Student Names and IDs:

- Omar Hany Ramadan Badr (ID: 2405728)
- Aly Ameen Abd-ElMawla (ID: 2405765)

Repository Link: https://github.com/omarbadrJr/ai_assignment_two

Connect Four AI Algorithm Analysis Report

1. Introduction

This report provides a comprehensive analysis of the two Connect Four AI algorithms implemented in the provided codebase: **Minimax (without pruning)** and **Minimax with Alpha-Beta Pruning**. The analysis includes a detailed look at the underlying data structures and algorithms, a comparison of their performance across various search depths (K), and a sample run with corresponding minimax tree visualizations.

2. Algorithms and Data Structures

2.1. Data Structures

The core data structure for the game state is managed by the `Board` class (`board.py`).

Data Structure	Purpose	Implementation Detail
Game Board	Stores the state of the 6x7 Connect Four grid.	A 2D Python list (`self.grid`) where each element is an integer: `0` (EMPTY), `1` (HUMAN), or `2` (AI).
Transposition Table (TT)	Caches the results of previously computed game states to avoid redundant calculations.	A Python dictionary (`self.transposition`) within the `Minimax` class. The key is a tuple representation of the board state (`board.as_tuple()`), the remaining search depth, and whether the current node is a maximizing node.

2.2. Algorithms

The AI decision-making is handled by the `Minimax` class (`minimax.py`), which implements two distinct search algorithms:

1. **Minimax (Without Pruning):** A standard recursive implementation of the Minimax algorithm that explores every node in the search tree up to the specified depth K.
2. **Minimax with Alpha-Beta Pruning:** An optimized version of Minimax that uses α (alpha) and β (beta) cutoffs to eliminate branches of the search tree that cannot possibly influence the final decision.

Both implementations share several key features:

- **Iterative Deepening (ID):** The `find_best_move` methods use Iterative Deepening, starting the search from depth 1 and increasing the depth until the maximum depth K is reached or a time limit is exceeded.
- **Move Ordering:** Moves are ordered based on a quick, shallow evaluation before the main search begins. This is a crucial optimization for Alpha-Beta Pruning, as evaluating better moves first increases the likelihood of early cutoffs.
- **Transposition Table (TT):** Both algorithms utilize a Transposition Table to store and retrieve previously calculated scores for identical board states, further reducing redundant computation.

2.3. Heuristic Function

The evaluation function, or heuristic, is defined in `heuristic.py` . It calculates a score for a given board state from the AI's perspective (maximizing player). The heuristic is a weighted sum of various features:

Feature	Weight	Description
Four-in-a-row	$W_4 = 10000$	A winning position for the AI.
Open Three	$W_3 = 100$	Three AI pieces in a line with both ends open.
Open Two	$W_2 = 10$	Two AI pieces in a line with both ends open.
Opponent Open Three	$W_{opp3} = -120$	A critical threat (three opponent pieces in a line with one open end) is penalized more heavily than a simple AI Open Three is rewarded, encouraging defensive play.

Center Control	$W_{center} = 3$	Rewards pieces placed in the center column, which is strategically valuable in Connect Four.

3. Assumptions and Clarifications

The following assumptions and details were noted from the codebase and experimental setup:

- **Definition of K:** K represents the maximum search depth for the Minimax algorithm. Due to the use of Iterative Deepening, the algorithm searches all depths from 1 up to K.
- **Time Limit:** The comparison script was run without a time limit to ensure a complete search up to depth K for accurate node count comparison.
- **Test Position:** All comparison data and sample runs were generated from a fixed, mid-game test position (as defined in `main.py`'s `compare_algorithms_with_trees` function):

  Plain Text

  ```
  . . . . . . .
  . . . . . . .
  . . . . . . .
  . . . . . . .
  . . . O . . .
  . . . X X . .
  0 1 2 3 4 5 6
  ```

  (AI is 'O', Human is 'X'. AI is the maximizing player and is to move next.)

4. Sample Runs and Minimax Trees

A sample run was executed on the test position with a search depth of $K=3$.

Algorithm	Best Move (Column)	Evaluation Value	Time Taken (s)	Nodes Expanded
Minimax (No Pruning)	2	13	0.093	856
Alpha-Beta Pruning	2	13	0.072	521

Both algorithms correctly identified the same best move (column 2) with the same evaluation value (13), confirming the correctness of the Alpha-Beta implementation.

4.1. Minimax Tree Snippet (No Pruning)

The Minimax tree explores all branches, as indicated by the absence of α and β bounds and no `PRUNED` nodes.

