Battleship AI: Algorithm Analysis

Sierra Svetlik
Ethan Rule
Priyadharshini Damodharan
Sai Thanmai Polamreddy

April 28, 2024

Contents

1 Abstract

Developing a strategic AI for the classic Battleship game presents a unique challenge, especially when the game incorporates variable ship sizes and hole counts. Unlike the standard game with fixed ship configurations, this dynamic environment requires the AI to adapt its decision-making process in real-time. Traditional Battleship AI algorithms may struggle with the increased complexity of unknown ship layouts and variable ship lengths. This project aims to develop an artificial intelligence (AI) system capable of playing the classic game of Battleship against a human opponent.

2 Game Overview

Battleship is a two-player board game where each player has two grids: one for placing their own ships and another for tracking their shots on the opponent's grid. The objective is to strategically guess the coordinates of the opponent's hidden ships and sink them. The players then take turns guessing the coordinates of their opponent's ships. The traditional Battleship features a set of ships with predetermined sizes (Carrier: 5, Battleship: 4, Cruiser: 3, Submarine: 3, Destroyer: 2). The challenge is to efficiently sink the enemy's ships before they can sink yours.

2.1 Gameplay

1. Players take turns calling out coordinates (e.g., "B5") on the opponent's grid to guess ship locations.
2. The opponent responds with:
   - "Miss" if the guessed square is empty.
   - "Hit" if it contains part of a ship.
3. If a hit occurs, the player continues guessing adjacent squares to sink the ship.
4. The first player to sink all of the opponent's ships wins.

3 Motivation

The decision to focus on Battleship stems from its inherent strategic nature. Unlike a random selection of coordinates, the AI will employ advanced algorithms—specifically the Monte Carlo and Hunt-Target approaches—to enhance its gameplay. These algorithms will allow the AI to make informed decisions, and enable the AI to efficiently locate and sink the opponent's ships, even with variable ship sizes increasing its chances of success.

4 Algorithmic Approaches

4.1 Monte Carlo

Monte Carlo methods are widely used in Battleship AI due to their effectiveness in handling uncertainty and exploring possible outcomes. The algorithm samples random game states, estimating the likelihood of hitting a ship at different board positions. By simulating many games, Monte Carlo builds a probability distribution of ship locations, informing the AI's next moves.

4.2 Hunt-Target

The board for tracking attacks on the opponent, known as the "attack board", has a list of coordinates where successful hits were landed. It also keeps track of if squares are unknown, hits, destroyed (if the square is part of a ship that has been successfully sunk), or misses. The algorithm first checks to see if there are any hits that have been landed that are not on sunk ships, and then targets each of the squares around that hit. If there are no available hits to check, the algorithm then tries hunting for new hits.

5 Algorithm Implementation

5.1 Monte Carlo Simulation

1. Input the current board state.
2. Create a copy of the board state.
3. Simulate a specific number of samples (defined by –monte_carlo_samples):
    4. Each sample represents a random placement of a remaining ship type.
    4. Consider constraints (e.g., legal placements, no overlapping ships).
    4. Stack all simulations and tally the total number of ships in each square (with emphasis on ships overlapping existing hits).
4. Calculate the mean frequency for each square, resulting in a heatmap.
5. Select the highest value corresponding to a legal move in the heatmap.
6. Repeat steps 1-6 until a satisfactory move is found.

5.2 Hunt-Target

1. Hunt Implementation requires checking for coordinates on the board that haven't been targeted yet.
2. If the coordinate is attackable, then its probability is set to 1 on the 2D board array.

3. Probabilities are then sent to the step function where it checks for a legal board hit.
4. Once a coordinate is “hit”, the algorithm switches to targeting mode.
5. Target implementation targets the North, West, South, and East coordinates from the most recently hit coordinate.
6. Once a ship is sunk, the mode is set back to hunt.
7. This process is repeated until the game ends.

6 Algorithm Behaviors

The next steps involve delving into the behaviors of the implemented algorithms, focusing particularly on the strategies and tendencies during gameplay. Understanding these behaviors will provide insights into the AI’s decision-making process and approach to winning Battleship games.

6.1 Monte Carlo Behavior

After implementing ship placement, and analyzing the Monte Carlo algorithm it became apparent that the Monte Carlo algorithm explored from the center coordinates outwards. This tendency was due to how the ship placement was implemented, and how each Monte Carlo simulation was calculated.

