Applying Minimax and Alpha-Beta Pruning to a Novel Connect 4 Variant

*

1st Gabriel Brehm
GMU College of Engineering and Computing
Fairfax, VA, USA
gabe.brehm@gmail.com

***Abstract*—This paper documents the use of minimax with Alpha-Beta pruning to create an artificially intelligent agent to act as a player opponent. A heuristic was created based on core objectives of the game, as well as algorithms to identify terminal game states. The agent was also given a varying difficulty level. The resulting agent was playtested using a simple scheme for varying first inputs, and the results are documented and displayed. Because the agent difficulty yielded a dichotomy of results, it is hypothesized that a method for varying agent difficulty based on randomization or deep learning to mimic human play might lead to a more interesting AI opponent for a player.**

***Index Terms*—minimax, Alpha-Beta pruning, adversarial search, game AI, Connect 4, artificial intelligence, zero-sum games, heuristics**

I. Introduction

Adversarial search is a type of solution-finding search problem that occurs in multiagent environments where the other agents have conflicting goals with the agent in question. These conflicting goals lead to competition. That goal of an agent in this space is therefore to not only achieve its own goals, but to prevent other agents from achieving their goals. [1]

In adversarial search, each level of the search tree corresponds to decisions made to alter the world state by one or more agents opposed to the agent(s) in the parent node. The search proceeds in such a way as to observe the outcomes of various possible decisions. The currently-choosing agent then uses the results of the search tree to produce contingency plans assuming optimal play from the opponent: that is, the opponent is assumed to make the best choice possible for itself. Therefore, the contingency plans are defined by the current agent's best response to the potential world state produced by an opponent, the opponent's best response to the response, and so on until either the entire game is mapped out or some computational limit is reached (e.g. time, memory, tree depth, etc.).

As minimax is a recursive algorithm, the key elements when solving a specific problem rest with the base cases:

1) Evaluation of terminal nodes.
2) Heuristic/cost function scoring for evaluation of max-depth nodes.
3) Maximum node depth.

Alpha-Beta pruning needs only to correctly pass the scores of the minimax evaluation along the recursive algorithm for application in a specific problem.

This project approaches applying minimax with Alpha-Beta pruning to a special variant of the usual Connect 4 game. The background discusses minimax and the Alpha-Beta pruning algorithms in greater depth, as well as the specific rules of the Connect 4 variant in question. The Proposed Approach section discusses the specific implementation of the algorithms used to create the artificially intelligent computer agent in the game, including heuristics and the agent intelligence level system. The Experimental Results section covers the experiment methods, purpose, and results. In the Conclusions, relevant inferences are made based on the experimental results, as well as potential future improvements and experimental expansions.

II. Background

The minimax algorithm used in this project implements the basic minimax algorithm directly (see "Fig. 1") by recursively producing a search tree of a given depth for an agent on its turn. The current agent is assumed to be MAX, an agent attempting to maximize some score. The opposing agent is then known as MIN, an agent attempting to minimize the same score. At each node, there is a check for an end game state (a victory/defeat for either player or some other terminating state). Non-terminal nodes not at the maximum depth are recursively expanded. Non-terminal nodes at maximum depth are evaluated using some cost function or heuristic. The agents then observe the choices available to them and make the choice that corresponds to their algorithmic role (MIN always chooses the smallest available value, MAX the largest). This information is passed up the search tree until finally, the MAX player is able to select the best-value move available (assuming optimal play from the opponent). See "Fig. 2" for an example.

This project also implements Alpha-Beta pruning in the minimax algorithm to improve compute time. Alpha-Beta pruning relies on the simple idea that no player will make a worse move than their current best move. Therefore, moves available to MAX yielding a smaller value than its current best move can be ignored, and moves available to MIN yielding a larger value than its current best move can be ignored. See

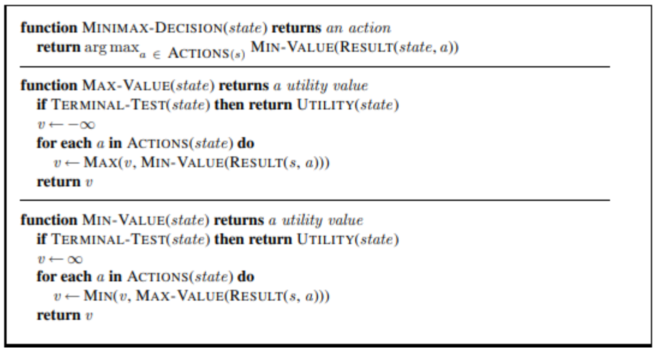
"Fig. 4" for an example, and "Fig. 3" for an outline of the algorithm.

