CS 342 Software Design - Professor Evan McCarty

Spring 2024

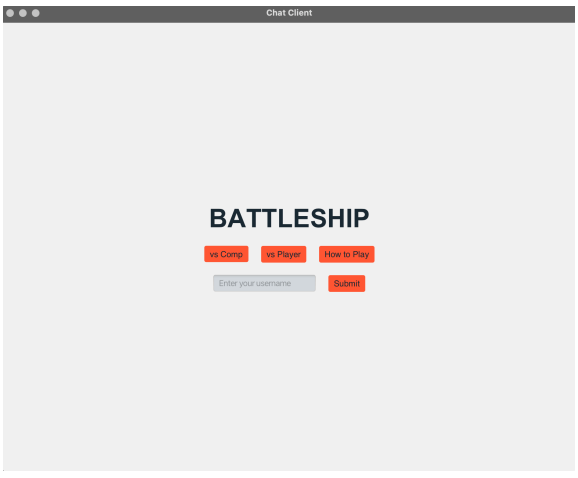
Team Members: Rahin Jain (665219123), Aarav Surkatha(667714562).

Contributions:

Rahin and Aarav collaborated on a project where their contributions were essential in different phases. Rahin was responsible for creating the wireframe and setting up the server class diagram and the board class, along with its supporting classes. He also handled the initial server testing, front-end coding, and documentation. Aarav contributed by reviewing and modifying the wireframes and class diagrams initially created by Rahin. Aarav took charge of creating the client class diagram, setting up the server code, and conducting the initial client testing. Aarav also led the creation of the final test. Both Rahin and Aarav worked together on coding and debugging, as well as final testing and making necessary changes to ensure the project's success.

Run through of application:

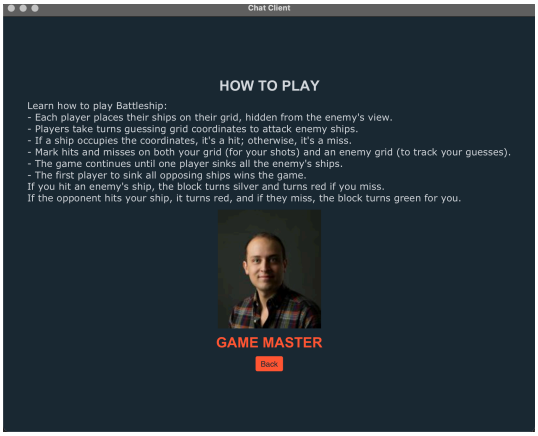
Home Page:

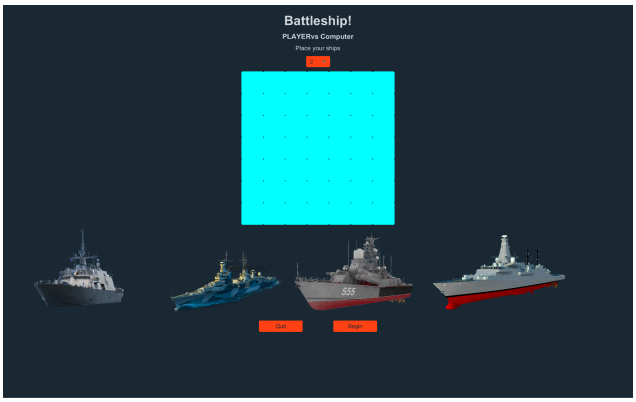


The homepage allows player to set up the initial components of the game. Button1: vscomp is for players to play against the Computer with the AIbot. Button2: vsPlayer is to play against an online player that is selected by the game. The home page does not allow the players to proceed without submitting their names first. Only then can players pick a computer or another player to start the game.

How to play page:

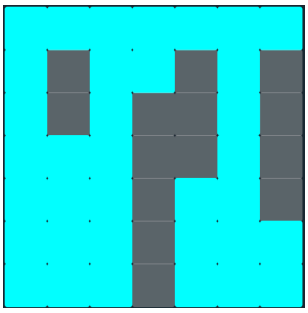
On the home page, the players have the option to click how to play, this explains the basic rules for the game and some simple tips and techniques.





