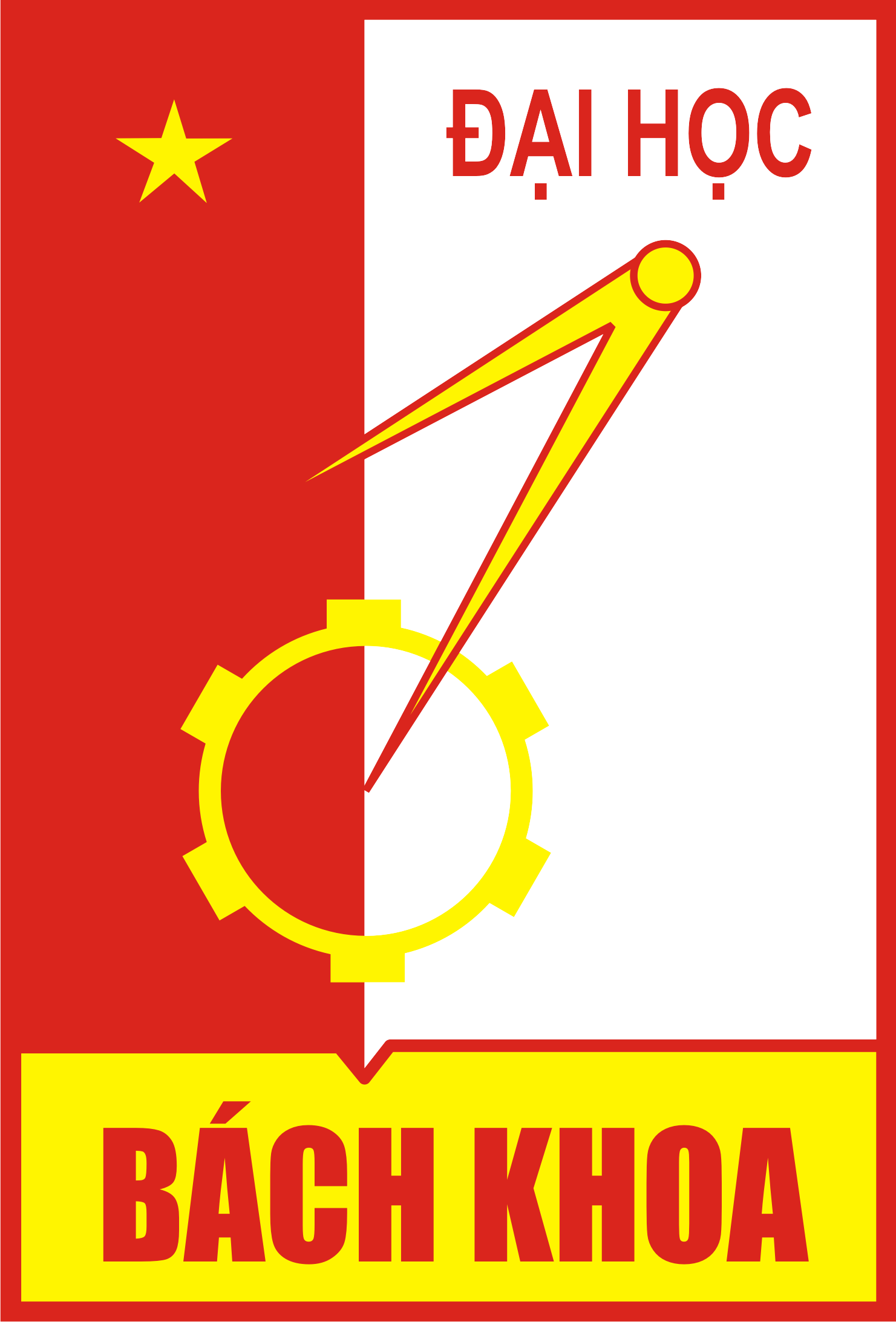
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and Communications Technology



# **Capstone Project Report:**

# **Battleship Online**

Course:

NETWORK PROGRAMMING

Supervisor: Assoc. Prof. - PhD. Truong Thi Dieu Linh

GROUP 8:

Lê Thanh Sơn – 20176860

Đào Cao Duy – 20176738

Trần Phi Hùng – 20176774

**Table of content:**

1. **Introduction**
2. **Objective**
3. **Game Rude**
4. **Game Architecture**
5. **Protocol Design**
6. **Progress update**
7. **Introduction.**
8. **Objective**

This documentation comprehends the description of our project, game “Battleship Online Project” application, from the game rule, the architecture, functionalities to the protocol design. This would be essential for everyone to understand what we have made for this project.

