CCP6124 Assignment: Space Battle

Trimester 2530 ☉ Teams should be of 3 or 4 people – form groups of 4 where possible; group of 3 if there are leftovers.

Topics tested: Information hiding, Inheritance, Polymorphism, and Function Redefinition/Overriding, Operator overloading, UML Class Diagrams

Submission Date: Wednesday 5pm 28 January 2026

Please inform us via the OOPDS CCP6124 T2530 WhatsApp Group if you find any errors in this document. We've checked this document but we're only human 😅

Get ready to command the cosmos in the ultimate **Space Battle Simulation**! The Zapezoids and the Rogoatuskans are battling for the control of the galaxy! You task is to write a sophisticated C++ battle engine that decides the fate of their battles! From the small agile *Jager* to the massive *Corazzata*, every line of code you write will determine who survives the void and who is reduced to space dust. Prepare for launch!

Zapezoids ships

Guerriero

- 123 hit points
- 1 pilot
- 1 light cannon with power 96
- 26% chance of being hit by enemy ship's light cannon.
- 6% chance of being hit by enemy ship's torpedo (because it is fast and can avoid them)

Medio

- 214 hit points
- 1 pilot
- 2 gunners
- 2 light cannons (one for each gunner) with power of 134
- 31% chance of being hit by enemy ship's light cannon
- 11% chance of being hit by enemy ship's torpedo.

Corazzata

- 1031 hit points
- 2 pilots
- 10 gunners
- 4 torpedo handlers
- 10 light cannons with power of 164
- 4 torpedoes with power of 293
- 50% chance of being hit by enemy ship's light cannon
- 25% chance of being hit by enemy ship's torpedoes

Rogoatuskan ships

Jager

- 112 hit points
- 1 pilot
- 1 light cannon with power 101
- 24% chance of being hit by enemy ship's light cannon.
- 5% chance of being hit by enemy ship's torpedo

Kreuzer

- 212 hit points
- 1 pilot
- 2 gunners
- 2 light cannons (one for each gunner) with power of 132
- 29% chance of being hit by enemy ship's light cannon
- 10% chance of being hit by enemy ship's torpedo.

Fregatte

- 1143 hit points
- 2 pilots
- 11 gunners
- 5 torpedo handlers
- 11 light cannons with power of 159
- 5 torpedoes with power of 282
- 60% chance of being hit by enemy ship's light cannon
- 30% chance of being hit by enemy ship's torpedoes

Rules

Strictly no copying from other sources except code given in this course. If detected, all parties involved will get 0 marks – so both the copying team and the copied team gets zero – so don't let other people copy as well. The code must be your own. You are required to implement this assignment using **standard** C++.

Please see us early if you have problems – whether it is problems understanding what you need to do, or problems with team members who are not doing their work. The earlier you meet us, the better chance you'll have of solving the problems. If you wait till the last week before the due date, it'll likely be too late to fix the problems.

If you find as the project progresses that some people are not contributing or problematic in any other way, **please contact your lecturer immediately** so that remedial action can be taken before it gets too late. If you do not do so, you will be liable for any loss of marks.

Game play

Loading

The user starts the program up with a command line like this:

```
XyloTT9L zShips1.csv zCrew1.csv rShips1.csv rCrew1.csv
```

Format of ships.csv: ID of ship, type of ship, name of ship

Format of crew.csv: ID of person, name of person, type of person (pilot/gunner/torpedo handler)

In this example, Xylo- is the team name, and TT9L is the tutorial group, so you should name your main cpp file according to your team name and tutorial group so that when we compile it, the file name will correspond to your team name and tutorial group. This will help us not mix up the submissions.

Assigning crew to ships

The program should read the ship data and the crew data and try to man as many ships as possible given the crew. (Up to you to pick a strategy of how to crew the ships, but the idea is to spread them out – don't bunch them all on a few ships leaving the others mostly empty.)

A ship cannot fly without a pilot, but could fly with lack of pilots. But 2-pilot vessels will be lowered in ability to avoid being hit by 25% if they only have 1 pilot. So for example, if originally 50%, it goes up to 62.5%.