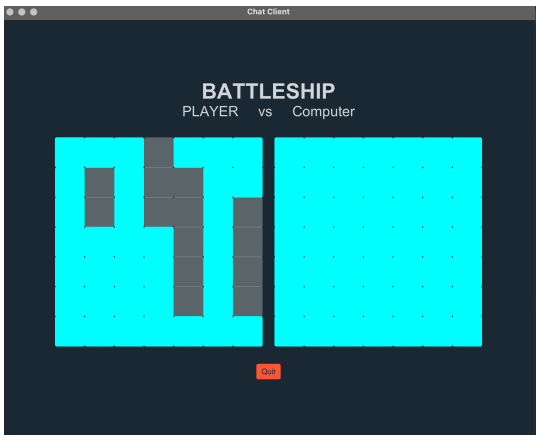
Vs computer ship setup page:

Players are allowed to place 4 ships exactly, 1 of each category i.e of size 2,3,4,5. They can only proceed once All 4 ships are placed on the board.

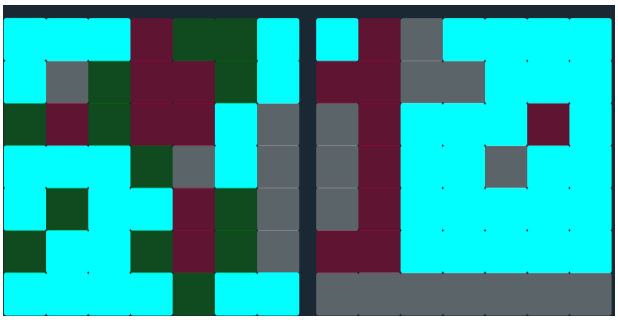


Vs computer game page:

The play page shows 2 girds , on the left player can see their own board set up and on the right they see the opponent's board which is initially hidden and it reveals the player clicks the grid buttons.The buttons on the player grid are disabled and not clickable. As the

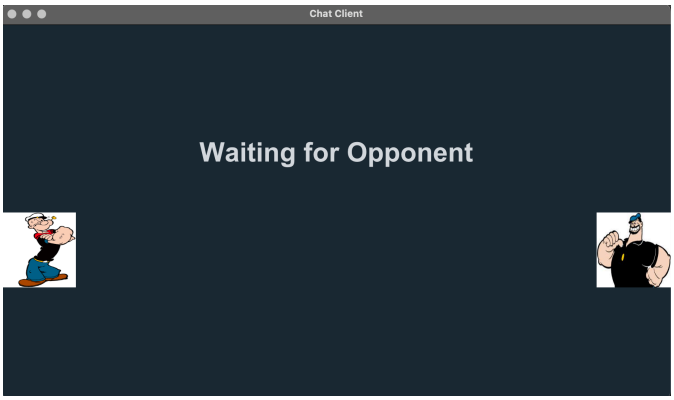


The player starts the grid as cyan(water) and as the player clicks, the button turns gray if there is a ship and red if it is a miss. On the other hand if the opponent hits their ship it turns red and if the opponent misses it turns green for the player.



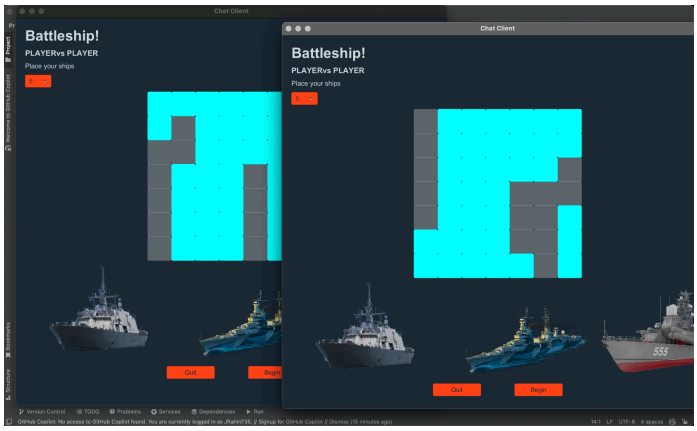
Waiting screen:

When a player chooses PvP mode but there is no ready player available who also chose PvP then it puts up a waiting screen while another player joins the server and becomes ready to play.

