The Four Musketeers

December 2024

1. Group Members

- Carl El Helayel
- Rayyan Kombargi
- Ahmad Chreif
- Housni Antar

2. GitHub URL

The source code and project details are available on our GitHub repository:
https://github.com/rayankombargi/TheFourMusketeers

3. High-level Description and Strategies Used

4. Issues

5. Resolutions

6. Limitations

7. Assumptions

High-level Description and Strategies Used:

High Level Description:

The battleship game is a turn by turn game that is played on a 10x10 grid and the aim of each player is to sink the ships of the opponent/bot by trying to guess the location they should target. In this assignment we also have acquired some special moves like radar sweeps which informs the user if there is a ship in a selected 2x2 area, smoke screen which hides the user's boats from the opposing radar sweeps, artillery which strikes a 2x2 area, and torpedo strikes which target a whole row or column in the grid.

Game Structure:

1. The grid starts out as an empty (~) 10x10 grid.
2. The grid tracks the hits (*) and misses (o) depending on the level selected by the user
3. Each player has 4 types of ships of different sizes (5,4,3,2). Ships also have orientation (vertical or horizontal).
4. In the game the players alternate turns after each move. A player loses their turn if an invalid move is entered.
5. Game ends when all the ships are sunk.

Strategies used:

1. Offensive Strategy:
    - Random Firing: Shots are fired at random locations to identify where ships are placed.
    - Targeted Firing: Once a ship is hit, shots are directed to adjacent cells (excluding diagonals) to maximize chances of sinking it.
    - Artillery: Targets a 2x2 grid, increasing the probability of hitting a ship.
    - Torpedo: Targets an entire row or column, making it particularly effective for locating larger ships.
2. Defensive Strategy:
    - Spread your ships apart to minimize the impact of artillery attacks.
    - Avoid aligning ships on the same row or column to reduce vulnerability to torpedo strikes.
    - Use smoke screens strategically to disrupt the attacker’s radar sweeps.
3. Bot Strategies:

- Adaptation and State Tracking: The bot maintains a state (BotState) that records the status of every cell (hit, miss, or unexplored). This helps it plan future moves intelligently by analyzing patterns and prioritizing unexplored or high-probability cells.
- Ship Placement: The bot places its ships strategically using a predefined algorithm to maximize the difficulty of locating them (no ships are placed adjacent to each other)
- Initial Targeting: The bot begins by randomly selecting grid cells to fire upon until it hits a ship.
- Follow-Up Attacks: When a ship is hit, the bot utilizes the pendingHits data in its BotState to target adjacent cells, ensuring it efficiently tracks and sinks the ship.
- Radar Utilization: The bot uses radar sweeps (limited to a maximum of 3) to reveal areas of the opponent's grid, improving its targeting decisions.
- Decision-Making Process: The bot switches between random targeting and guided targeting modes based on the situation, ensuring a balance between exploration and exploitation.

Issues:

Phase 1:

1. Location validation and boundary cases to be able to check if the boat is in the correct location
2. Faced issues with the random turns
3. Problem with skipping a players turn after using radar whenever the radar was unavailable
4. Multiple display issues where the grid was not aligned perfectly
5. Tried implementing the special moves but constantly got incorrect output
6. While implementing levels we had problems trying to improve the hits and miss tracking
7. Problem with smoke screen and how it works
8. Endgame condition failed to produce the correct output

Phase 2:

1. Faced the issue where the algorithm dedicated for the bot to prompt a move and coordinates was used for the Player’s turn and vice-versa.

