CITS 3002
Computer Networks

THE UNIVERSITY OF WESTERN AUSTRALIA

Project

Battleships

Engage in Explosive Rivalry

CITS 3002
Computer Networks

Due date : 19 May 2025 11.59pm
Submission : via LMS
Weight : 30%
Format : Group of 2 people*

Deliverables : (1) Report, (2) code file(s), and (3) demo video

*In case you cannot find your partner, you can do the project by yourself (the scope of the project is feasible to do by one person). However, the marking will be done as if it was done as a group regardless, and it will be treated as a group project for the purpose of UAAP and special considerations – as it is originally intended to be done as a group project.

Introduction

You have been hired by *Socket & Sunk* to develop a networked, turn-based Battleship game named "Battleships: Engage in Explosive Rivalry", or simply BEER. Although an early prototype exists for the client, the overall project lacks a functioning multiplayer server to orchestrate real-time gameplay across multiple connections.

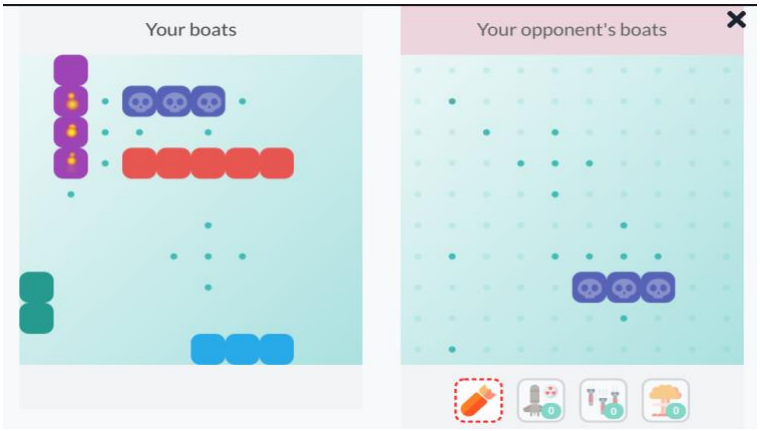
Your task is to **implement the Battleship server and client(s)** that can handle players, manage game states, broadcast results, and deal with typical network edge cases. You may use any programming language, but Python or C/C++ is recommended.

BEER Overview

In the standard Battleship board game:

1. Each player secretly places a set of ships on a 10×10 grid.
2. Players take turns "firing" at specific grid coordinates (e.g., B5).
3. The server announces whether each shot is a *hit*, *miss*, or *sunk* a particular ship.
4. The game ends when one player's entire fleet is destroyed.

BEER also follows the above general gameplay.



CITS 3002
Computer Networks

Tasks

You must implement a server that coordinates at least two players (clients) in BEER match, ensuring proper game flow (ship placement, turn-taking, tracking hits/misses). To get started, you are provided with:

- **battleship.py**: This module defines essential constants and gameplay logic which is shared by both the client and the server. You can run this module using `python battleship.py` to play a local, single-player version of Battleship to see the expected gameplay loop.
- **server.py**: This module implements a basic server that allows a single (1) client to play a single-player version of Battleship with no other participants, and very little error checking.
- **client.py**: This module implements the game client with a minimal implementation for connecting to the server.

The server and client implementations are currently barebones and likely be buggy. Hopefully you will be able to either fix them up or re-implement your own server and client modules. To guide you in this, we provide **four tiers** of tasks to complete. Solutions which meet Tier 1 requirements are minimally viable, higher tiers offer more granulated mark allocations for solving more complex problems.
We will only assess **a single submitted implementation** (i.e. the tiers *stack* on each other rather than implementing the requirements of each separately).

Tiered Requirements

We structured this project in tiers of increasing complexity. Achieving Tier 1 demonstrates a basic solution, while higher tiers add robustness and polish. Aim to complete as many as possible.

CITS 3002
Computer Networks

Tier 1 – Basic 2-Player Game with Concurrency in BEER

Aim: Enable two players to connect and play BEER in a straightforward, turn-based manner.