Alpha-Beta pruning can reduce the number of node expansions/computations. The true minimax algorithm has a computation time on the order of that given in 1, where b is the "branching factor" (number of branches at each node) and d is the tree depth. Minimax with Alpha-Beta pruning, on the other hand, has a computation time between the standard minimax at worst case and that given in 2 at best case. [1]

$$O(b^d) \tag{1}$$

$$O(\sqrt{b^d}) \tag{2}$$

The game in question is a Connect 4 variant that might be called "Connect 4 in a Square." A winning state in the game involves 4 same-color pieces in a square, as seen in "Fig. 5". A "cat's game" occurs when neither player achieves victory and no open spaces remain, which is a possible terminal state for the game. No other game states receive a score or carry consequence in the game.



```
function MINIMAX-DECISION($state$) returns $an\ action$
    return $\arg\max_{a\, \in\, \text{ACTIONS}(s)}$ MIN-VALUE(RESULT($state, a$))

function MAX-VALUE($state$) returns $a\ utility\ value$
    if TERMINAL-TEST($state$) then return UTILITY($state$)
    $v \leftarrow -\infty$
    for each $a$ in ACTIONS($state$) do
        $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT($s$, $a$)))
    return $v$

function MIN-VALUE($state$) returns $a\ utility\ value$
    if TERMINAL-TEST($state$) then return UTILITY($state$)
    $v \leftarrow \infty$
    for each $a$ in ACTIONS($state$) do
        $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s$, $a$)))
    return $v$
```

Fig. 1. Outline of the minimax algorithm. [1]

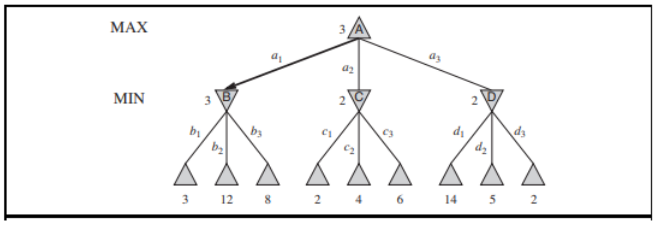
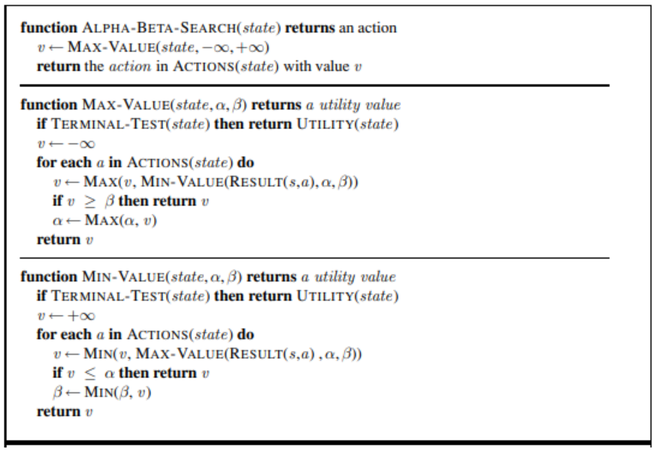


Fig. 2. Example of minimax at work. [1].

III. Proposed Approach

As stated in the Introduction, Alpha-Beta pruning in minimax is a recursive algorithm whose specifics depend on its base cases: the maximum depth, the game's terminal state(s), and the function to evaluate max-depth, non-terminal states (refer to "Fig. 3" for an outline of the algorithm).

The maximum depth was selected based on both computational limitations of hardware and light trial-and-error by the author. It was simply noted that on the author's hardware,



```
function ALPHA-BETA-SEARCH($state$) returns an action
    $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
    return the $action$ in ACTIONS($state$) with value $v$

function MAX-VALUE($state, \alpha, \beta$) returns $a\ utility\ value$
    if TERMINAL-TEST($state$) then return UTILITY($state$)
    $v \leftarrow -\infty$
    for each $a$ in ACTIONS($state$) do
        $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT($s$,$a$), $\alpha$, $\beta$))
        if $v \geq \beta$ then return $v$
        $\alpha \leftarrow$ MAX($\alpha$, $v$)
    return $v$

function MIN-VALUE($state, \alpha, \beta$) returns $a\ utility\ value$
    if TERMINAL-TEST($state$) then return UTILITY($state$)
    $v \leftarrow +\infty$
    for each $a$ in ACTIONS($state$) do
        $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s$,$a$) , $\alpha$, $\beta$))
        if $v \leq \alpha$ then return $v$
        $\beta \leftarrow$ MIN($\beta$, $v$)
    return $v$
```

Fig. 3. Outline of the Alpha-Beta pruning algorithm. [1].

