' ships instead of a single valid coordinate (e.g., F2) they could enter F2 E2 G2 (one coordinate per ship) – once entered, details on any 'hits' and/or 'misses' are clearly provided, and all appropriate boards are updated to reflect this salvo.

Beta/Release Candidate – extended v2 so that your game also supports:

8: Improved customised game and configuration settings (based on changing the 'adaship_config.ini' file):

- Unlimited additional boats: each additional boat should follow the file's existing structure; for the purpose of this assessment and as there is a logical and rational limit 'unlimited' simply means your solution's design and logic technically supports any number of boats.

9: An extended **Set-Up** interface in which a player can with minimal effort:

- Select from a game menu to start a:
  - One player v computer game
  - Two player game; *essentially replacing the computer with a second player*
  - One player v computer (salvo) game
  - Two player game (salvo) game
  - One player v computer (hidden mines) game
  - Two player game (hidden mines) game
  - Computer v computer (hidden mines)
  - Quit

10: Improved game play "hidden mines" variation:

The hidden mines implementation updates the basic game play with five randomly dropped mines. The hidden mines are essentially added to each 'players' shipboard and remain 'hidden' during the set-up phase (optionally they could be added on completion of each set-up). However, they should be clearly displayed during game turns as part of each 'players' shipboard.

	A	B	C	D	E	F	G	H	I	J
1										
2		C		M					D	
3		C							D	
4		C					M		D	
5		C								
6		C			B	B	B	B		
7										
8					M					
9			S/M	S	S			M	P	
10									P	

example of shipboard showing position and orientation of each ship and hidden mines

If an opponent's torpedo 'hits' a hidden mine that location and the eight immediately surrounding it 'explode'. If any of the players ships directly intersect with the 'explosion' they are 'hit' as per normal play. Positions 'outside' the board can be ignored.

11: Improved 'computer player' targeting problem:

Design and implement a separate, optimised search and/or targeting algorithm, it is likely that your solution has implemented a generally random or basic algorithm to 'pick' targets; research and implement a better than random solution to this problem.

Submission

You are required to submit two pieces of evidence via the module's Google Classroom by the appropriate submission date (see cover) –

1. A single, complete, timely with clear commits and tested GitHub linked repo.
2. A suitable 5/10 min (approx.) well planned and narrated videoed demonstration of your project running including a 'post demo' well-planned code review and/or walk-through; you will need to determine optimal times between the 'demo' and recorded code review – for example 4min demo, 6 min code review.

Your submission is expected to demonstrate multiple, personal commits and provide:

- **10 Marks:** A suitable GitHub project link – (e.g., an AdaShip REPL.IT compatible and testable C++ project, or a complete, approved, appropriate, and lecturer agreed project).
- **35 Marks:** A formatted well-structured and planned, academic and professional README.md; your README document should be contained in your GitHub repo - see Structuring and completing your Readme document (below for more detailed information).
- **55 Marks:** A 5/10 min (approx.) video demonstrating an intuitive, innovating and sophisticated run-time version of your project's most recent or final release; your demonstration should be well narrated and planned to help you highlight, showcase, and focus on all the expected and implemented features and functions. Your video can be recorded using Google Meet, or any other Ada Compatible method; if you are unsure about the format of your video, please test its compatibility then discuss this with the module delivery team or lead lecturer. Your video should include an initial title sequence indicating your name, project outline and the key features being demonstrated and supported with a 'post demo' code review and/or critical code walk-through. You are responsible for ensuring your demonstrated problem and solution complexity is sufficient to achieve the overall mark you would like to achieve; review the project outline above for an example of the expected level of sophistication.

Reviewing your GitHub Repo: 10 marks

Clear, suitably time-stamped, and well-presented, implemented evidence of 'good' programming standards (see below) across your entire development project – your submission code will be generally reviewed and will be expected to provide appropriate evidence of quality and be at the expected problem complexity, inconsistent or poorly presented work may impact your mark for this section:

- None/minimal: 0 – 29%
- Minimal/Some: 30 – 39%
- Good overall and consistent evidence of implementing some (ideally 5+) standards: 40 – 59%
- Clear, consistent evidence of many (ideally 8+) well implemented 'good' standards: 60 – 69**%**
- Clear, consistent evidence of most (ideally 12+) well implemented 'good' standards: 70+**%**

Structuring and completing your README document: 35 marks

The three titles below provide and outline the structure and subsections expected as part of your submission; in addition to previous notes or discussions, for more information on achieving higher grades in written work please review the AdaHelp document available via the Google Classroom.

