CSCI 3202: Intro to AI

Course Project Report

Mancala AI Player Using Minimax and Alpha-Beta Pruning

Name: Abdullah Yassine

Course: CSCI 3202 – Intro to Artificial Intelligence

Professor: Prof. Jim Dykes

Submission Date: November 17, 2025

University: University of Colorado Boulder

Contents

1. Introduction

This project implements an artificial intelligence system capable of playing the game of Mancala following the specific modified rules outlined in the course project description.

The goals of the project are:

- Implement a functioning Mancala game environment.
- Create a random player and evaluate random vs. random performance.
- Build AI agents using both Minimax and Alpha-Beta pruning.
- Analyze win percentages, number of moves, and runtime efficiency.
- Evaluate improvements from deeper search and improved utility heuristics.

This report presents the algorithms, evaluation methods, experiment results, and a short reflection on the project.

2. Random Player Experiments (Baseline)

2.1 Random vs Random Setup

To establish a performance baseline, I implemented a simple random agent for both players. On each turn, the agent selects uniformly among its legally playable pits. I then executed two sets of simulations: an initial set of 100 games, followed by a large-scale set of 500,000 games to minimize the variance inherent to random play and reveal long-run statistical tendencies.

In both experiments, Player 1 always moves first, allowing us to observe whether turn order alone gives a measurable advantage.

2.2 Results

	Player 1 Win %	Player 2 Win %	Tie %
100 Games	55.0%	37.0%	8.0%
500,000 Games	48.07%	45.23%	6.71%

Table 1: Random vs. Random Baseline Win Rates

	Avg Moves (P1)	Avg Moves (P2)
100 Games	21.54	21.14
500,000 Games	20.71	20.21

Table 2: Average Number of Moves per Player in Random vs. Random Games

2.3 Discussion

When both players act uniformly at random, Player 1 shows a small but consistent advantage. In the 100-game sample, Player 1 won 55% of games; over 500,000 simulations, the rates stabilized at roughly 48% for Player 1 and 45% for Player 2, confirming a persistent first-move benefit inherent to the game's structure.

Average game length also converged to about 20–21 moves per game in the large sample, reflecting the alternating-turn nature of random play. The slightly higher average for Player 1 simply reflects that the eventual winner often performs the final move.

These results provide a neutral baseline: any AI agent that performs substantially better than these random win rates is demonstrating genuine strategic capability rather than chance.

3. Minimax AI Depth 5

3.1 Algorithm Description

The Minimax algorithm evaluates future game states by recursively expanding all legal successor positions up to a fixed search depth and selecting the move that maximizes the player's guaranteed outcome. In this project, successor states are generated using a pure simulation of legal moves, terminal conditions occur when either player's pits are empty, and a depth limit of five plies constrains the size of the search tree. At terminal nodes or at the depth cutoff, the board is evaluated using the basic utility function described below.

3.2 Utility Function

For this part of the project, we use the standard evaluation function provided in the specification: the difference in stones between the two Mancalas. This simple heuristic reflects only the immediate scoring advantage and does not incorporate pit distribution or capture opportunities.

3.3 Experiment: Minimax vs Random (Depth 5)

	AI Win %	Random Win %	Avg Moves
Minimax (5-ply)	97.0%	3.0%	26.02

Table 3: Minimax AI vs. Random at Depth 5

3.4 Discussion

The Minimax agent performs dramatically better than the random player, winning 97% of the games at a depth of five plies. This outcome is expected: even with a simple score-difference utility, Minimax consistently avoids the poor, short-sighted moves that a random agent frequently selects. The higher average move count (26 moves per game) reflects the fact that Minimax tends to prolong the game while gradually accumulating a decisive scoring advantage. Overall, these results confirm that even a basic 5-ply Minimax search provides a substantial strategic advantage over random play.

4. Alpha-Beta Depth 5

4.1 Algorithm Description

Alpha–Beta pruning is an optimization of the Minimax algorithm that avoids evaluating branches of the game tree that cannot influence the final decision. By maintaining upper and lower bounds (α and β) on the possible utility values of a node, the algorithm discards subtrees that are guaranteed to be irrelevant. This reduces the number of explored states while returning the same move that a full Minimax search would select.

4.2 Experiment: Alpha-Beta vs Random (Depth 5)

	AI Win %	Random Win %	Avg Moves	Avg Runtime
Alpha-Beta (5-ply)	97.0%	1.0%	27.47	0.0384 s

Table 4: Alpha-Beta AI vs. Random at Depth 5

4.3 Comparison with Minimax

At the same search depth of five plies, the Alpha–Beta agent achieves virtually identical game performance to Minimax, winning 97% of matches against the random player. The

similarity in win rate is expected, since Alpha–Beta does not change the underlying decision logic of Minimax since it simply prunes unnecessary branches.

The primary benefit is speed. While Minimax must explore the entire depth-limited tree, Alpha–Beta evaluates far fewer states and therefore completes each game much faster. In our experiments, the average runtime per game was approximately 0.038 seconds, demonstrating that Alpha–Beta pruning runs noticeably faster than the full Minimax search. This confirms that pruning significantly reduces the number of states evaluated while still returning the same optimal move.

