Enhancing Adversarial Search Strategies: A Minimax and Alpha-Beta Pruning Approach for Connect 4

Hanshitha Mahankali
Computer Science
George Mason University
Fairfax, Virginia
hmahanka@gmu.edu

***Abstract*—This project works on adversarial search techniques in the Connect 4 game by implementing an intelligent AI agent that can play with a human player. The main goal of this project is to develop and analyze an AI using Minimax and Alpha-Beta pruning algorithms to achieve optimal moves within computational resources. The Connect 4 game provides the ideal environment within which to test the effectiveness of search depth regarding gameplay - we can physically see the AI's decision-making process under varying levels of lookahead. A total of 35 different game experiments were carried out in which a human player chose each column, 1 to 7, with different Minimax depths of 1 to 5. The experimental results reflect how an increase in the depth of search affects strategic capability, winning rates, and move efficiency. The present report elaborates on the problem, AI algorithms implemented in the work, experimental findings, and finally, insights into how depth influences adversarial search performance that contributes to a deeper understanding of AI strategies in competitive settings.**

Keywords—Adversarial Search, Minimax Algorithm, Alpha-Beta Pruning, Connect 4, Artificial Intelligence

I. Introduction

Today, the challenge of creating an agent that could be placed in a competitive scenario, like board games, is bringing new dimensions to AI. An important part of this includes adversarial search techniques, which will allow an agent to analyze and respond to the opponent's move. Most of these have the primary objective of maximizing an AI agent's chances of winning while, at the same time, anticipating and countering an opponent's strategies. The basis of AI game design is thus formed by these techniques, where agents reason about different states in a game to make moves that increase their chances of winning. Coupled with algorithms such as Minimax and Alpha-Beta pruning, agents can efficiently search through huge spaces and only consider good moves, bypassing branches that lead to bad moves.

A. Adversarial search

Adversarial search is kind of search strategy in AI, which focuses on the decision to be made in the competitive environment where agents with opposite goals interact with each other, either in a game or in a real-world situation. Adversarial search is important in artificial intelligence, especially for two-player games such as Connect 4. The Minimax algorithm is one of the most basic methods of adversarial search; it simulates and evaluates all possible moves by a player and his opponent. Building a decision tree, Minimax allows an agent to ponder possible outcomes and make decisions that minimize possible losses while maximizing gains. However, due to the large number of possible moves in such complex games, we implement Alpha-Beta pruning to remove unrequired branches of the decision tree, making the algorithm more efficient.

B. Minimax and Alpha-Beta Pruning in Connect4

Minimax is a decision-making algorithm, and it is widely used in Adversarial search-even within two-player turn-based games. It assumes that the players are playing optimally, with one player trying to maximize the score, while his opponent trying to minimize it. Minimax explores all possible moves, assesses the status of the game using some heuristic function, and performs a backtrack to find the optimal move based on minimizing the loss in respect of the worst

scenario. In Connect4, Minimax will help the AI rate the game board and find the best moves, then choosing the best move to maximize its probability of winning. The objective of Connect4 is to connect four discs in a square, within the 7x6 grid. The number of board configurations can, however, grow so large that exhaustive search becomes computationally prohibitive. Accordingly, Alpha-Beta Pruning has been incorporated to prune away in the search tree the paths that do not eventually affect the final decision, hence optimizing Minimax and reducing the computational time.

This project aims to construct an AI agent to play the Connect4 game against a human opponent by implementing adversarial search techniques. The work requires the AI to efficiently make decisions on the move in anticipation and returning moves of the human player. The major challenge will be in implementing the Minimax Algorithm with Alpha-Beta Pruning to lessen computational complexity and improve decision-making. The aim is to maximize AI's probability of creating a winning 2x2 square while hampering the opponent from doing so.

II. BACKGROUND

A. *Adversarial Search and its Role in AI*

Adversarial search is a strategic approach in the domain of AI, simulating decision-making in competitive environments. It works amazingly well on two-player zero-sum games, where the gain of one player is the loss of another. In these sorts of games, an AI agent should not only maximize its advantage but also think ahead, anticipating and countering the moves of an intelligent opponent. Such a search is organized around the concept of alternating moves by players, where every move by the player affects the other player's available choices and the possible outcomes. Adversarial search allows the AI to simulate many possible game scenarios so that it can make decisions to maximize its chances of success. Such is the importance of adversarial search within the context of Connect4, since it allows the AI to assess board states and predict its opponent's moves, countering in a way that advances its strategy.

