Backgammon AI

Peter Manolis, Cody Leithem, and Mark Dzwonczyk

December 19, 2024

1 Introduction

Backgammon is a two-player board game combining elements of strategy and luck. Each player has 15 checkers that move around 24 points on a board, according to the roll of the two dice. The objective is to move all of one's checkers into their home board and bear them off before the opponent does. A player can hit an opponent's exposed checker, sending it to the bar, where it must re-enter play before the opponent can continue advancing. Winning is achieved by moving all of your checkers into your home board and bearing them off the board before your opponent does. Developing an AI to play backgammon presents several challenges. One of the main difficulties is handling the game's randomness, introduced by the dice rolls. Unlike chess, where the state is fully deterministic, backgammon requires the AI to make probabilistic decisions and account for a wide range of potential future states. The AI needs to balance between short-term tactical moves, such as hitting or blocking an opponent, and long-term strategic goals, such as safely advancing checkers and bearing off. Backgammon has many variations. For example, Hypergammon is a fast-paced version where players start with only three checkers, demanding quick and precise moves. Chouette allows multiple players, with one playing solo against a team, making it a more social and strategic game. Nackgammon introduces a different starting position for checkers, creating a more complex and strategic game from the outset. Acey-Deucey, popular in military settings, offers the chance for extra moves when rolling a one and two, adding unpredictability. Russian Backgammon has players start all checkers off the board, making the game more of a race to get checkers onto the board and across it. For our project, we are focusing on the most common and basic version of the game. For our project, we are using alpha-beta pruning, an optimization technique for the minimax algorithm. The minimax algorithm is a decision-making algorithm used in two-player, zero-sum games like Backgammon. It evaluates the potential outcomes of a game by simulating all possible moves, with the goal of maximizing the player's minimum guaranteed outcome, while minimizing the opponent's maximum guaranteed outcome. Alpha-beta pruning optimizes this process by "pruning" branches of the game tree that don't need to be explored, reducing the number of nodes evaluated. Alpha-beta pruning works by maintaining two values, alpha and beta. Alpha represents the best score the maximizing player can guarantee so far, and beta represents the best score the minimizing player can guarantee. As the algorithm explores the tree, it compares the current node's value against alpha and beta to determine if further exploration is necessary. If a node's value is worse than the current alpha or beta, that branch is pruned, meaning it is skipped, and the algorithm moves on to other branches. The alpha-beta pruning algorithm has a time complexity of O(bd/2), where "b" is the branching factor (the average number of moves per position) and "d" is the depth of the game tree. This is an improvement over the standard minimax algorithm, which has a time complexity of O(bd). The space complexity is O(b*d), similar to minimax, as the algorithm still needs to store the game tree nodes. The primary limitation of alpha-beta pruning is that it assumes a fully deterministic game and doesn't account for the randomness introduced by dice rolls in backgammon. This reduces its effectiveness in games with an element of chance, as the algorithm can't fully account for the probabilistic outcomes of dice rolls. An alternative to alpha-beta pruning is Monte Carlo Tree Search, which is better suited for games with randomness, but is more computationally expensive.

2 AI Thought Process

When thinking about how we can make our AI model human like behaviors, we used alpha beta pruning to get this done. Alpha beta pruning is an enhancement to the minimax algorithm. It uses two key ideas, it focuses on the most promising moves available to the player and disregarding clearly inferior decision makings. This is the same idea that skilled players make when playing decision based games. They try their best to remove suboptimal plays even if it looks good in the current situation. This relates directly to backgammon. The player must look at potential consequences several turns into the future. Some heuristics that a human might use in their decision making is making prime formations, avoiding blots, hitting and re-hitting, anchoring, building the home board, escaping the back checkers, total dice rolls needed, opponents vulnerabilities, and risk vs reward. Our AI works in a similar way. It looks at heuristic values based on the board state. It takes these values and looks for positional advantages, blocking opponents, and advancing to bearing off checkers. Alpha Beta pruning models mental shortcuts that humans would make when playing the game. If one decision is more likely to lead to a bad outcome, it is quickly disregarded. Our AI will "prune" a path when the move is considered suboptimal, just like a human would disregard a strategy that is not worth pursuing. Adaptive decision making is another key factor in the decision making process. Humans change their strategy as the game goes on. Your decisions may change if it's the start, middle, or end of the game. They may also change depending on if you're losing, winning, or about even. For instance, if a player is ahead, they may take a more conservative approach. If they are behind, they may take riskier moves, trying to come back. The Ai emulates this by evaluating the board and game state and using the factors it has in front of them to make better decisions that would hopefully lead to a better outcome. A final key feature to our implementation is trade offs. In a game of Backgammon, you may have an opportunity to make a strong offensive move, or reinforce your defense. Each of these decisions seem valid for the human player. A human may tend to play one way or the other based on what seems better in the moment. Our AI assigns values to both decisions and ultimately chooses what will lead to the best outcome in the future.