T1.1 Concurrency Issues to Fix

- Your current BEER client has a critical issue with message synchronization, even in single-player mode (but still online with the server). When you run the client and play the game, you'll notice:
  - Server responses appear out of order or delayed.
  - Prompts (like "Enter coordinate") only show up after you've already provided input.
  - Error messages refer to previous inputs rather than your most recent command.
- For example, this interaction shows the problem:

```
Welcome to Online Single-Player Battleship! Try to sink all the
ships. Type 'quit' to exit.
>> ok?

[Board]
1  2  3  4  5  6  7  8  9 10
A  . . . . . . . . . .
B  . . . . . . . . . .
C  . . . . . . . . . .
D  . . . . . . . . . .
E  . . . . . . . . . .
F  . . . . . . . . . .
G  . . . . . . . . . .
H  . . . . . . . . . .
I  . . . . . . . . . .
J  . . . . . . . . . .
>> what do i do now?
Enter coordinate to fire at (e.g. B5):
>> A1
Invalid input: invalid literal for int() with base 10: 'K?'
>>
```

- Notice how "Enter coordinate" only appears after you've already typed your next input, and the error message doesn't match what you just entered.
- This problem occurs because of how the client handles I/O: the client reads one message from the server, then waits for user input, then reads the next message from the server. But the server might send multiple messages in sequence before expecting input.
- To fix this issue, modify the client to use concurrency (threading) to separate the receiving and sending operations:
  - Create a thread that continuously reads and displays server messages
  - Keep the main thread for handling user input and sending commands
- For a complete solution, you'll also need to make the server handle multiple clients by accepting connections in a loop and processing each client in its own thread
- If you're unfamiliar with Python's threading library, you can refer to the official documentation at https://docs.python.org/3/library/threading.html. Pay special attention to the Thread class, thread coordination with flags, and basic thread safety considerations.

CITS 3002
Computer Networks

T1.2 Server and Two Clients

- Your server should accept connections from exactly two players. Once two players join, the game begins.
- You do not need to handle more than two clients at this stage.

T1.3 Basic Game Flow

- Implement standard Battleship mechanics: each player places ships, then they alternate firing at coordinates.
- Report “hit” or “miss” outcomes and detect when a ship is fully sunk.
- End the game when one player has all ships sunk.
- When the game ends, either by a player sinking all the other player's ships, or by a player forfeiting (the forfeit behaviour will be detailed in further tasks), the server may simply close all connections (at least for this Tier).

Note: The provided `battleship.py` script will come in handy, and can be edited as necessary.

T1.4 Simple Client/Server Message Exchange

- You may define minimal messages (e.g. `PLACE A1 H BATTLESHIP`, `FIRE B5`, `RESULT MISS`, etc.) or use a structured approach (JSON, etc.).
- At this tier, you can **assume** clients send valid commands in sequence (no malicious or malformed inputs).

T1.5 No Disconnection Handling

- You can assume that connections remain stable throughout the game, with no unexpected dropouts.
- If a player disconnects, it can simply end the current game or the server can shut down.

CITS 3002
Computer Networks

Tier 2 – Gameplay Quality-of-Life & Scalability

Aim: Extend your basic solution to be more robust and capable of running multiple games, one after the other.

T2.1 Extended Input Validation

- Players might send invalid commands (wrong coordinates, out-of-turn firing).
- Your server should gracefully respond with an error message or ignore these commands, rather than crashing.

T2.2 Support Multiple Games

- After a game ends, your server should be able to start a new game with the same players or with new connections, without needing to restart the server process.
- Optionally, add a small delay or confirmation step so players can see the final result before a new game starts.

T2.3 Timeout Handling

- Implement a timer that tracks when a player last submitted a valid move.
- Introduce a modest inactivity timeout (e.g., 30 seconds) where a player forfeits or the turn is skipped if they exceed the timeout period.

T2.4 Disconnection Handling

- Your server should detect when a socket connection is closed or broken.
- If a player disconnects mid-game, treat it as an immediate forfeit or handle it gracefully (e.g., skip their turn).
- The server should remain stable and continue running (potentially waiting for new players to start a new match).

T2.5 Communication with Idle or Extra Clients

- If there would be more than two total connections (e.g., 3 clients), you can either reject additional clients or place them in a “waiting lobby” until the current game ends.

CITS 3002
Computer Networks

Tier 3 – Multiple Connections

Aim: Handle more complex network scenarios: multiple concurrent connections, spectators, and partial disconnects.

T3.1 Multiple Concurrent Connections

- Your server can accept more than two clients at once.
- Exactly two of them are active players in any single match, and any extra clients become *spectators*.