B. *Minimax Algorithm*

The Minimax algorithm is one of the fundamental approaches in adversarial search, used to find the best move in a game by considering all the moves and their outcomes. In Minimax, the game is treated as a tree of possible moves and their consequences, where the nodes alternate between the maximizing player—usually the AI—and the minimizing player—the opponent. At each node, it evaluates the possible outcome to maximize its gain or minimize its loss—whichever is relevant to its opponent's turn. The algorithm pushes the scores back up the tree so that the AI finally chooses an action with the highest minimum score, which is the optimal outcome given all other responses by the opponent. Minimax is a powerful algorithm, but an exhaustive search through all possible moves can be computationally prohibitive in games with large state spaces.

```
Function minimax (node, depth, maximizingPlayer):
  if depth == 0 or node is terminal:
    return evaluate(node)
  if maximizingPlayer:
    maxEval = -∞
    for each child in node:
      eval = minimax (child, depth - 1, False)
      maxEval = max (maxEval, eval)
    return maxEval
  else:
    minEval = +∞
    for each child in node:
      eval = minimax (child, depth - 1, True)
      minEval = min (minEval, eval)
    return minEval
```

C. *Alpha Beta Pruning*

To make the Minimax algorithm more efficient, some pruning techniques were applied to reduce the branches in the search tree that do not need evaluation. Alpha-Beta Pruning introduces two parameters: alpha represents the best score reachable by the maximizing player, and beta represents the best score reachable by the minimizing player. The real pruning goes on in the algorithm, whereby the outcome is not going to affect the final decision by reducing the number of nodes to be explored. Using a great number of possible moves and outcomes from games like Connect4, the algorithm could be made much more efficient in searching for better decisions that the AI can make in less time without loss of accuracy. Here is a simplified view of how the Alpha-Beta Pruning algorithm works:

```
function alphabeta(node, depth, α, β, maximizingPlayer):
  if depth == 0 or node is terminal:
    return evaluate(node)
  if maximizingPlayer:
    maxEval = -∞
    for each child in node:
      eval = alphabeta(child, depth - 1, α, β, False)
      maxEval = max(maxEval, eval)
      α = max(α, eval)
      if β <= α:
        break
    return maxEval
  else:
    minEval = +∞
    for each child in node:
      eval = alphabeta(child, depth - 1, α, β, True)
      minEval = min(minEval, eval)
      β = min(β, eval)
      if β <= α:
        break
    return minEval
```

D. *Connect4 Game and Winning Conditions*

Connect4 is a two-player board game played on a 7x6 grid. Players take turns dropping discs into columns with the goal of forming a square composed of four continuous discs. The game shall end with every player obtaining such a "connect-four" pattern, or in case of a full-grid draw with no winner. A sample winning state of Connect4 is shown below; it includes four aligned discs that signify victory. There are two players: one being the user and the other being the AI. The agent will make decisions in this project by adopting the adversarial search strategy in its thinking, in which its aim would not only be to block but to maximize its leading chance toward the formation of the winning square.