3 Methods and Techniques

Our AI employs a methodical approach that allows for usability when someone new approaches the program. We use many different decision-making algorithms and board evaluation heuristics. The core algorithms are MiniMax and Alpha-Beta Pruning. MiniMax recursively looks through possible game states to find the most optimal move. It also takes in a depth parameter, where the user can pick a depth that they desire. Each move is calculated with a heuristic function that looks at:

- **Borne-off pieces**: The number of pieces successfully removed from the board.
- **Pieces on the bar**: The number of pieces currently trapped on the bar.
- **Blocking**: The effectiveness of blocking the opponent's movement.
- **Prime points**: Consecutive occupied points that restrict the opponent's progress.
- **Controlling the home board**: The strength of the player's position in their home board.

Alpha-Beta uses the same logic as MiniMax but enhances computational efficiency. Branches of the "tree" are disregarded as the maximizing or minimizing player can't influence that branch. This greatly reduces the number of board states that our AI needs to analyze. These algorithms are stored in the $PlayerMM$ and $PlayerAB$ classes. Our board is made up of 24-point arrays, one for each player. We also have separate arrays for the bar and borne-off pieces that represent the special parts of the board. Now, onto the actual gameplay. The turn attribute is used to track if it is player 1's or player 2's turn, alternating between 0 and 1. The dice roll happens next. These are two random integers between 1–6 that are then stored in a variable. This variable is then used in the `get_valid_moves` method that generates all the valid moves, given the board state and dice roll. Finally, `apply_move` and `undo_move` are used to simulate future states and revert them when necessary. Our AI will get the best possible move and then apply it to the board. This cycle continues until one of the players wins the game.

4 Results

When looking at the empirical analysis of our AI, we decided the best way to analyze the success of the AI was to get its win rate. Backgammon is just a game at the end of the day, and winning is the best tell. The first test was Alpha-Beta vs. humans. We represented the humans. We then ran the Alpha-Beta version against MiniMax and against itself.

We found that our implementation was able to win against us around 90% of the time. The couple of games in which we won were strictly lucky. It was also able to beat MiniMax by around 75%. And finally, when playing itself, the player who went first had just over a 50% win rate at 50.8%.

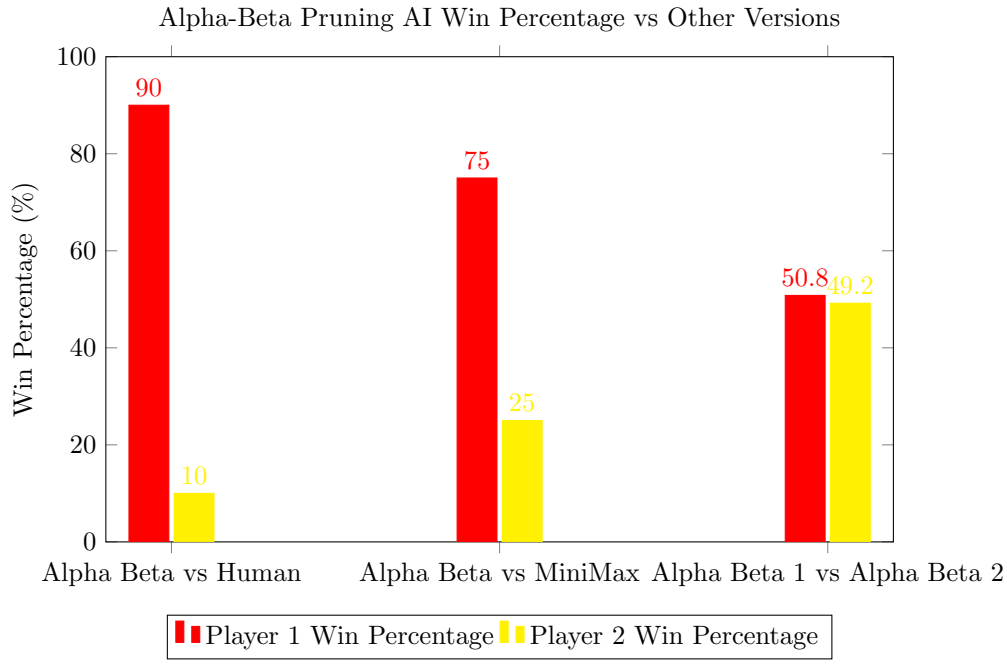

Figure 1: Alpha-Beta Pruning AI Win Percentage vs Other Versions

4.1 Depth

Next, we wanted to look at the effects of alpha beta pruning depth, and how it changed win rate and number of nodes evaluated. When changing the depth from 1-5, the results are shown below. This test was also run against humans. Playing the Ai at a depth of 5 was nearly impossible to win, and you needed to get doubles on many dice rolls to be victorious.