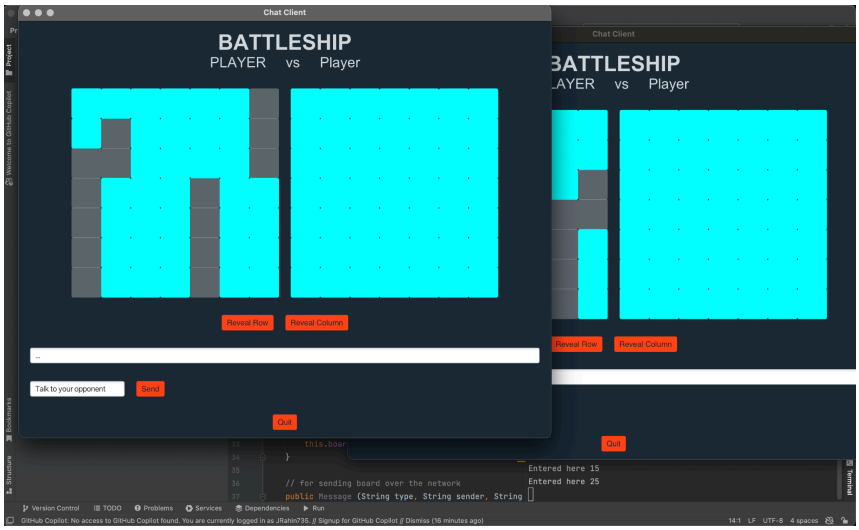


pvp ship setup screen:

Similar to the set up screen in Vs computer mode, both the players get the same screen with the same conditions mentioned previously. Each player gets to set up 4 ships, 1 of each size and game can begin using the begin button below once both the players have set up their grid with all the ships. (game does the error checking itself to not allow ships to be placed onto each other and/or out of the bounds of the grid.

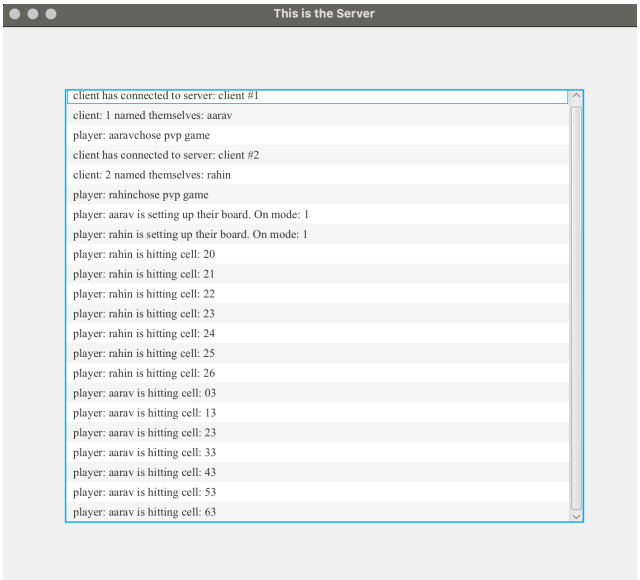


pvp game screen:



Both player's game screens get updated simultaneously as one of them plays their turn. Similar to the computer rules they have the color of their grid cell changes to red if their ships are hit and green if the opponent misses. The color for the opponent's grid changes on the player screen to gray if there is a ship hit and red if they miss the opponent's ship. Additionally, players also get two power ups explained later and also they get a chat box to message the opponents also explained later in the report.

Functionalities:



Server logs: The serverGUI of the project maintains a log of every message received from a client and sent to a client. This includes play moves, game creation, join notices, etc. This feature was helpful during debugging and can also be used to ensure correct transfer of data from server to client.

AI bot:

The `BattleshipAI` class implements an AI for playing Battleship, managing its own and the opponent's game boards, and making strategic decisions based on the game state. It has two main modes of operation: "hunting" mode and "targeting" mode.

1. Board Initialization: The AI uses two `Board` instances to track the state of its own and the opponent's boards. Ships are placed randomly on its board using `placeShipsRandomly`, ensuring no overlap or out-of-bounds placement.

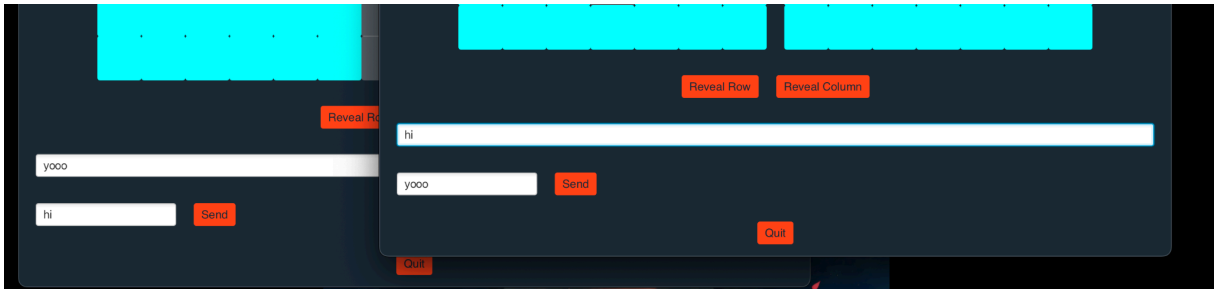
2. Target Selection: During its turn, the AI chooses where to attack on the opponent's board. If it's not in "hunting" mode, it selects a random target that hasn't been hit before. If a previous hit was successful, the AI switches to "hunting" mode, focusing on adjacent cells to potentially hit more parts of a ship.

