Written Report

# . Write-Up

## How the Minimax Algorithm Works

The Minimax algorithm is a decision-making algorithm used in two-player, zero-sum games, where one player's gain is the other player's loss. It is based on the idea that each player strives to maximize their own outcome while minimizing the opponent's chances of winning.  
  
 Here’s how the Minimax algorithm works step-by-step:

1. **Tree Structure:**

The game is represented as a tree, where each node corresponds to a possible game state. The root node represents the current state, and child nodes represent possible future states after each player’s move.

2. **Maximizing and Minimizing**:

The algorithm alternates between the two players. One player is assumed to play optimally by maximizing their score, while the other player minimizes the score of the first player.  
 - The **Maximizing Player (AI)** aims to select the move that gives the highest possible score, assuming the opponent will play optimally.  
 - The **Minimizing Player (Opponent)** tries to choose the move that results in the lowest possible score for the Maximizing player.

3. **Evaluation**:

At the leaf nodes (terminal states), the algorithm assigns a score based on the outcome of the game (e.g., +1 for a win, -1 for a loss, and 0 for a draw). These scores are propagated back up the tree.

4. **Decision**:

The algorithm chooses the optimal move for the Maximizing player by selecting the child node with the highest score, while the Minimizing player selects the child node with the lowest score.

## The Benefits of Alpha-Beta Pruning

Alpha-Beta Pruning is an optimization technique used to improve the efficiency of the Minimax algorithm. It reduces the number of nodes that are evaluated in the search tree, resulting in faster performance without affecting the outcome of the game.  
  
 The key concepts of Alpha-Beta Pruning are:  
 1. **Alpha and Beta Values**:  
 - **Alpha** represents the best value found so far for the Maximizing player (the AI).  
 - **Beta** represents the best value found so far for the Minimizing player (the opponent).  
 2. **Pruning**:   
 - During the search, if the algorithm finds that a particular branch cannot influence the final decision (i.e., if the value of a node is worse than an already explored node), that branch is "pruned" (cut off from further evaluation).  
 - If the current node’s value is worse than the `alpha` value for the Maximizing player or worse than the `beta` value for the Minimizing player, further exploration of that node’s children is unnecessary and is skipped.  
 3. **Efficiency Gains**:

By cutting off these unnecessary branches early, Alpha-Beta Pruning reduces the number of nodes explored, making the algorithm faster than plain Minimax. In the best case, it can halve the number of nodes explored, improving the time complexity from **O(b^d)** (for Minimax) to **O(b^(d/2))**.

## The Performance Differences Observed During Testing

During testing, we compared the performance of the Minimax algorithm and the Alpha-Beta Pruning algorithm using several test cases. The performance was measured by the number of nodes explored and the time taken to compute the best move.  
  
 Test Setup:   
 - We ran both algorithms on the same game (Tic-Tac-Toe) and used identical starting conditions for both algorithms.  
 - Each test was conducted multiple times to average out variations in performance.  
  
 1. **Nodes Explored**:  
 - The **Minimax** algorithm explored all possible game states. As the game tree grows exponentially with each level (as each player has multiple possible moves), Minimax had to evaluate all potential moves before making a decision. This resulted in a higher number of nodes being explored.  
 - The **Alpha-Beta Pruning** algorithm, by contrast, pruned many of the non-promising branches. In some cases, Alpha-Beta Pruning reduced the number of nodes explored by half or more, particularly when a cutoff occurred early in the search. For example, in a game of Tic-Tac-Toe, where the depth of the tree is relatively small, Alpha-Beta Pruning explored fewer nodes than Minimax, making the search more efficient.  
  
 2. **Time Taken**:  
 - The **Minimax algorithm** took significantly longer to compute the best move due to its exhaustive search. As the game tree increases in size, the algorithm’s runtime increases exponentially.  
 - The **Alpha-Beta Pruning** algorithm, thanks to pruning unnecessary branches, reduced the runtime considerably. In some of the tests, the time taken by Alpha-Beta Pruning was noticeably shorter compared to Minimax, especially when the search tree depth increased, and more pruning was possible.  
  
 3. **Observed Performance Differences**:  
 - **Without Pruning**: In games with deeper search trees, the difference in performance between Minimax and Alpha-Beta Pruning was significant. Minimax would explore a large portion of the tree, while Alpha-Beta Pruning would skip over many branches, leading to a faster computation of the best move.  
 - **With Pruning**:

In smaller search spaces (like Tic-Tac-Toe), the performance difference between the two algorithms was less dramatic. However, even in these cases, Alpha-Beta Pruning still offered some advantage in terms of fewer nodes explored and faster decision-making.

**Effect of Heuristic Evaluation**

With the **heuristic evaluation function**:

1. The **Minimax and Alpha-Beta Pruning** algorithms can now evaluate intermediate game states more effectively.
2. The algorithm will Favor positions where the AI has the potential to win soon and avoid situations where the opponent might win.
3. This heuristic evaluation improves the algorithm's decision-making by assigning meaningful values to non-terminal states, allowing the AI to make more strategic moves.

**Conclusion:**

Alpha-Beta Pruning significantly outperforms the Minimax algorithm, especially in games with larger decision trees. By reducing the number of nodes explored, Alpha-Beta Pruning not only improves time efficiency but also makes it feasible to explore deeper game trees within a reasonable time frame. This makes it a valuable optimization technique, particularly in more complex games.