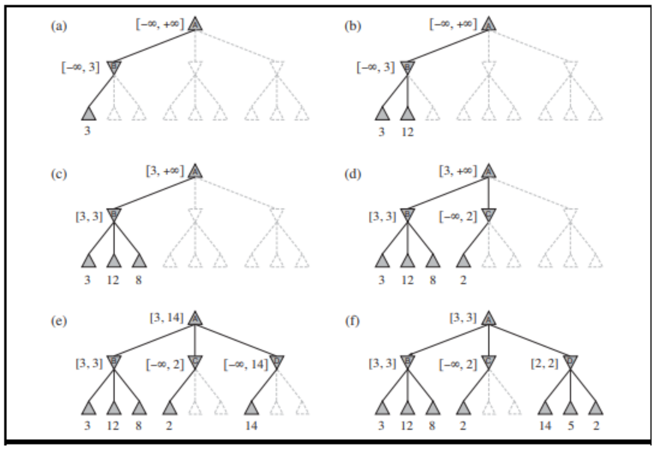


Fig. 4. Example of Alpha-Beta pruning at work. [1].

tree depths beyond 5 began to substantially slow the AI turns. Similarly, the AI able to access depths of 5 appeared to be nigh-unbeatable. Therefore, a tree depth of 5 was selected as the maximum. This aligned well with the requirement for the player to be able to select AI intelligence on a scale of 1 to 5, with 5 being the most intelligent. AI intelligence therefore corresponds directly to maximum tree depth.

Terminal states occur in one of two circumstances: either a player achieves a victory, or all spaces are taken with no victory claimed. In the case of the former, a winning move for the MAX player corresponds to an exceedingly large score, while a winning move for MIN corresponds to an exceedingly negative score. These can then be assigned to alpha and beta

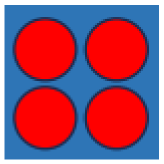


Fig. 5. A win in "Connect 4 in a Square".

values as necessary in the algorithm. In the case of the latter, both MIN and MAX observe a score of 0. This could be modified to attempt to produce an AI that is averse to ties; see the Conclusion for discussion.

Evaluating max-depth states was done via heuristic. With the goal being to produce a square, pieces on the edge can participate in half as many squares as pieces not on the edge (see "Fig. 6" and "Fig. 7" for examples).

Therefore, game states with more of the player's pieces near the edge penalized according to 3.

$$penalty = 2 * edge \tag{3}$$

The other heuristic involves the core goal of the game: creating a square. Board states with more horizontal pairs of player pieces in a row with empty spaces above are favored, as these are potential future victories. (see an example in "Fig. 8").

However, paired arrangements surmounted by an opposing piece can no longer lead to a win, and these arrangements give a score of 0 (see "Fig. 9" for an example).

Finally, grids of 4 with 3 pieces are heavily favored as these are only 1 move away from victory (see "Fig. 10" for an example).

These three factors are evaluated in a scoring function that works across each $2x2$ square on the board such that horizontal pairs of 2 horizontal are considered good, groups of 3 are heavily favored, and playing on the edge is discouraged. The function is given in 4.

$$bonus = 7^{Bottom} + 3 * (Top + Bottom) \tag{4}$$

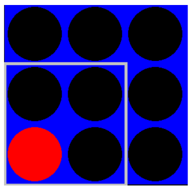


Fig. 6. A piece on the edge can participate in only one potential victory.

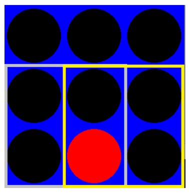


Fig. 7. A piece away from the edge might participate in either the gray or yellow victory square.

IV. Experimental Results

The experiment was carried out with the author always playing the first move and placing this first piece in each of the seven columns. The game was then continued with a reasonable attempt to win against the AI. This method was carried out for each column and AI level for a total of 35 experiments; the results can be seen in "Fig. 14".

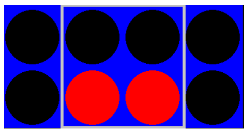


Fig. 8. A horizontal pair with open spaces can lead to a win.

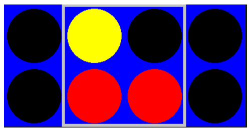


Fig. 9. Once a pair is surmounted by a single opposing piece, it can no longer lead to a win.