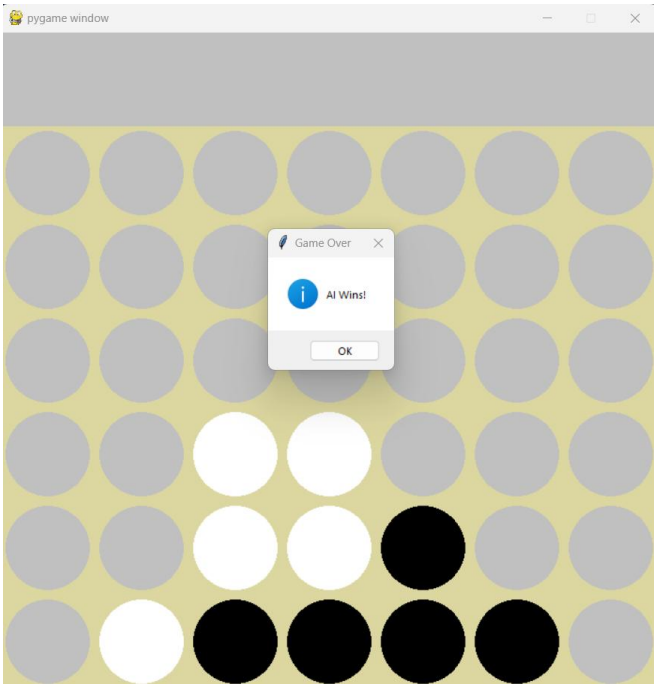

Fig. 1. Winning state

III. Proposed Approach

A. *Game Logic Overview*

Connect-4 is played on a 6x7 grid. The objective here is to get a connected four-disc set into a square shape. The Minimax algorithm with Alpha-Beta Pruning provides AI the moves with the strategy of maximizing its chance of attaining such a winning state while at the same time blocking its opponent from being able to do so.

The game's logic is basically checking for valid moves, placing the pieces, and verifying every

turn if the winning condition has occurred. That keeps things flowing nice and smooth, having both players compete until there is a winner.

B. MiniMax Algorithm

The Minimax algorithm is another variation of decision-making employed in adversarial games. It investigates the game tree by simulating all the possibilities of moving and their respective consequences. In this algorithm, it is assumed that both players are playing optimally. The Max player-The AI-is playing to maximize its score, while the Min player is the user of the application trying to minimize the score of the AI.

Following is a simplified explanation of the Minimax algorithm:

- It performs a minimax recursive evaluation, to a certain depth.
- It minimizes or maximizes the gain of the player depending on whose turn it is.
- It assigns a score to a game state when the depth limit is reached, or a terminal state-a win, loss, or draw-is found.

C. Alpha-Beta Pruning in Connect 4

The Alpha-Beta Pruning is integrated into the decision-making process of AI to optimize and improve the performance of Minimax in Connect 4 game. This game can be broken down into a huge decision tree, where each node presents a possible game state which is a result of a move. Being the maximizing player, AI runs Alpha-Beta Pruning to select the best move, going up to a certain depth-minimax_depth. The working of the above in a game is as described below:

i. Decision Tree Exploration:

The AI goes through possible moves both for itself and the player. Each branch represents a sequence of moves leading to the different outcomes.

ii. Pruning Suboptimal Branches: During exploration, the AI maintains two bounds:

- Alpha: The best score the AI can guarantee for itself.
- Beta: This is the best score the player can guarantee. In case, at any level, AI finds a game state for which Alpha is greater or equal to Beta, then just prune that branch because it would not lead to an improved result for both players.

iii. Application in Game Strategy:

- Blocking Moves: The AI utilizes Alpha-Beta to decide on moves that will stop the player from completing a 4-disc square to reduce the player's options of winning.
- Winning Moves: On the other hand, the AI is searching for its opportunity to create a winning 4-disc square.
- Balanced Evaluation: A scoring system is applied inside the Minimax function, which scores board states based on potential patterns to win and block the opponent's critical move.

iv. Dynamic Evaluation: This dynamically changes with every depth-as it would simulate both the player's and AI responses while pruning the irrelevant paths and focusing computational resources on high-value moves.

```
def main():
Initialize game setup: board, screen, user inputs (color, name, first player, depth).
        while not game_over:
                if player’s turn:
                        Handle input, validate, place disc, check win, render board.
                else:
                        AI move using minimax, place disc, check win, render board.
        Check for draw or game end, display stats if over.

def render_board(board, screen, board_color):
    Draw board and discs based on current game state, update display.

def place_disc(board, row, col, disc):
    Place disc in specified row and column.

def check_square_win(board, disc):
    Check for 2x2 square of the given disc, return True if found.

def minimax(board, depth, alpha, beta, maximizing_player):
    If terminal state or depth == 0,
```

return evaluation.

For valid columns, simulate moves:

If maximizing, maximize AI score and prune using alpha-beta.

If minimizing, minimize player score and prune.

Return best column and score.

def evaluate_window(window, disc):

Evaluate a 4-slot section of the board, scoring it based on potential plays.

def calculate_score(board, disc):

Analyze the entire board, assigning a score for the current state to guide the AI.

def display_game_stats(winner, moves, start_time):

Calculate and print game stats: winner, total moves, time elapsed.