If there are not enough gunners or torpedo handlers, then the weapons with no crew will not be able to shoot during the battle.

How battles work

For simplicity, battles are fought till one side is defeated (i.e. all their ships are destroyed.). In each round,

- Every weapon randomly chooses one surviving enemy ship to target.
- The probability of the shot hitting is given in the ship information above.
- If the enemy ship is hit, their hit points are reduced by the amount according to the ship information above.
- If the hit points get to zero or less, the ship is destroyed.
- Both sides shoot simultaneously, so it is possible for a ship to shoot, and then is destroyed, in the same round.

The progress is shown on the screen in text at every round, showing which ship shot which ship, how many hit points is left after the round, and if the ship is destroyed.

At the end of the battle, the winner is declared, with the surviving ships and their hit points.

Please see the sample data and sample runs provided, in order to get a good idea of what we are looking for in terms of functionality. Note that due to randomisation, your runs might be different – see sample 2, which clearly illustrates how different runs might have different results.

Source code comments

The source code should have at the top of each file a comment showing the team name and tutorial group and group members, e.g.

```
// Xylo TT5L
// Frank Phang Tzia Wen
// Zarina Aminah binti Muhammad Khairul
// Telagarani a/p Pannir Jegadasan
// Tommy Richard anak Peter
```

In addition to normal things like explaining the purpose of the code, etc. also comment as to who wrote a particular piece of code, method, or even part of a method, if more than one person made changes to a method. If you did pair programming for a particular piece of code, write both names.

Object Oriented Requirements

Your programme must utilise a proper OOP structure. You cannot simply have one giant main.cpp file. You must have abstract classes for polymorphism and proper object-oriented design. You must use encapsulation, inheritance, polymorphism, function redefinition and/or overriding and operator overloading.

You must submit an **accurate** UML Class in a format that is clear and easy to read, detailing all the classes, attributes, and methods, including their visibility. If your class diagram does not match your code, **you will lose marks**. If your class diagram does not use the correct UML notation, you will also lose marks.

Marking Rubrics

UML Class Diagram Notation & Clarity

Score	Description
4	Diagram is done excellently, laid out well, and easily understood at a glance.
3	Diagram is clear and uses proper UML notation.
2	Diagram mostly follows proper UML notation.
1	Diagram is severely flawed.
0	No UML class diagram

UML Consistency with Code

4	The code structure mirrors the diagram exactly.
3	High consistency, but minor discrepancies exist.
2	Significant differences found.
1	The diagram appears to describe a completely different programme.
0	No UML class diagram

Encapsulation

3	All domain data is strictly private with proper controlled access.
2	Inconsistent protection, .e.g. leakage due to unprotected pointers being returned via accessors, etc.
1	Poor encapsulation. Much is exposed.
0	No encapsulation.

Inheritance Structure

4	Elegant hierarchy. A clear abstract base class defines the interface and handles common logic. Derived classes only contain specifics relevant to their type. Constructors initialise parent data correctly.
3	Good hierarchy, but some logic is duplicated across derived classes that should have been moved to the parent class.
2	Hierarchy exists, but is poorly designed. The distinction between base and derived responsibilities is blurred, or the parent class is merely an empty shell.
1	Syntax of inheritance is used, but logically the classes are unrelated or the inheritance makes no sense.
0	No inheritance.

Polymorphism

4	The simulation loop interacts purely with generic pointers/references. Subclass-specific behaviours execute automatically without manual type-checking.
3	Polymorphism is used correctly for the main logic, but there are minor instances where the code unnecessarily checks the specific type of an object.
2	Polymorphism is attempted, but the main loop relies heavily on if-else or switch statements to determine which function to call for different types.
1	“Fake” polymorphism. The code casts pointers back to their specific derived types before using them.
0	No polymorphic call at all.