Plain Text

```
ENTER | col=None | val=MAX (a=N/A, b=N/A)
├── ENTER | col=4 | val=MIN (a=N/A, b=N/A)
| ├── ENTER | col=3 | val=MAX (a=N/A, b=N/A)
| | ├── LEAF | col=4 | val=63 (a=N/A, b=N/A)
| | ├── child | col=4 | val=63 (a=N/A, b=N/A)
| | ├── LEAF | col=2 | val=43 (a=N/A, b=N/A)
| | ├── child | col=2 | val=43 (a=N/A, b=N/A)
... (continues to explore all 7 branches)
| ├── EXIT | col=3 | val=63 (a=N/A, b=N/A)
| ├── child | col=3 | val=63 (a=N/A, b=N/A)
| ├── ENTER | col=4 | val=MAX (a=N/A, b=N/A)
... (continues to explore all 7 branches)
| ├── EXIT | col=4 | val=66 (a=N/A, b=N/A)
...
```

4.2. Minimax Tree Snippet (Alpha-Beta Pruning)

The Alpha-Beta tree shows cutoffs, significantly reducing the number of nodes explored. In the snippet below, a pruning event occurs at line 23, where the α value (66) exceeds the β value (63), indicating that the current branch is worse than a previously found alternative and can be safely ignored.

Plain Text

```
ENTER | col=None | val=MAX (a=-inf, b=inf)
├── ENTER | col=4 | val=MIN (a=-inf, b=inf)
| ├── ENTER | col=3 | val=MAX (a=-inf, b=inf)
... (explores 7 branches)
| ├── EXIT | col=3 | val=63 (a=63, b=inf)
| ├── child | col=3 | val=63 (a=-inf, b=inf)
| ├── ENTER | col=4 | val=MAX (a=-inf, b=63)
| | ├── LEAF | col=3 | val=66 (a=-inf, b=63)
| | ├── child | col=3 | val=66 (a=-inf, b=63)
| | ├── PRUNED | a=66 | b=63  <-- Alpha-Beta Cutoff
```

```
| ├── EXIT | col=4 | val=66 (a=66, b=63)
...
```

5. Performance Comparison

The two algorithms were compared across search depths $K=1$ to $K=6$. The results clearly demonstrate the performance benefits of Alpha-Beta Pruning.

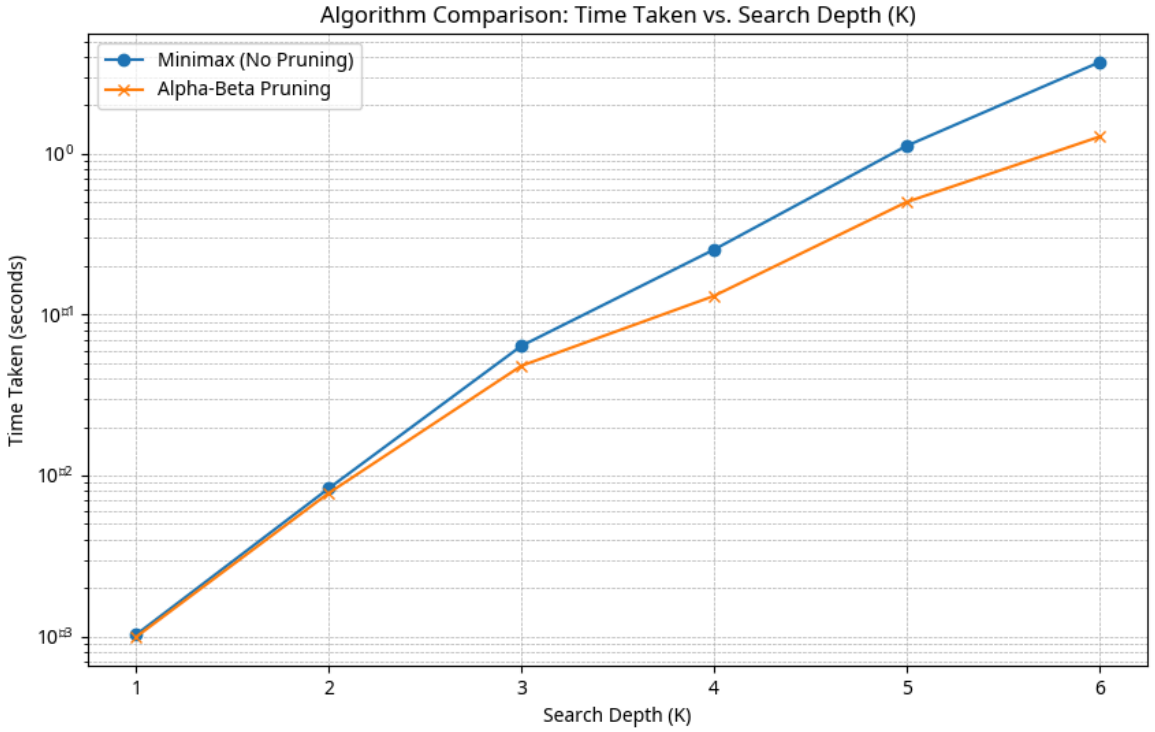
5.1. Raw Data Comparison

K (Depth)	Algorithm	Time Taken (s)	Nodes Expanded	Node Reduction (%)	Speedup (x)
1	Minimax	0.0010	16	0.00	1.03
1	Alpha-Beta	0.0010	16	-	-
2	Minimax	0.0083	121	10.74	1.07
2	Alpha-Beta	0.0078	108	-	-
3	Minimax	0.0639	856	39.14	1.33
3	Alpha-Beta	0.0481	521	-	-
4	Minimax	0.2535	3046	54.40	1.94
4	Alpha-Beta	0.1307	1389	-	-
5	Minimax	1.1192	14924	67.46	2.23
5	Alpha-Beta	0.5010	4856	-	-
6	Minimax	3.6996	44803	72.90	2.90
6	Alpha-Beta	1.2743	12146	-	-