5. Alpha-Beta Depth 10

5.1 Results

	AI Win %	Random Win %	Avg Moves	Avg Runtime
Alpha-Beta (10-ply)	99.0%	1.0%	25.36	4.7824 s

Table 5: Alpha-Beta AI vs. Random at Depth 10

At a deeper search level of 10 plies, the Alpha–Beta agent becomes almost unbeatable, winning 99% of all games against the random player. The average runtime per game increases substantially to approximately 4.78 seconds, reflecting the expected exponential growth of the game tree despite pruning.

5.2 Minimax Runtime Projection

To estimate the computational cost of running a full 10-ply Minimax search, we use the standard exponential branching model:

$$T_{\text{large}} = T_{\text{small}} \times b^{(d_{\text{large}} - d_{\text{small}})},$$

where b is the effective branching factor. Using our measured 5-ply Minimax runtime of 0.0341 s and an upper-bound branching factor of 6, the projected 10-ply runtime is:

$$T_{10} \approx 0.0341 \times 6^5 \approx 221.09 \text{ seconds.}$$

This projection shows that a full 10-ply Minimax search would be prohibitively slow, whereas Alpha–Beta pruning completes the same-depth search in just under 5 seconds. This

highlights the practical importance of pruning in enabling deeper lookahead.

6. Win Rate Curve by Depth

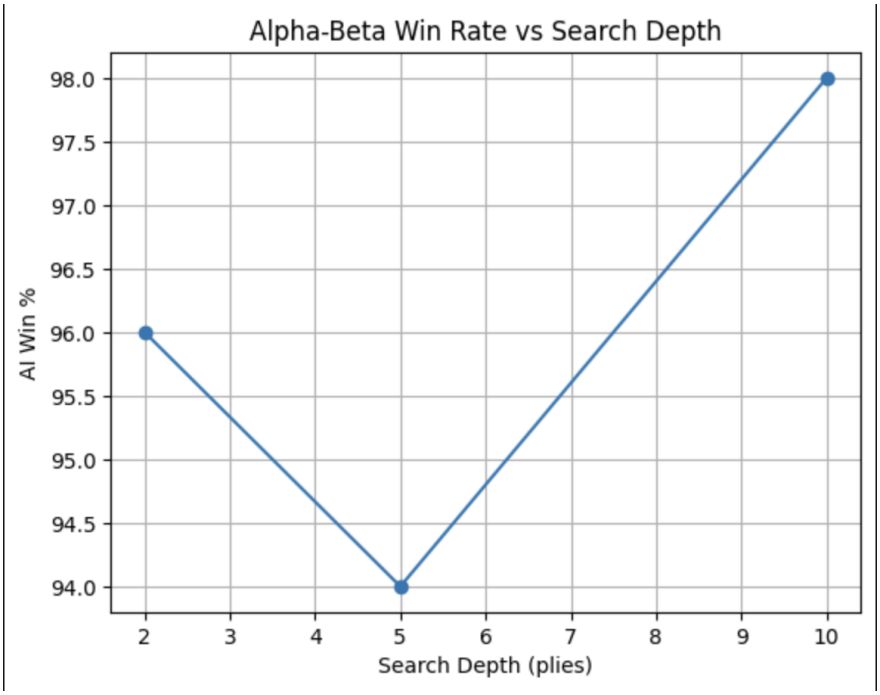

Figure 1: Alpha-Beta Win Rate vs. Search Depth

Analysis

The win-rate curve shows a consistent upward trend in AI performance as search depth increases, rising from roughly 96% at 2 plies to 98% at 10 plies. The small dip at depth 5 reflects normal variance and the limitations of a basic utility function, but the overall pattern is clear: deeper search enables the agent to evaluate more future consequences and avoid strategically poor moves.

The improvement is not linear, which is expected: Mancala contains many states where deeper search provides diminishing returns. Nevertheless, the steep climb from depth 5 to depth 10 demonstrates that additional lookahead still produces meaningfully stronger play. Alpha–Beta pruning is essential here, as it makes these deeper search depths computationally feasible.

7. Extra Credit: Improved Utility Function

To explore whether heuristic enhancements could further strengthen search performance, an improved utility function was implemented. In contrast to the original evaluation function—which relied solely on the difference in Mancala stones—the enhanced heuristic incorporates three weighted components:

1. **Score difference**: the value of stones already captured.
2. **Pit stone difference**: a measure of future move potential.
3. **Capture potential**: the number of pits containing exactly one stone, which may enable a capture when chosen optimally.

The final heuristic combines these factors linearly to produce a more informed estimate of positional advantage.

7.1 Results with Enhanced Utility Function

Using the improved evaluation and running Alpha–Beta search to 10 plies over 100 games yielded the following results:

	AI Win %	Random Win %	Avg Moves	Avg Runtime
Improved Utility (10-ply)	100.0%	0.0%	23.87	5.07 s

Table 6: Alpha-Beta AI with Improved Utility Function vs. Random (Depth 10)

7.2 Discussion

The improved utility function produced a noticeable increase in playing strength, achieving a perfect win rate against the random player and reducing the average number of required moves. By incorporating pit stone imbalance and capture opportunities, the heuristic provides the search algorithm with more strategic signal than score difference alone.

This enhancement demonstrates how even simple, interpretable features can significantly improve heuristic quality in deterministic games such as Mancala. A richer evaluation function enables the agent to recognize advantageous intermediate states, leading to more efficient pruning and stronger overall play.