3. Shooting and Strategy Adjustment: Upon shooting at a target, if a hit occurs, the AI stores adjacent cells for future targeting, enhancing the chance of hitting the remaining parts of a ship. Conversely, if it misses, it exits "hunting" mode.

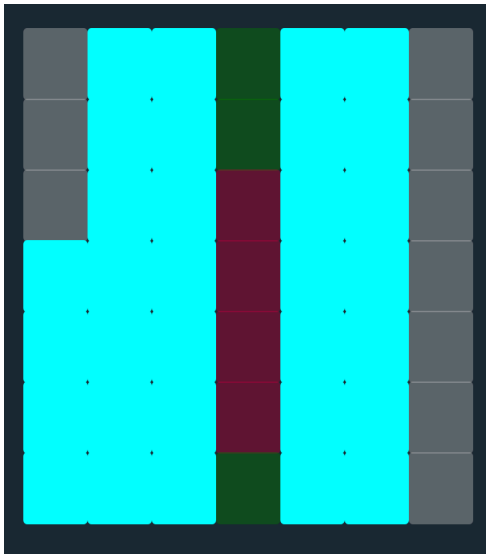
This dual-mode approach helps the AI switch between searching for ships and methodically taking them down once found, making it more effective and adaptive in gameplay.

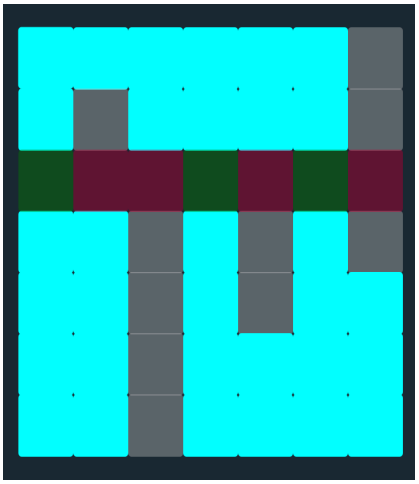
Additional Features -

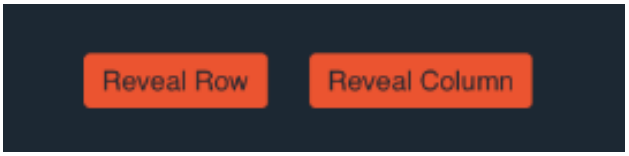
Messaging Service: Messaging feature has been introduced in the PVP game allowing players to chat with their opponents directly during matches, aiming to increase interactivity and engagement, encouraging additional connections and dynamic gameplay experience.



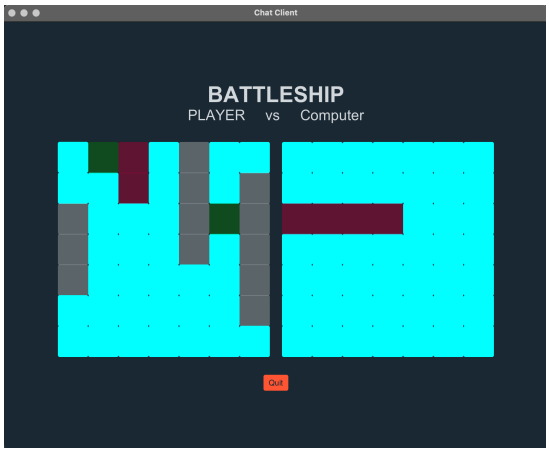
Power Ups!: In the game, each player has the option to target an entire row or column on their board at least once during the match. This feature is intended to make the gameplay more engaging and dynamic. It encourages players to employ new strategies, adding depth to the game and enhancing the overall experience. To make things interesting and more challenging these powerups choose random rows or columns to be revealed so the player cannot have any unfair advantages but have exciting elements.