The objective of this project is to help us understand and create a small network server to host a game throught the functionalities of a client and a server.

1. **Game Rule**

***Rules for BattleShip***

***Game Objective:***

The object of Battleship is to try and sink all of the other player's before they sink all of your ships. All of the other player's ships are somewhere on his/her board. You try and hit them by calling out the coordinates of one of the squares on the board. The other player also tries to hit your ships by calling out coordinates. Neither you nor the other player can see the other's board so you must try to guess where they are. Each board in the physical game has two grids: the lower (horizontal) section for the player's ships and the upper part (vertical during play) for recording the player's guesses.

***Starting a New Game:***

Each player places the 10 ships somewhere on their 10 x 10 board. The ships can only be placed vertically or horizontally. Diagonal placement is not allowed. No part of a ship may hang off the edge of the board. Ships may not overlap each other. No ships may be placed on another ship or adjacent to each other.

Once the guessing begins, the players may not move the ships.

The 10 ships are: 1 Carrier (occupies 4 spaces), 2 Battleships (3 spaces), 3 Cruisers (2 spaces) and 4 Destroyers (1 space).

***Playing the Game:***

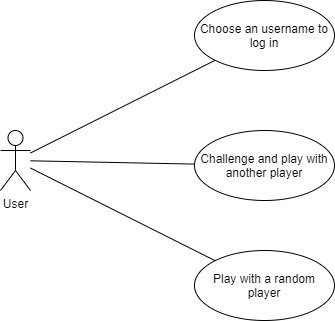
Player's take turns guessing by calling out the coordinates. The opponent responds with "hit" or "miss" as appropriate. Both players should mark their board with pegs: red for hit, white for miss. For example, if you call out F6 and your opponent does not have any ship located at F6, your opponent would respond with "miss". You record the miss F6 by placing a white peg on the lower part of your board at F6. Your opponent records the miss by placing.

When all of the squares that one your ships occupies have been hit, the ship will be sunk. You should announce "hit and sunk". In the physical game, a red peg is placed on the top edge of the vertical board to indicate a sunk ship.

As soon as all of one player's ships have been sunk, the game ends.

3. Application Features:

We intend to develop the game with three main functionalities, which are shown in the usecase diagram:



1. **Game Architecture**

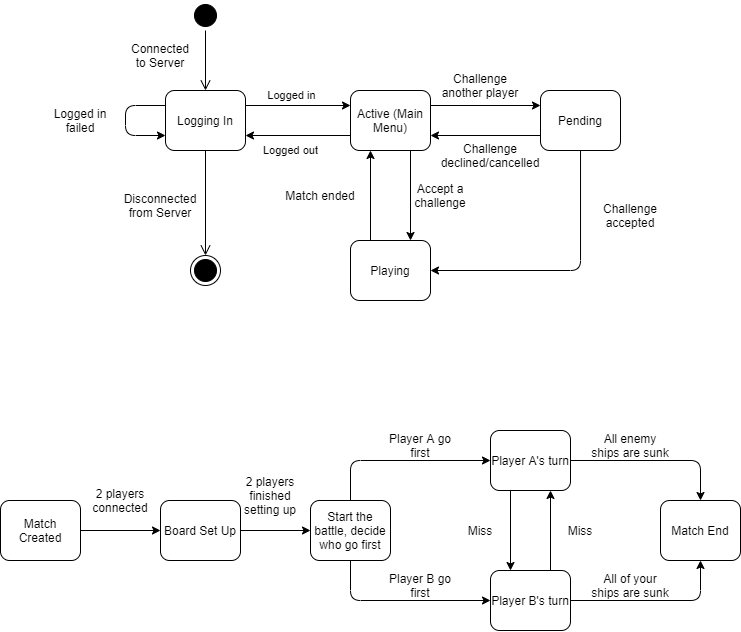
***Network Architecture:*** Client/Server (multiple clients connecting to server and playing against each other)

***Message type between Server & Client:*** String

***System design:*** The application processes depend on many states of the client and the game. Below are the two state machine diagrams showing the transition between the states in the system. In each state, the server and the clients will use a different set of messages to communicate each other (which will be presented in details in Protocol Design section of the report).