"Fig. 11" makes clear that difficulty had a dichotomous impact on difficulty for the player. At difficulty level (i.e. search depth) up to 3, the agent was unable to win a single game. At difficulty/depth 4 and 5, the AI won a single game, with the remainder being cat's games (ties).

The mean time of the game also increased dramatically at the same cutoff points, with the author having to expend more effort attempting to block opposing moves while formulating their own plans; see "Fig. 13".

Careful examination of "Fig. 13" also hints that the column played first may have also had an impact on game time. "Fig. 12" makes this more explicit by examining the relationship directly, and shows a clear decrease in game time after the first column is played. The simple explanation is that each AI difficulty was first encountered in the game played in the first column. As the player grew accustomed to the new difficulty level, the time spent on a game at that difficulty level evened out.

V. Conclusion

The experimental results make clear that with an agent able to peer past a certain depth, winning the game becomes very difficult for a human (or at least it does for the author, who concedes they are not the most apt player of this particular game). It is unclear how powerful the heuristics actually are, as the nature of the game seems to be biased towards the ability to block the opponent's pieces to prevent a win. This would indicate that the agent's ability to foresee an opponent's win, and its own, likely trumps any major heuristic measurements (although in the early stage of the game, it may be that

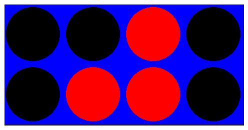


Fig. 10. Red is about to win here, so this receives the highest evaluation short of a victory.

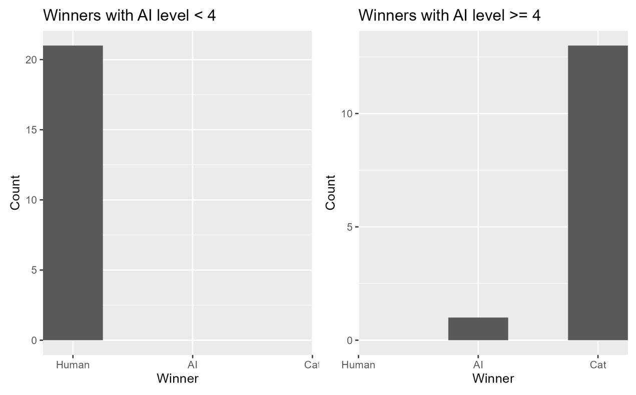


Fig. 11. Number of wins for each player depending on difficulty level.

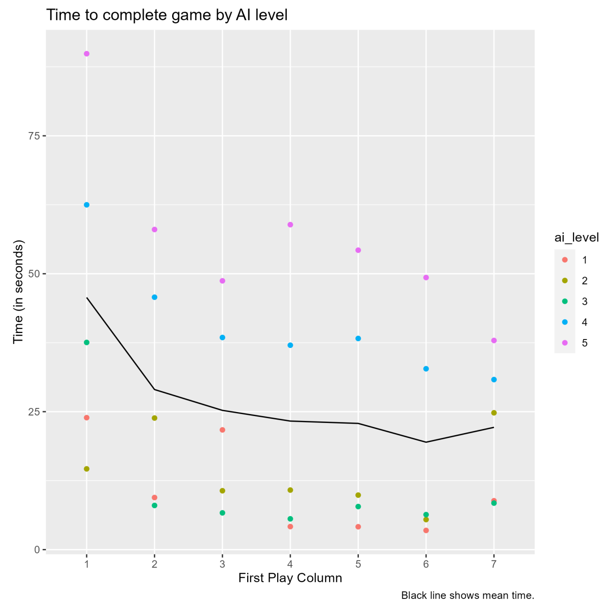


Fig. 12. Impact of first column on game time.