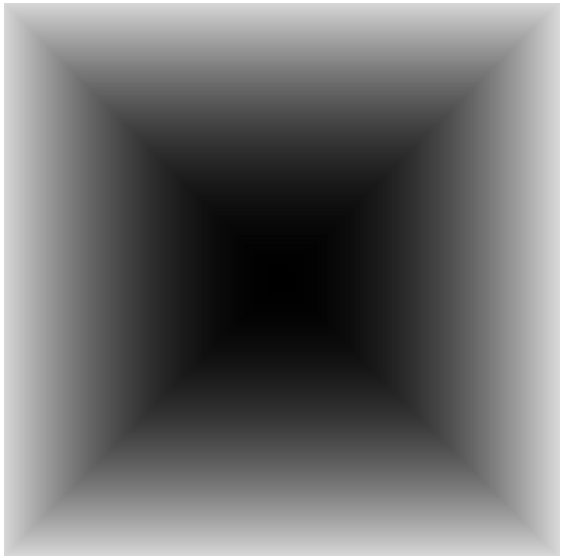

Figure 1: probabilities

Figure 1. Displays the probabilities of where ships are placed. Dark black means very high likelihood, while white displays lower. This is due to the implementation of randomizing ship placements. A ship is placed randomly, and if it is in an illegal position i.e. stacked on another ship or out of bounds, it is re-randomized and placed. This inherently means that any ship on the outskirts of the board will likely be replaced, and any near the center will likely not be replaced due to the bounds of the board.

With this ship placement in mind, the Monte Carlo tended to explore from the center outwards, and from a sunken ship outwards. In the event of ships being placed all on the outskirts of the board, the Monte Carlo algorithm would take a long while and would likely be beaten by the player.

A strength of the Monte Carlo algorithm was the sinking ships that were found. The algorithm checks and simulates coordinates next to the "hit" and selects the highest probability coordinate. This probability simulation enables the Monte Carlo algorithm to thrive when the first hit of a ship is detected.

6.2 Hunt and Target Behavior

The Hunt and Target AI algorithm consists of two modes. The first mode is the "Hunt" mode, within this mode, the algorithm searches systematically starting from the center of the board outwards in a diagonal pattern for ships. Here is an example of the end state of a game.

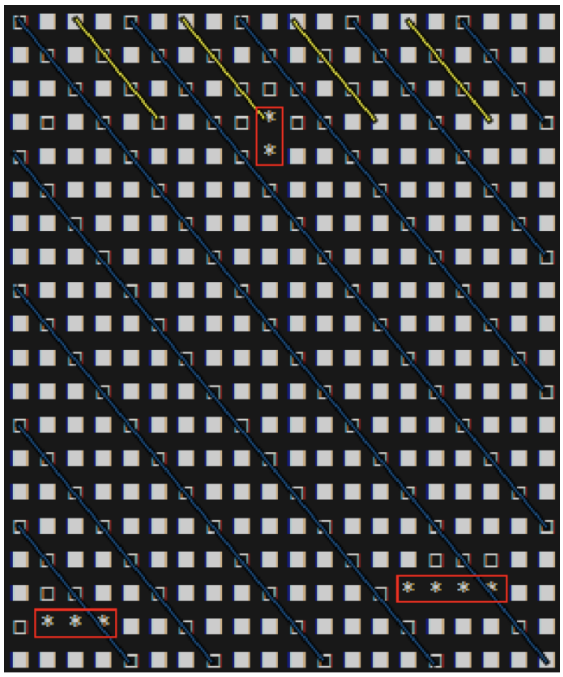

Figure 2: Hunt and Target Behavior

In Figure 2. the game was started with the boat sizes [2, 3, 4]. The hollow squares represent where a coordinate was selected. Notice how the hollow squares only have 3 non-selected squares between them both horizontally and vertically? This systematically searches the board so that the largest boat will be found. Then these diagonals are split in half once more which will find all boats larger than the size of 2.

The second mode is the Target mode. This mode is enabled once a "hit" has been registered from the previous turn. During this phase, the computer selects coordinates North, West, South, and East of the previously "hit" coordinate. Once the boat is fully sunk, it resumes the sequential Hunt search.

Additionally, efficiency in both algorithms is finishing off a ship once it's found. An AI can simply win since targeting a coordinate to finish off a ship can be twofold. It can both eliminate a ship and uncover another ship. Some players may think to find all the ships first in a global "hunt" phase, and then eliminate each ship afterward in a large targeting phase. Additionally knowing which ship has been sunk in the case of the Monte Carlo approach can reduce the probability of certain coordinates while searching and sinking the remaining ships. This gives the AI a tactical advantage.