Function Redefinition and/or Overriding

4	Redefinition and/or Overriding is implemented and used meaningfully. The distinct signatures clearly handle different data types or parameter counts.
3	Redefinition and/or Overriding is present and correct, but the use case is somewhat trivial or could have been handled simply with default parameters.
2	Redefinition and/or Overriding is implemented but causes ambiguity or is used confusingly (e.g., two functions doing unrelated tasks share the same name).
1	Syntax is incorrect or the attempt at Redefinition and/or Overriding results in compilation errors that were commented out.
0	Not implemented.

Operator Overloading

3	Done correctly, and also makes sense in the functionality of the program.
2	Works, but doesn't really make sense in the functionality of the program.
1	Attempted but doesn't work.
0	Not implemented.

Battle Simulation Logic

3	Flawless simulation. Correctly handles simultaneous fire, and targeting is random as required.
2	Logic flaw: Unfair advantages like first strike exist. Loop works otherwise.
1	Simulation runs but behaves strangely (e.g., ships resurrect, infinite loops, or crashes when a side is wiped out).
0	Simulation does not run.

Implementation of Battle Rules

4	All specific game rules (e.g., spreading out the available crew to as many ships as possible, evasion penalties for under-crewed vessels, different hit probabilities for different weapon types) are calculated accurately.
3	Most rules are correct, but one edge case is missed (e.g., the penalty is applied to the wrong stat or the probability math is slightly off).
2	Basic rules are there, but complex conditions (like the under-crewed penalty) are ignored or hardcoded incorrectly.
1	Mathematical logic is largely incorrect. The simulation ignores the stats provided in the scenario.
0	No rules implemented.

Input Handling & File I/O

4	Robust I/O. Reads filenames strictly from command line arguments. Parsers correctly handle the specific file formats and link related data (e.g., Crew to Ships) and recovers gracefully on invalid data.
3	Functional I/O, but lacks error handling or has minor issues mapping crew to ships. Errors are not written to the standard error output, or the program does not recover gracefully and continues on invalid data.
2	Ignores command line arguments, or parsing is fragile and breaks easily if the file format varies slightly.
1	Hardcoded data. The programme expects specific file names or cannot parse the text files at all.
0	No file handling at all.

Code Quality & Attribution

3	Proper spelling and indentation. Every method is clearly attributed to specific author(s). Variable names are descriptive and meaningful.
2	Inconsistent style, or mixing naming conventions, or attribution is missing from several key functions.
1	"Spaghetti code" or poor indentation, or meaningless variable names (e.g., var1, x), or no attribution comments whatsoever.
0	Unreadable or obviously generated without review.

Total: **40 marks**

Deliverables

Submit 1 zip file containing:

- Source code in one file (For example `XyloTT9L.cpp`).
  - We must be able to just compile this cpp file and it will run. Do not give us your executable file – we will compile it ourselves with a standard C++ compiler, hence the requirement that you use standard C++.
  - Make sure you test that will compile from the command line and not only from your IDE, e.g. g++ `XyloTT9L.cpp -o XyloTT9L`
  - Do not include the data files – we will test with other data files than the sample ones we gave you.
- Design document containing class diagram and explanations in PDF, DOCX, or ODT, with a header like this:

> CCP6124 Project
>
> Trimester 2530
>
> *by <<TEAM NAME>> <<Tutorial Group>>*
>
> Team Leader: Name, Student ID, phone number, email
>
> Team members:
>
> Name, Student ID, phone number, email
>
> Name, Student ID, phone number, email
>
> Name, Student ID, phone number, email

Submission Deadline

Wednesday 5pm 28 January 2026

Late policy: 10% will be deducted if the project is submitted on 1 day late. 20% will be deducted if it is 2 days late. 30% will be deducted if it is submitted 3 days late. 40% will be deducted if it is submitted 4 days late. No submissions will be accepted after that. So, for example, if your mark is 30 out of the possible 40, and it's 2 days late, it will be 30×0.8=24.

Interview

Note that all the marks can be **adjusted** by the interview. If, during the interview, it is shown that you do not know the code and how it works, for example, because someone else, or an AI, did the work, then you will **fail**. If we determine that you only did a little bit of the work and someone else did much more of the work, your marks will be decreased. So, you may Google and ask AI to explain things to you, but **write the code yourself**.