Shape, background pattern

Description automatically generated

AI-Assignment-03

Prepared by: Abad Naseer

Submission Date

31 October, 2024

# AI Battleship Game Using Minimax and Alpha-Beta Pruning

## 1. **Introduction**

This project is a two-player grid-based strategy game, similar to a simplified battleship-style game, where each player controls ships on a 10x10 grid. The objective is to destroy the opponent's flagship while protecting one’s own. The game involves strategic movement, attacking, and defense, with an AI opponent implementing Minimax with Alpha-Beta Pruning and Expectimax for optimal performance.

## 2. **Game Rules and Mechanics**

* **Game Board**: Represented as a 10x10 grid, with each player’s ships marked by unique identifiers.
* **Ships**: Each player has one flagship and three support ships. The flagship has 10 health points, and support ships have 3 health points.
* **Damage and Destruction**: Each attack reduces a target ship's health by 2 points. The flagship is destroyed after 5 hits, and support ships after 2 hits.
* **Victory Condition**: The game ends when one player’s flagship is destroyed.
* **Player Actions**: On each turn, a player may move a ship or attack a nearby grid cell. Ships can move or attack within a one-step range, including diagonal.

## 3. **AI Strategy**

The AI player is designed to play optimally by utilizing the **Minimax algorithm** with **Alpha-Beta Pruning** for efficiency and **Expectimax** to account for any uncertain outcomes (such as randomness in attack success). The AI strategy focuses on:

* **Balancing Offense and Defense**: Attacking the opponent’s flagship when in range while positioning its own ships defensively.
* **Evaluating Board States**: The AI scores moves based on minimizing the player’s advantage while maximizing its own, helping it prioritize moves that damage the opponent’s flagship or protect its own flagship.

### 3.1 **Minimax Algorithm**

Minimax is used to explore possible moves and counter-moves up to a defined depth. This allows the AI to simulate future actions and choose the move that maximizes its chances of winning or minimizes the opponent's advantage.

* **Evaluation Function**: The evaluation function assigns higher scores to moves that damage the opponent’s flagship or protect its own flagship. Defensive moves have high value when the AI’s flagship is under attack.



### 3.2 **Alpha-Beta Pruning**

To reduce computational complexity, Alpha-Beta Pruning eliminates moves that don’t improve the AI's score beyond already evaluated moves, enabling faster decision-making.

* **Efficiency**: By pruning unimportant moves, Alpha-Beta reduces the search space, allowing deeper evaluations without exceeding time constraints.

A screen shot of a computer

Description automatically generated

### 3.3 **Expectimax**

Expectimax can be added to simulate potential outcomes, especially when randomness or probability affects gameplay (e.g., variable attack success rates or unpredictable player moves). This allows the AI to consider a broader range of possibilities, making it adaptable to uncertain situations.

## 4. **Game Efficiency and Complexity**

* **Iterative Deepening**: The AI uses iterative deepening to handle time constraints, beginning with shallow depth and exploring deeper based on available time.
* **Move Evaluation Depth**: The AI aims for balanced depth, enough to capture complex scenarios but not so deep as to cause delays. Depth is typically limited to 3 levels in Minimax, balancing foresight with reasonable response times.

## 5. **Code Structure and Implementation**

### 5.1 **Modules and Functions**

1. **Grid Initialization**: The 10x10 grid is created with each player’s ships placed in predefined positions.
2. **User Input**: The get\_user\_move() function gathers player input, verifying ship movement and attack choices.
3. **AI Move Selection**: The AI evaluates moves with the Minimax function, using Alpha-Beta Pruning to select optimal moves.
4. **Move and Attack Logic**: Functions move\_ship() and attack() handle ship movement and attacks, applying damage and checking for ship destruction.
5. **Victory Condition**: The game\_over() function checks if a flagship is destroyed after each move, ending the game accordingly.

A screen shot of a computer

Description automatically generated

### 5.2 **Code Quality and Documentation**

The code is modular, with well-defined functions for each game mechanic. Clear documentation accompanies each function, detailing inputs, outputs, and purpose.

## 6. **Testing and Results**

* **Correctness**: The game logic adheres to specified rules, and the AI makes optimal moves under test scenarios, balancing offensive and defensive actions effectively.
* **Efficiency**: Alpha-Beta Pruning significantly reduces the number of states evaluated, allowing the AI to respond within a reasonable timeframe.
* **Complexity**: The AI consistently evaluates up to 3 levels of possible moves, maintaining performance while balancing complex decision-making scenarios.

A screenshot of a computer

Description automatically generated

## 7. **Conclusion**

This project successfully implements a battleship-style strategy game with an AI opponent that uses Minimax, Alpha-Beta Pruning, and Expectimax to make optimal moves. The AI manages to balance offensive and defensive strategies, adapting to game states to protect its flagship while targeting the opponent’s flagship. The modular code structure, along with efficient algorithms, allows for a smooth gameplay experience and effective AI decision-making.