+ First diagram: State diagram of a client connecting to Server and using its services.

+ Second diagram: State diagram of the game system in the Server controlling the process of a battleship match between 2 players.



1. **Protocol Design**

**1. Login**

Client connects to Server, enters an username and sends to Server:

**(Client)** login: [username]

Server receives and check **[username]** (which must include only alphabet characters and digits). Then send a message replying if it is valid or not to client. If there is also some error logging in the game, the server will reply with a different message.

+ valid username → (**Server)** login: 1

+ invalid username → **(Server)** login: 0

+ error (too many clients) → **(Server)** error: 11

+ error (service unavailable) → **(Server)** error: 10

User has to enter another username if the previous one is invalid. After entering a valid one and managed to login, client (with an username) enters the game network. While being active in the network, the game server will send the list of online users to users so they can invite each others.

The Server will add a nametag after usernames to distinguish between similar ones.

**(Server)** userlist: [username1]#[tag]/[state] , [username2]#[tag]/[state] , ...

example: userlist Josh#20/A, Paimon#33/P, Josh#69/A, Steve#20/P.

(User state: Active (**A**), Playing (**P**) )

In game lobby Client may: log out, challenge another user, accept/decline a challenge, find a match against random opponent, refresh user list. The messages respective for those functions are:

+ **(Client)** logout

+ **(Client)** challenge: [username]#[tag]

+ **(Client)** accept: [username]#[tag]

+ **(Client)** decline: [username]#[tag]

+ **(Client)** random

+ **(Client)** getlist

Server sends the invitation to the challenged user (with username and tag of challenging user):

**(Server)** invite: [username]#[tag]

When a challenge is accepted, Server sends a message to both clients to start a match:

**(Server → 2 Clients)** matchstart

When a challenge is declined, Server sends a message to the challenging clients:

**(Server)** decline

In the case when a match is about to start, if one of the two players is missing (disconnected), the server notices the other player and return him back to game lobby:

**(Server)** error: 21 (player not found)

When two players are randomly chose to play against each other, the server will also add the opponent’s username and tag to the message:

**(Server → 2 Clients)** matchstart: [opponent’s username]#[tag]

When a match is created between 2 players, they will set up ship formation on their client side and send to server:

**(2 Clients → Server)** setup: [ship 1 info], [ship 2 info], … , [ship 10 info].

The grammar for [ship info]:

**[Horizontal/Vertical]–[Ship length]–[Coordinate X]–[Coordinate Y]**

Example: **(2 Clients → Server)** setup: V-4-3-10 , H-3-5-2 , … , H-1-8-8 .

After both players send their ship formations, the server send messages to 2 players and start the game with one of the two players playing his/her turn first:

**(Server → 2 Clients)** gamestart: 1 ( **1**: go first, **2**: go second )

In-game: Players in their turn send coordinate information to shoot each other’s board to Server, Server process and send result to both players. Message grammar and example:

**(Client 1)** fire: [coordinate x]-[coordinate y]

**(Server → Client 1)** [hit/miss]

**(Server → Client 2)** fire: [coordinate x]-[coordinate y]-[hit/miss]

**(Client 1)** fire: 4-4

**(Server → Client 1)** miss

**(Server → Client 2)** fire: 4-4-miss

When all the ships of a player sunk, Server sends result message to 2 players. Then both players return to game lobby.

**(Server → Client 1)** matchend: win

**(Server → Client 2)** matchend: lose

While the match is progressing, if any player is disconnected, the server will notice the other player to get a victory and end the match.

**(Server)** matchend: opponentleft

1. **Progress update (17/5/2021)**

Programming language used for the project: Java (External library: JavaFX)

1. Server and game system: (progress: 25%)

+ The current version of server-client network is using the synchronous type, will change to asynchronous type later.

+ Currently able to help clients log in; able to store, update and send clients the list of online users and their current state.

+ The game system/gameplay is not yet implemented, but some modules are already built, being developed and tested.

2. Client and GUI: (progress: 20%)

+ The GUI displaying different windows is being developed and mostly finished for the main parts (Window layouts and game images). However, the resources used might get changed later in the future.

+ The request messages from client side are still being developed one by one.