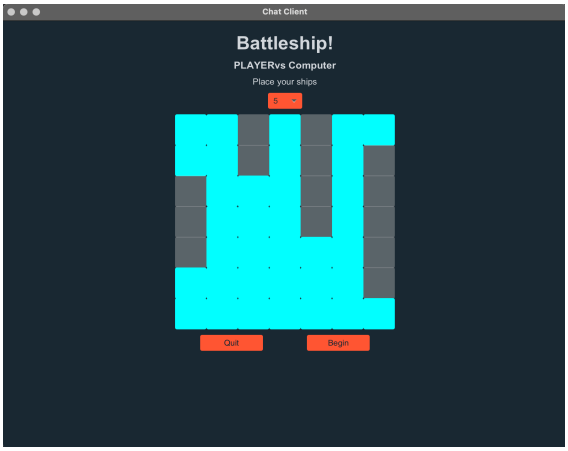




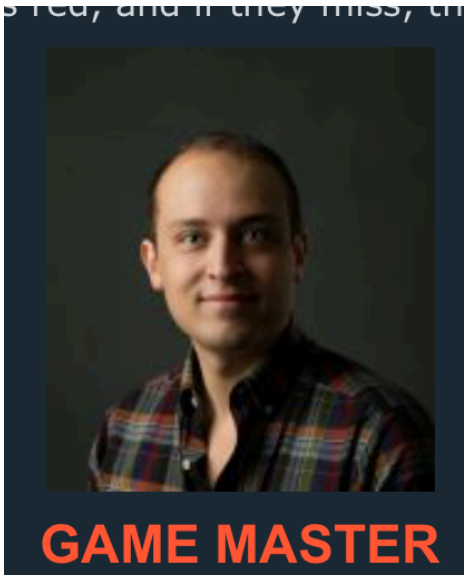


Aesthetically pleasing: We wanted to make the gameplay aesthetic and look simplistic. Cyan was chosen as the main color for the board to represent water and silver was used to represent modern army ships.





Reference to Prof. <3



Extensibility:

- Currently the size of the board is not dynamic and can be changed later to introduce different difficulty levels.
- Avatars can be introduced to represent each player. Maybe implementing a database would be a good idea to let users store their avatars and stats too!
- Betting could also be introduced (strongly condemned) but a fun way to engage players.
- Many power ups can be added as the players earn xp points and level up . Like “Area Splash” that reveals all the surrounding blocks of one designated block chosen by the player. “Free reveal " allows players to reveal one block that definitely contains a ship. And many more.

AI Contributions:

1) AI bot pseudo-code obtained from ChatGPT:

Example Code (Pseudocode)

plaintext — Copy code

```
function playBattleship():
    while not game_over:
        if mode == "hunt":
            cell = select_random_cell()
            result = make_move(cell)
            if result == "hit":
                mode = "target"
                target_stack.push(cell)
        elif mode == "target":
            if target_stack is not empty:
                next_cell = target_stack.pop()
            else:
                next_cell = find_next_target_cell()
            result = make_move(next_cell)
            if result == "hit":
                add_adjacent_cells_to_stack(next_cell)
            if check_ship_sunk():
                mode = "hunt"
                clear_target_stack()

function add_adjacent_cells_to_stack(cell):
    for each direction in [up, down, left, right]:
        adjacent = cell + direction
        if is_valid_cell(adjacent) and not_visited(adjacent):
            target_stack.push(adjacent)

function find_next_target_cell():
    # Use probability grid or simple adjacent checks
    return highest_probability_cell()

function check_ship_sunk():
    # Check if the hit ship is sunk
    return ship_sunk_status
```

2) Board Cell and Board Class shell code obtained from ChatGPT:

1. Define the Board Structure

The board is typically represented as a 2D array where each cell can hold information about whether it contains part of a ship, whether it has been hit, or whether a miss has occurred.

java — Copy code

```
public class Board {
    private final int size;
    private Cell[][] grid;

    public Board(int size) {
        this.size = size;
        grid = new Cell[size][size];
        initializeBoard();
    }

    private void initializeBoard() {
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                grid[i][j] = new Cell();
            }
        }
    }
}
```

2. Define the Cell Class

Each cell in the grid can be represented by a `Cell` class that contains the status of the cell, such as whether it's empty, has a ship, or has been hit.

java — Copy code

```
public class Cell {
    private boolean hasShip;
    private boolean isHit;

    public Cell() {
        this.hasShip = false;
        this.isHit = false;
    }

    public boolean hasShip() {
        return hasShip;
    }

    public void setShip(boolean hasShip) {
        this.hasShip = hasShip;
    }

    public boolean isHit() {
        return isHit;
    }

    public void setHit(boolean hit) {
        isHit = hit;
    }
}
```