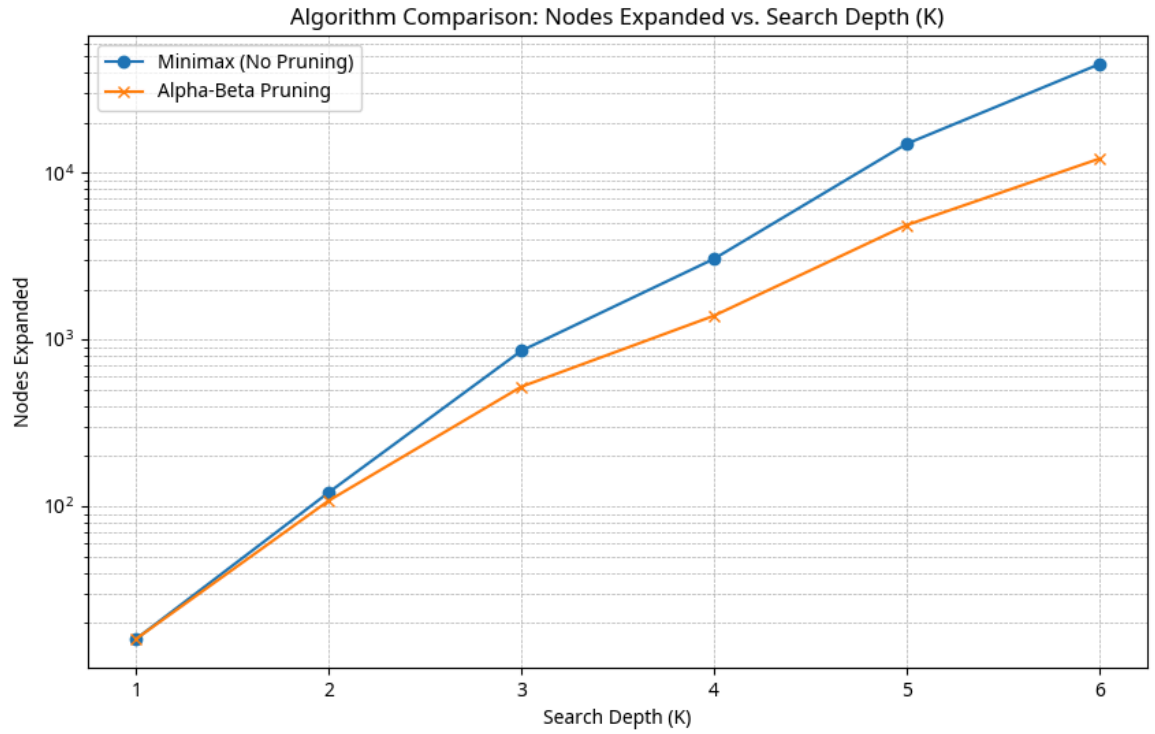
5.2. Visual Comparison

The following charts illustrate the exponential growth of both time and nodes expanded as the search depth K increases, and the significant performance gap created by Alpha-Beta Pruning. Note that the y-axis is on a logarithmic scale.

Time Taken vs. Search Depth (K)



Nodes Expanded vs. Search Depth (K)



5.3. Analysis of Results

The experimental results confirm the theoretical advantage of Alpha-Beta Pruning:

1. **Nodes Expanded:** The percentage of nodes pruned increases dramatically with depth. At $K=6$, Alpha-Beta Pruning reduced the number of nodes expanded by **72.90%** compared to the unpruned Minimax algorithm. This is the primary source of the performance gain.
2. **Time Taken:** The time taken for the search is directly proportional to the number of nodes expanded. At $K=6$, Alpha-Beta Pruning resulted in a **2.90x speedup** over the standard Minimax algorithm. This speedup is crucial for increasing the effective search depth within a fixed time limit, which is essential for a stronger AI player.

6. Extra Work

The comparison of the two algorithms across a range of search depths ($K=1$ to $K=6$) and the generation of comparative performance charts were performed as extra work to provide a comprehensive understanding of the efficiency gains offered by Alpha-Beta Pruning. The data clearly shows that the efficiency of Alpha-Beta Pruning improves as the search space grows larger (i.e., as K increases).

7. Conclusion

The implemented Connect Four AI successfully utilizes the Minimax algorithm, enhanced by **Alpha-Beta Pruning**, **Iterative Deepening**, **Move Ordering**, and a **Transposition Table**. The experimental comparison demonstrates that Alpha-Beta Pruning is highly effective, reducing the number of nodes expanded by over 70% at $K=6$ and providing a nearly 3x speedup, allowing the AI to search deeper and make better decisions in a practical time frame. The use of a well-tuned heuristic function further ensures that the AI's evaluation of non-terminal states is strategically sound.