1. **Challenge Outline (academic standard: pass level detail: section required for pass) – 10%**
    a. Summary and review of the problem, overall proposed solution.
    b. UML style diagram illustrating initial overall solution (linked to 1a)
    c. Initial working plan, overall approach, development strategy and approach to quality (linked to 1a, 1b).
    d. Analysis and decomposition of the overall problem into key 'epic' style tasks (linked to 1b, 1c).
    e. Initial object-oriented design ideas and planned phased breakdown into smaller tasks (linked to 1d).

2. **Development (academic standard: merit level detail: section required for merit) – 15%**
    a. Adoption and use of 'good' standards (linked to 1a, 1b, 1c).
    b. Phase 1 development: tasks, code review and changes (linked to 1d,1e).
    c. ..repeated for each development phase.
    d. Phase n development: tasks, code review and changes (linked to 1d,1e).
    e. Ensuring quality through testing and resolving bugs (linked to 1a, 1b, 2a, 2b..2c).
    f. Reflection on key design challenges, innovations and how they were solved (with examples).

3. **Evaluation (academic standard: distinction level detail: section required for distinction) – 10%**
    a. Analysis with embedded examples of key code refactoring, reuse, smells.
    b. Implementation and effective use of 'advanced' programming principles (with examples).
    c. Features showcase and embedded innovations (with examples) - opportunity to 'highlight' best bits.
    d. Improved algorithms – research, design, implementation, and tested confirmation (with examples).
    e. Reflective review, opportunities to improve and continued professional development.

Computer System, Application and Demo/Video: 55 marks

- 0 – 29%: None/minimal evidence of developing an advanced application.
- 30 – 39%: Minimal/Some evidence of developing and/or being able to test an advanced application (may include one or more blocking errors) supported with some basic code walk-throughs and/or reviews.
- 40 – 59%: A well-structured demo, providing good overall evidence of the development and running your advanced application (without significant blocking errors) and is supported with a generally correct and technical code walk-through / review of some of the features demonstrated.
- 60 – 69%: Comprehensive, error free and well narrated demo highlighting many of your 'designed' features & functions supported with a commentary on issues, design challenges, innovations and solutions.
- 70+%: Ideally a complete, error free and comprehensive demo with full narration highlighting (ideally) all of your 'design' features and functions supported with technically correct, detailed and insightful critical commentary on issues, challenges, innovations, outstanding problems and possible solutions.

Issues likely to impact your grade:

- *Commits made after the submission date and time.*
- *Incompatibility and/or issues with GitHub and Repl.IT importing*
- *Errors preventing compilation, execution, etc.*
- *Incomplete implementations or a lack of problem, solution, implementation complexity.*
- *Missing evidence and/or inconsistencies in the narration, source or level of critical explanation.*
- *Inconsistent and/or missing information, clear related or linked evidence, differing standards in coding and documentation.*
- *Poor refactoring - Code refactoring is the rewriting of code for clarity and efficiency rather than bug fixing; in essence, this is similar to writing or improving academic 'paper' drafts.*
- *The design and implementation of a less than suitably complex and advanced programming project.*

Appendix A. Expected Standards - Adoption and use of good programming standards:

1. *Include good quality, clear, jargon free and up to date internal documentation / comments; adopt the philosophy of writing comments for non-programmers.*
2. *Eliminate or minimise code duplication & any unnecessary redundancy.*
3. *Strive for simplicity in logic and flow.*
4. *Use a consistent naming convention for functions, variables, objects, etc to provide clear contextual value, improved comprehension, and quick readability.*
5. *Use appropriate and consistent indentation, logical grouping and spaced blocks within your codebases; adopt tabs or a set number of spaces (ideally tabs) for indenting.*
6. *Use space consistently to separate operators and delimiters.*
7. *Be consistent when aligning braces; use a vertically or slanted style.*
8. *Avoid deep nested conditionals.*
9. *Avoid single (long) lines of code containing multiple operations; consider 'one line one instruction".*
10. *Keep variable lifetimes and scope as short and as small as possible.*
11. *Avoid multipurpose functions and variables.*
12. *Conserve system resources.*
13. *Minimise forced type conversion, coercion, or casting.*
14. *Know and test your code: adopt a personal and rigorous testing strategy; don't just see it if works - test and fix its limits.*
15. *Test early and often, fail fast and resolve effectively.*