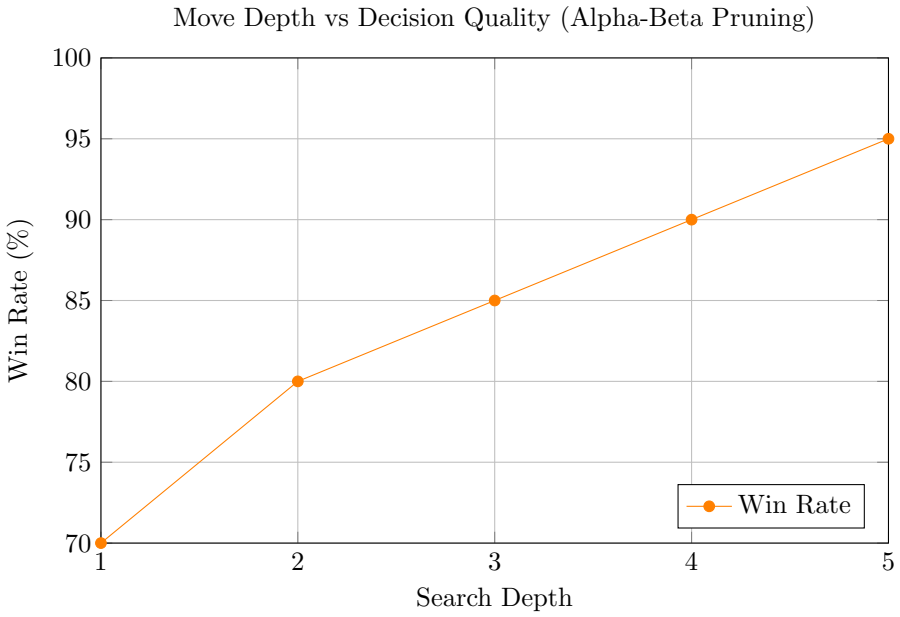

Figure 2: Move Depth vs Decision Quality (Alpha-Beta Pruning)

Next, is the number of nodes evaluated. As you can see, both algorithms linearly increase the number of nodes evaluated as the search depth increases. Alpha-beta looks at around half as many nodes as minimax at all depths.

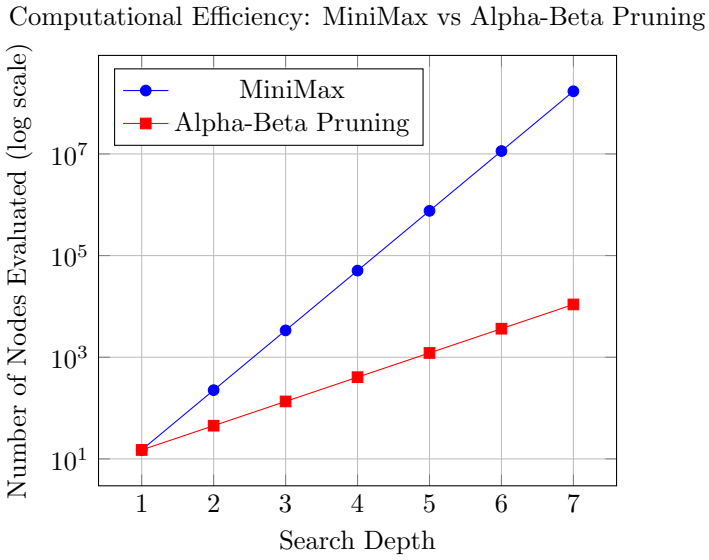

Figure 3: Comparison of Nodes Evaluated by MiniMax and Alpha-Beta Pruning

5 Conclusion

In conclusion, we built a backgammon AI using the Minimax algorithm, which is capable of evaluating and selecting optimal moves by simulating all possible game states and choosing the one that maximizes its chances of winning while minimizing its opponent's potential. While the Minimax algorithm provides a strong foundation, it is limited in handling the large branching factor of backgammon's state space, further complicated by the randomness of dice rolls. We also implemented alpha-beta pruning to optimize search efficiency. Through empirical analysis, we observed the AI's competitive capabilities against humans and against each other. Notably, we found that the alpha-beta pruning algorithm outperformed humans, winning 90% of the games played. Overall, this project provided valuable insights into the practical application of Minimax and alpha-beta pruning in the context of game AI development. By implementing these algorithms, we gained a deeper understanding of how to simulate game states and optimize decision-making in an adversarial environment. We also learned how important techniques like alpha-beta pruning are for enhancing the efficiency of search algorithms, particularly when dealing with large and complex state spaces like backgammon. Looking forward, our work could be expanded by incorporating machine learning to enhance the AI's decision-making. An approach we could implement would be to use supervised learning to train a neural network on historical game data. This allows the AI to predict the value of board positions and adapt its strategy over time. A key challenge we faced was balancing defensive and aggressive strategies, deciding when to protect checkers and when to take risks. This tradeoff is central to backgammon and could be further optimized through machine learning, offering significant improvements in the AI's performance and adaptability in the future.