7 Empirical Insights and Results

The next step involves further analysis of the empirical data obtained from gameplay statistics. Statistics like hits-to-misses ratio and player turn times are not included since they are less meaningful in drawing tangible conclusions.

7.1 Average Turn Times

After playing against the Monte Carlo AI for 5 games with a sample size of 5000, a board size of 8, and a ship size of [2, 3, 4], the average computer turn time was 0.23 seconds. Alternatively, running 5 games against the Hunt Target AI with the same board and ship size, the average computer turn time was 0.000421 seconds. The computer turn time was quicker than the Monte Carlo turn time, this was due to 5000 samples before each turn the Monte Carlo had to run. For reference 10000 samples had an average turn time of 0.44 seconds per turn.

7.2 Average Turns to Win

The Monte Carlo AI with the same previous board, ship sizes, and a sample size of 5000 played over 5 games takes on average 27.6 turns to win. The Hunt and Target AI takes an average of 30.8 turns to win. However, when tested on a board with a size of 20, the Monte Carlo takes an average of 191.4 turns, and the Hunt and the Target take an average of 140.4 turns. The larger the board, the more efficient the Hunt and Target becomes.

7.3 Player VS the AI

Playing as a human against the Hunt-Target algorithm, the human is unable to beat the computer without some luck. The Hunt-Target strategy is clearly visible in the following image (the computer's grid is on top), taken at the end of a round. The average human strategy is to aim for the most empty areas of the grid, while the computer moves methodically diagonally. The only disadvantage to the current implementation is that the computer doesn't know to aim in a straight line when it finds a series of hits. This can be seen on the upper grid in the image. Wherever there is a series of hits (represented with *), the computer has aimed around it as well.

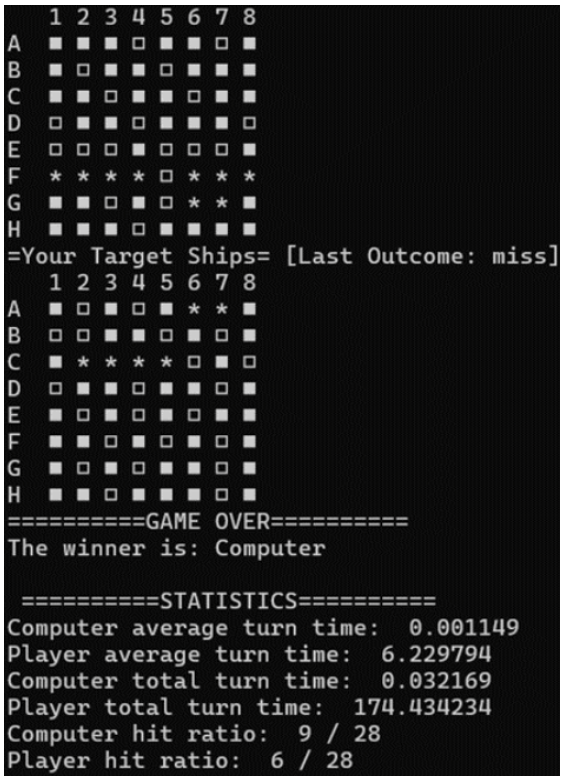

Figure 3: Statistics

Despite that disadvantage, the computer has a 50% higher accuracy rate than the human player. Depending on luck and strategy, the computer can have almost double the accuracy.

A human playing against the Monte Carlo algorithm has a better shot of winning, as the algorithm is less efficient at finding new ships than the Hunt-Target algorithm. As can be seen in the image below, the computer starts in the middle and expands outwards from there.

As a result, the human strategy of searching all over the board in the emptier areas has an advantage... as long as the human has luck on their side. With some bad luck, the human can still lose. The Monte Carlo algorithm might be less efficient at searching, but it can target better since it has learned to target ships when it knows the vicinity of the ships.