2. Faced the issue where the coordinates of the placed ships which were placed by each player were off the grid.
3. Initially, we attempted to implement a structure to represent the bot's memory. However, the algorithms associated with this structure failed to ensure the bot could accurately target the coordinates of a detected ship. The memory was intended to store information about the bot's states. For instance, if the bot successfully hit a ship, it would transition to a state where it searches the surrounding area vertically and horizontally before returning to the random firing state.
4. We encountered an issue with the radar memory implementation, where we attempted to enable the bot to remember the locations it had scanned with the radar and record whether a ship was detected in those areas.
5. The bot's targeting logic struggled to perform better than random firing, limiting its strategic capabilities.
6. The large number of lines in phase 2 proved to be a setback as we struggled to navigate through the code

Resolution:

Phase 1:

1. Added a function to validate boat placement by checking for overlaps in the 2D array.
2. Implemented a random turn generator using the current time as a seed.
3. Created a function to skip a player's turn if their move is invalid, and mark the turn as done if valid.
4. Fixed the grid alignment by adjusting the for-loop index to ensure proper formatting.
5. Successfully implemented methods for each special move, ensuring correct output.
6. Incorporated a tracking system for hits and misses in the 2D array, adjusting based on the user's difficulty level.
7. Developed a method to properly implement the smoke screen feature, using a hasSmoke function in the 2D array.
8. Added a CheckWinCondition method to verify when the last ship is sunk and determine when the game should end.

Phase 2:

1. This was resolved by correcting the conditional logic to ensure the appropriate algorithm is executed for each turn.

2. Adjustments were made to the ship placement algorithm, ensuring the input coordinates align with valid grid locations.
3. The memory structure was replaced with a more effective *BotState* system, which utilizes the player's grid to verify the presence of ships at specified coordinates and their adjacent cells.
4. The radar memory concept was ultimately scrapped due to its inefficiency
5. Enhanced the bot's logic by integrating pattern recognition in its BotState, enabling it to prioritize high-probability targets based on past hits and misses. Additionally, state-based transitions were improved to optimize targeting during focused search or radar-assisted moves
6. In phase 2, one of the key improvements wasn't in the gameplay strategies but in how we structured and organized our code. We transitioned from a large, monolithic program to a more manageable and organized codebase by separating the code into relevant header (`.h`) and source (`.c`) files based on their functionality. While this change might seem minor in terms of gameplay output, it greatly enhanced the clarity, readability, and overall structure of our code. This organization made it much easier to navigate the code, locate specific components, and work more efficiently, ultimately improving our ability to debug and enhance the program.

Limitations:

1. We were unable to implement a method for the bot to strategically decide when to use the radar move. Currently, the bot is restricted to using radar moves only at the beginning of the match, before any ships have been located.
2. The torpedo and artillery moves are triggered exclusively when the bot has detected the location of a ship based on a pre-designated priority.
3. The bot does not utilize smoke moves. Implementing this functionality would require additional algorithms to determine when to deploy smoke, such as when some of its ships are under threat or have started to be eliminated and could prove to be inefficient in our BattleShip bot.

Assumptions:

Fixed Grid Size and Ship Placement:

- The game uses a fixed 10x10 grid. Ship placements are assumed to be valid and non-overlapping, however our program handles edge cases where input is invalid.

Player and Bot Turn Logic:

- The program alternates between the player's turn and the bot's turn. The logic for prompting moves and coordinates for each turn is handled by separate algorithms to ensure they are not swapped.
- The bot's move algorithm is designed to select random targets initially, transitioning to a more focused strategy after hitting a ship.

Bot Memory and State Management:

- The bot uses a state-based logic (represented by the BotState struct) to track hits, misses, and the status of nearby grid cells.
- The bot will switch between random firing and targeted attacks based on the results of previous moves, ensuring that it adapts to the game state.
- The bot cannot make decisions beyond the predefined strategies (e.g., the bot doesn't decide when to use radar more than once beyond the start of the game).
- The bot does not adapt or improve its strategy over time based on player actions. The bot's logic remains fixed throughout the game, and it does not use advanced machine learning techniques to improve its decision-making.

Special Abilities (Radar, Artillery, Torpedo):

- Special abilities like radar, artillery, and torpedoes are used based on the bot's state. The radar move is restricted to the start of the match when no ships have been found yet.
- Artillery and torpedoes are only used when the bot detects the location of a ship.
- The bot does not utilize smoke moves, as implementing such functionality would require additional logic to determine when to deploy smoke effectively, such as when its ships are under threat.