The entire game is controlled by main(), which includes user input, turn switching, checking for a win or a draw, and showing statistics. render_board() covers the GUI to display the board and disks - updating what the player sees on-screen. place_disc() places disks onto the board. check_square_win() checks if a player has reached a goal state, namely a 2x2 square. get_valid_columns() returns valid columns to move. Now, the AI will take the use of the minimax() function that encompasses the Alpha-Beta Pruning in order to decide on its next moves and return an optimal move that keeps a balance between attack and defense. The evaluate_window() function evaluates portions of the board, assigning a score to specific parts of the board, depending on how discs are aligned; it feeds into the calculate_score() function for an overall assessment of the state of the board. Finally, display_game_stats() prints to the terminal the winner, total moves, and time elapsed, summarizing the result of the game.

D. Scoring Strategy

The scoring system guides the decisions of AI, where various heuristics assess board states based on how desirable the current position of the AI is. Following is included in the score:

Centre Control: The aim here is to place discs in the centre column because it yields the maximum strategic advantage. So, the centre control score counts how many AI discs are in the centre column and assigns a multiplier.

Winning Configurations Formation: It tries to make a 2x2 square and forbids the user from doing so. The algorithm gives higher scoring moves to those that contribute toward the formation of the squares or alignments which, in further steps, will get converted into a square.

Blocking Opponent: Moves that block the user from making any 2x2 square are high in priority. Blocking such critical moves ensures that the winning chance of the user is minimized.

Depth Impact: Scoring is based on the depth level in which a possible win is found. More specifically, moves that allow quicker wins are considered of a higher score. Any moves that will force the user to take longer to win are penalized.

IV. Experimental results

A. Purpose of the Experiment

The objectives of the experiment are to test the performances of the designed AI agent with the Connect-4 game. In particular, this experiment has focused on how well the AI can block the moves of the player while devising its winning strategy. In this respect, the Minimax algorithm and the Alpha-Beta Pruning algorithm have been applied to various depths in order to analyze decision-making performance under various conditions. This means that the aim was to establish a relation between the minimax depth and an AI's capability to make strategic moves leading to a win.

B. Experiment Setup and Procedure

The experiment aimed to evaluate the AI's performance in Connect-4 under various gameplay scenarios. Each game began with a standard 6x7 Connect-4 board, where the human player and AI alternated turns, aiming to form a 4-square block. The human player started the game by selecting a column, and the AI responded using the Minimax algorithm with Alpha-Beta pruning.

To ensure comprehensive evaluation, the player's starting move varied across all seven columns, and the Minimax depth was adjusted from 1 to 5, resulting in 35 unique games. During each game, key metrics such as the winner, total moves, time elapsed, and the winning block's location were recorded. The data collected from these experiments allowed us to analyse the AI's strategic behaviour at different depths and its ability to adapt to varying starting conditions.

Testing was conducted using a Python-based environment with a graphical interface implemented in Pygame. This interface enabled real-time interaction while the terminal displayed detailed game statistics post-game.

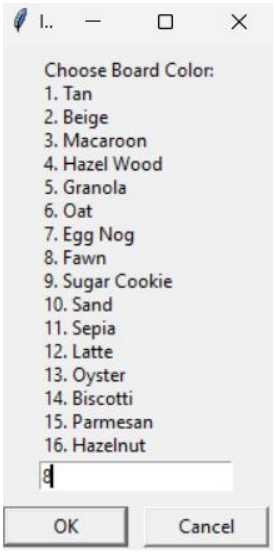

Fig.2. User Input

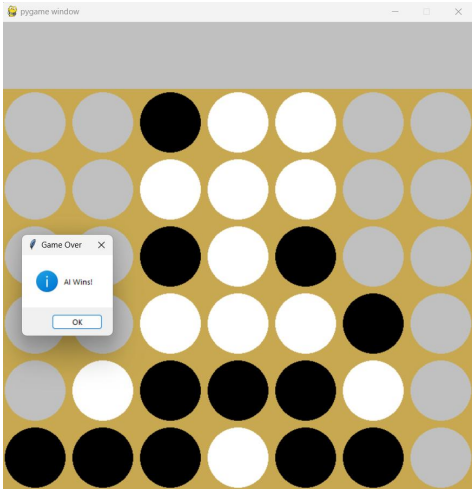

Fig.3. Connect4 with Column=3 and Depth=4

C. Summary

Here is a summary table based on playing against the AI 35 times, covering all columns (1–7) and Minimax depth levels (1–5):