T3.2 Spectator Experience

- Spectators are not active players but should still receive real-time game updates:
  - Board update announcements.
  - Shots fired, hits/misses, and the final outcome.
- Any command or input from a spectator (like `FIRE`) should be ignored or produce an error.

T3.3 Reconnection Support

- Design a simple mechanism to identify returning players (such as a username or client ID provided at initial connection)
- If a client disconnects mid-game (e.g., due to a network drop, or by them quitting), allow them to reconnect within a short timeframe (e.g., 60 seconds) and resume control of their existing board.
- You should maintain the game state during this reconnection window so they can continue from where they left off.
- Clearly specify how you handle the scenario if the other player has already won or if the reconnection window is exceeded.

T3.4 Transition to Next Match

- When the game ends, the server should have a defined method for selecting the next two players from all connected clients.
- This could be first-come-first-served from the waiting lobby, or another selection method you design.
- The server should clearly communicate to all clients who will be playing in the next match and when it will begin.
- Messages from spectators or disconnected players should not disrupt the current match.

CITS 3002
Computer Networks

Tier 4 – Advanced Features

Implement additional capabilities to address real-world networking concerns or enhancements. Complete **TWO OR MORE** of the below tasks, where one of the tasks must be T4.1 Custom Low-Level Protocol with Checksum. More tasks you complete, better marks you could receive for this tier. Provide details in your report as indicated.

T4.1 Custom Low-Level Protocol with Checksum

- Move beyond simple send/recv usage of TCP or UDP by crafting your own packet format and verifying data integrity.
- Define a header (e.g., containing sequence number, packet type, game-specific fields, and a checksum).
- Implement a checksum mechanism (e.g. CRC-32 or a simpler sum-based approach) to detect corrupted packets on receipt.
- Decide on your error-handling policy if a packet fails the checksum (discard, request retransmission, etc.).
- In your report, you must include:
    - A clear specification of the packet structure (fields, sizes, arrangement).
    - How you generate and verify checksums.
    - What your protocol does with corrupted or out-of-sequence packets.
    - Statistical demonstration (optional but recommended): E.g., artificially inject errors or scramble bits, measure how many packets get flagged as corrupted.

T4.2 Instant Messaging (IM) Channel

- Add a chat system so players and spectators can send text messages in real time.
- Provide at least one command in your protocol (e.g. CHAT <message>) that broadcasts <message> to all connected participants.
- If you use your low-level protocol from above, define a CHAT packet type or command ID.
- Ensure concurrency is managed: multiple chat messages might arrive during a turn, or in parallel from spectators, etc.
- The server (or a dedicated chat thread) must route messages to all active clients.

Note: This task will require more depth than you might initially expect. How do you track who's message is who's? Do clients now need identifiers (usernames, etc.)? Think carefully about not just the networking aspects, but also the user experience.

T4.3 Encryption Layer

- Implement a basic **symmetric** encryption scheme for your protocol, e.g. AES in CTR mode, or a simplified approach with a shared secret key.
- Define how you exchange keys (if at all) or assume an out-of-band channel.
- Show how you incorporate encryption into your custom packet structure from T4.1 (e.g., encrypting the payload, or each packet's data, ensuring you still can verify checksums or use an integrity-protected encryption scheme).
- Report how you handle replay attacks, partial packet corruption, or IV generation (if using a block cipher).

CITS 3002
Computer Networks

T4.4 Security Flaws & Mitigations

- Analyse potential security vulnerabilities in your Battleship implementation (e.g., session hijacking, impersonation, replay attacks).
- Demonstrate at least one way your current design could be exploited. For example, capturing a valid session token in transit and replaying it, or forging coordinates if there's no authentication.
- Implement at least one robust solution to address these issues (e.g., embedding session tokens in an encrypted or checksum of the payload, adding sequence numbers/timestamps).
- Test and demonstrate that the exploit is now blocked or significantly mitigated.
- In your report, discuss the security flaws you identified and explain your approach to fixing or mitigating them (protocol-level changes, token-based checks, encryption, etc.). Show how you tested the exploit scenario again and verified your fix.

CITS 3002
Computer Networks

THE UNIVERSITY OF WESTERN AUSTRALIA

Deliverables

Three deliverables:

- Report in PDF, submission via LMS
- Code in ZIP, submission via LMS
- Demo video in publicly available platform, a link shown clearly in the report.