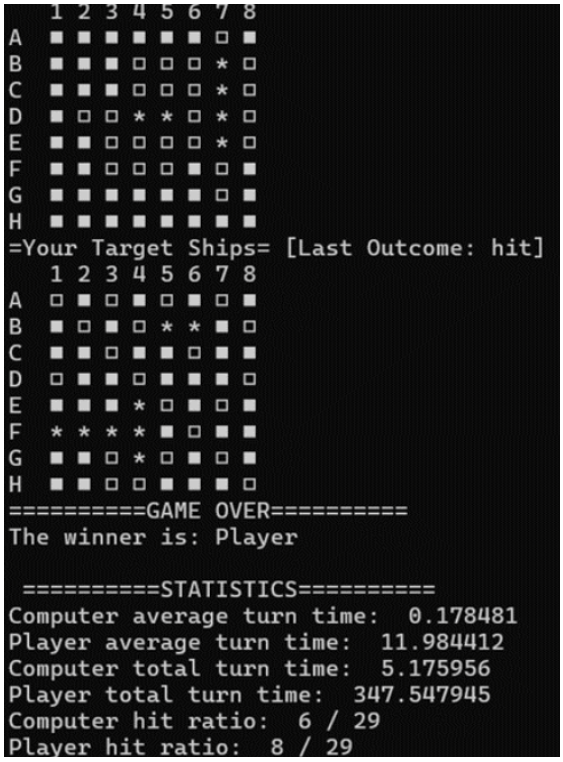

Figure 4: Statistics

8 Future Work

8.1 Algorithm Improvements

8.1.1 Monte Carlo

1. **Adaptive Sampling:** Implement adaptive sampling techniques in the Monte Carlo algorithm. This would allow the algorithm to dynamically allocate more simulations to areas with higher probabilities of containing ships, potentially improving its search efficiency.

2. **Parallelization:** If computational resources allow, explore parallelizing the Monte Carlo simulations to reduce the turn time for the AI.

8.1.2 Hunt-Target

1. **Linear Hit Detection:** As mentioned in your report, the Hunt-Target AI could be improved by incorporating the ability to identify and exploit linear ship placements based on hit sequences. This would make it even more effective against players who don't randomize ship placements strategically.

2. **Heuristic Refinement:** Explore refining the heuristic used by the Hunt-Target algorithm to prioritize search areas based on the remaining ship sizes. This could potentially improve its efficiency.

8.2 New Algorithmic Approaches

1. **Genetic Algorithms:** Investigate the use of genetic algorithms to evolve ship placement strategies and attack patterns for the AI. This could lead to the emergence of even more sophisticated and unpredictable AI behavior.

2. **Machine Learning:** Explore the potential of training a machine learning model on historical game data to learn effective ship placement and attack strategies. This could lead to an AI that adapts its tactics based on the human player's behavior.

8.3 Expanded Functionality

1. **Difficulty Levels:** Implement difficulty levels for the AI by adjusting parameters like the number of simulations in Monte Carlo or the search patterns in Hunt-Target. This would allow players to choose a challenge level that suits their skill.

2. **Visualizations:** Develop visualizations to show the AI's decision-making process. This could be helpful for understanding how the AI is evaluating the board and making its moves.

8.4 Multi-Player Options

1. **Cooperative AI:** Explore the possibility of developing a cooperative AI that can team up with a human player against another human or AI opponent.

2. **Networked Battleship:** Implement network functionality to allow players to compete against the AI or other human players over a network.

9 Conclusion

This project tackled the challenge of creating a strategic Battleship AI capable of outsmarting a human opponent, even with variable ship sizes and an unknown ship layout. We implemented and analyzed two algorithms: Monte Carlo and Hunt-Target. The Hunt-Target AI emerged as a highly accurate opponent, employing a methodical search pattern to efficiently locate and sink ships. The Monte Carlo AI, while less efficient in initial searches, excelled at targeting ships once a hit was confirmed. The project's findings highlight the impact of board size on each algorithm's performance. The Hunt-Target AI thrived on larger boards, while the Monte Carlo AI benefitted from smaller ones. Future improvements include incorporating linear hit detection and heuristic refinement for the Hunt-Target algorithm. The Monte Carlo method could benefit from adaptive sampling and parallelization techniques. Additionally, exploring new approaches using genetic algorithms and machine learning hold promise for even more strategic AI behavior.

10 References

1. J. B. Blake, "Monte Carlo Method in Battleship AI," *Building Intelligent Systems: A Guide to the Engineering of Knowledge-Based Systems*, Morgan Kaufmann Publishers, 2009.

2. S. J. Russell and P. Norvig, "Heuristic Search Algorithms for Battleship AI," *Artificial Intelligence: A Modern Approach* (3rd ed.), Pearson Education Limited, 2009.

3. M. Kaufman and M. Michalik, "An AI Approach to Battleship with Variable Ship Sizes," in *2014 14th International Conference on Advanced Robotics (ICAR)*, July 2014, pp. 1-6.

4. M. Buro, "Multi-Agent Systems for Battleship," in *Proceedings of the First IEEE International Conference on Intelligent Systems Design and Applications (ISDA 2003)*, 2003, pp. 843-848.