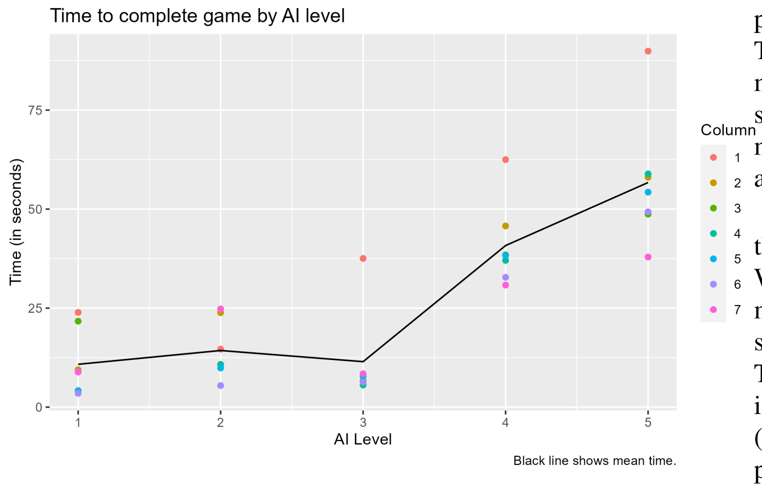
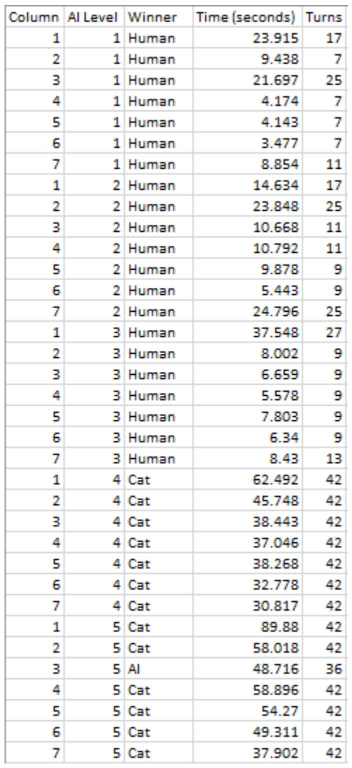


Fig. 13. Time vs. AI Level.



Column	AI Level	Winner	Time (seconds)	Turns
1	1	Human	23.915	17
2	1	Human	9.438	7
3	1	Human	21.697	25
4	1	Human	4.174	7
5	1	Human	4.143	7
6	1	Human	3.477	7
7	1	Human	8.854	11
1	2	Human	14.634	17
2	2	Human	23.848	25
3	2	Human	10.668	11
4	2	Human	10.792	11
5	2	Human	9.878	9
6	2	Human	5.443	9
7	2	Human	24.796	25
1	3	Human	37.548	27
2	3	Human	8.002	9
3	3	Human	6.659	9
4	3	Human	5.578	9
5	3	Human	7.803	9
6	3	Human	6.34	9
7	3	Human	8.43	13
1	4	Cat	62.492	42
2	4	Cat	45.748	42
3	4	Cat	38.443	42
4	4	Cat	37.046	42
5	4	Cat	38.268	42
6	4	Cat	32.778	42
7	4	Cat	30.817	42
1	5	Cat	89.88	42
2	5	Cat	58.018	42
3	5	AI	48.716	36
4	5	Cat	58.896	42
5	5	Cat	54.27	42
6	5	Cat	49.311	42
7	5	Cat	37.902	42

Fig. 14. Experimental results.

heuristics can be useful to set the agent up for a successful game overall).

An interesting approach to the AI may be to change its weightings regarding ties. The current algorithm is indifferent to ties, and the structure of the game seems to make victories difficult between players. However, by giving a tie a negative value approaching that of a defeat, it may be possible to push the agent towards more aggressive play that may up the win rate. Increasing search depth would also likely increase the win rate.

Minimax with Alpha-Beta pruning can sometimes be improved by focusing on searching better moves at the start. This is done by a separate heuristic that evaluates likely good moves for node expansion, rather than to evaluate the board state itself. Searching ”killer moves” or other good moves might speed the pruning process and allow for a more effective agent while maintaining lower average computer times. [1]

In terms of the agent as an opponent in a computer game, this agent is likely not a satisfying experience for a player. While exploring the aspects of AI behavior that make games more fun is beyond the scope of this paper, it suffices to say that continuous losses are usually not fun for humans. To improve the AI’s behavior in this regard, it may be more interesting to mix in random moves past a certain tree depth (such that the agent can block immediate wins but sometimes plays randomly otherwise) to allow the player a fighting chance. For a potentially more “realistic” experience, a deep learning model could be used to ascertain key aspects of human play and create an artificial agent that replicates these nuances. Chess AI such as Maia have become more interesting opponents for humans, and could serve as a reference for further work [3].

References

[1] S. Russsell, P. Norvig, "AI: A MODERN APPROACH," 4th Ed. (US edition). UC Berkeley, 2022.

[2] Keith Galli, "How to Program a Connect 4 AI (implementing the minimax algorithm)", www.youtube.com. https://www.youtube.com/watch?v=MMLtza3CZFMab$_channel = KeithGalli.(accessedJul20, 2023)$.

[3] Reid Mcllroy-Young, Russell Wang, Siddhartha Sen, Jon Kleinberg, Ashton Anderson, "Learning Personalized Models of Human Behavior in Chess," https://www.microsoft.com/en-us/research/publication/learning-personalized-models-of-human-behavior-in-chess/, August 2020.