Your will prepare and submit a report outlining the details, explanations and justification of your design choices in your implementations (especially for Tier 4 tasks). You will also submit your code(s) as a single zip file, with a README file clearly outlining how to use your code (especially if you are not using Python or C). We will make an effort to run your code and debug minor errors, but any significant errors not enabling us to run the code as you outlined in the README file would not receive ANY MARKS for what you claim it does. We suggest you give the code and instructions to your friends to see whether they are able to follow the instruction and run it as you have intended.

The report and the zip file should include both of your student IDs, followed by the word "BEER". The order does not matter. For example:

22221234_22224321_BEER.pdf
22221234_22224321_BEER.zip

Please note, the report format MUST BE PDF.

Any other files and formats submitted **will be IGNORED** (no leeway, no exceptions, no excuses, no begging).

You will also submit a demo video of your game, highlighting key features you implemented in each tier. This video should not be more than 10 minutes. You can upload it to your preferred hosting site (e.g,. YouTube), and include the link to the video in the report within the first two pages.

Final reminder:
Try to make your report succinct and to the point. An unnecessarily long report is not a high-quality report. The same applies to your demo video. As a guide, you can prepare a report that is no longer than 10 pages, including everything (e.g., images, figures, tables and references). Unnecessarily long report will not get you additional marks, but rather not get marks for missing clarity in the description. The demo video should focus on demonstrating the features implemented for achieving the outlined tasks. You may edit the video to cut out unnecessarily long pauses or descriptions.

Submission

All submissions are made via LMS.
Only one member submits the project deliverables.
No other submission methods are accepted.

CITS 3002
Computer Networks

THE UNIVERSITY OF WESTERN AUSTRALIA

Rubric

Component	N	P	CR	D	HD
Tier 1 (40%)	The BEER game cannot be played by two players.	The BEER game is played by two players, but there are still some minor concurrency issues.	The BEER game is working as intended.	[CR] plus; Code is well documented. The game is easy to play and intuitive.	[D] plus; Code follows industry best-practice coding style guidelines. Good design has been used to implement the game.
Tier 2 (15%)	Tier 2 requirements are not adequately implemented.	At least three requirements are met. Other requirements may not fully function.	At least four requirements are met.	(1) All tier 2 requirements were implemented correctly. (2) Design and implementations are of high standard.	[D] plus: The details of design choice, implementation and justifications, experimental design are presented.
Tier 3 (15%)	Tier 3 requirements are not adequately implemented.	At least three requirements are met. Other requirements may not fully function.	At least four requirements are met.	(1) All of tier 3 requirements were implemented correctly. (2) Design and implementations are of high standard.	[D] plus: The details of design choice, implementation and justifications, experimental design are presented.
Tier 4 (20%)	Tier 4 requirements are not adequately implemented.	Task 4.1 is correctly implemented. Other tasks in this tier may not fully function.	[P] plus; (1) Another task implemented and is functional. (2) Discussions of the design choices, implementation, justifications are provided in detail.	[CR] plus; (1) Another task implemented and is functional. (2) Comprehensive overview of the tasks are given with in-depth understanding of advanced networking skills and details.	[D] plus; (1) Another task implemented and is functional. (2) Demonstrate advanced knowledge in computer networks and its relevance to the project.
Report and Demo (10%)	(1) The demo did not show the completed tasks. (2) The report did not match the demo. (3) The report formatting is poor and difficult to read.	(1) The demo somewhat showed the completed tasks. (2) The report somewhat matched the demo. (3) The report formatting is reasonable to read and understand.	(1) The demo showed the completed tasks adequately. (2) The report matched the demo. (3) The report formatting is clear and well layed-out.	(1) The demo showed the completed tasks in high quality. (2) The report is of high quality and matched the demo. (3) The report is structured and sections well defined to clearly demonstrate implementations, justifications and discussions.	(1) The demo showed the completed tasks in professional quality. (2) The report is of professional quality and matched the demo. (3) The report formatting is outstanding, with clear points of the implementations, justifications and discussions.

You do **not** have to implement features strictly in the order listed, but Tiers must be satisfied for full marks.
It's possible to partially complete a tier and move on, but partial credit is given only if earlier features are stable.
Start early, focus on **networking fundamentals**, and ensure your program runs without crashes or deadlocks.