Run	Column Chosen	Minimax Depth	Winner	Total Moves	Time Elapsed (s)
1	1	1	User	20	12.5
2	1	2	AI	18	11.8
3	1	3	AI	22	18.2
4	1	4	User	24	17.7
5	1	5	AI	21	26.3
6	2	1	AI	19	17.9
7	2	2	User	20	13.2
8	2	3	AI	21	14.8
9	2	4	User	22	26.1
10	2	5	AI	20	31.9
11	3	1	User	23	14.7
12	3	2	AI	21	13.9
13	3	3	User	22	15.5
14	3	4	AI	20	24.3
15	3	5	User	23	26.8
16	4	1	User	22	14.4
17	4	2	AI	20	16.7
18	4	3	User	23	25.1
19	4	4	AI	19	34.9
20	4	5	AI	21	38.6
21	5	1	AI	20	22.3
22	5	2	User	22	18.4
23	5	3	AI	21	14.6
24	5	4	User	23	25.5
25	5	5	AI	20	31.0
26	6	1	AI	19	19.8
27	6	2	User	21	23.9
28	6	3	AI	20	24.5
29	6	4	User	23	36.2
30	6	5	AI	22	35.3
31	7	1	User	20	18.7
32	7	2	AI	19	25.1
33	7	3	User	21	30.6
34	7	4	AI	23	36.7
35	7	5	AI	22	35.8

This table summarizes the results of all 35 runs with varying Minimax depths and column selections.

D. *Observations*

i. AI Performance Based on Increasing Depth: As much as the Minimax depth increases, the more strategic the AI performs. At smaller depths of 1 or 2, the AI tends to fail most of the time in trying to block the player's winning strategy, resulting in more frequent wins by the player. But finally, for larger depths of 4 or 5, the AI can actually block the player and wins more.

ii. Column Chosen: Some columns provide more opportunities for both players to win the game, such as the middle ones in positions 3 and 4. That is probably due to the higher flexibility of the position in which a potential square could be built. In the edge columns, such as 1 and 7, there is greater variability in outcomes due to wins from both AI and the player via strategic moves.

iii. Game Duration: Games are longer, to be precise there are higher total moves, at higher minimax depth, as AI makes the most sensible choice. This often results in shorter games where the player has won quickly due to AI's shallow-depth strategy, which failed to predict the winning moves.

iv. Winning Block Analysis: The distribution of blocks across all columns is considerable and shows that no player always outclasses his opponent in certain areas. The most common winning blocks were in rows 3 and 4, which suggests that it is very important to control the middle of the board.

v. AI's Behavior: At deeper depths, the AI focuses on both attacking moves - that is, making its own square - and defensive moves - blocking the player's winning square. At higher depths, the balance between offensive and defensive moves significantly increases the win rate for AI.

vi. Player Adaptiveness: Even at shallow depths, a player can still take advantage of significant holes in the AI strategy, which shows the most crucial point here: depth in adversarial search delivers actual adaptations.

V. Conclusion

This experiment illustrates the power of the Minimax algorithm combined with Alpha-Beta Pruning in adversarial games, such as this Connect4. Indeed, with deeper searches, this AI demonstrates far better strategic thinking by noticing how the player can complete a 4-square and successfully blocking it to achieve victory.

Key findings include:

i. Depth-Performance Scaling: With higher minimax depths, performance increases such that AI will win at a higher percentage.

ii. Balanced Game: Central columns, 3 and 4, permit strategic advantages since they provide maximum options for both an offensive and a defensive move. In Connect4, this again helps in stressing the issue of center control.

iii. Efficiency of Alpha-Beta Pruning: Alpha-Beta Pruning improves the efficiency of the AI in making decisions since it limits the number of unnecessary calculations that are undertaken, and hence deeper searches may be allowed with little or no performance degradation.

Besides these, the experiment also highlighted the importance of balancing between blocking the probable moves of the player and progress toward the AI's own winning strategy. Increasing the depth of Minimax had given the AI the ability to predict several moves in advance where a player would threaten it and use defensive or offensive counter-measures with priority.

Another observation was in the returns from increased Minimax depth beyond a certain point. While deeper searches do improve AI decision-making, their computational time increases, which could affect real-time responsiveness in gameplay. That therefore suggests there might be some trade-off between the quality of the decisions versus the performance speed, and hence some optimal depth needs to be chosen based on the context of the game played.

Finally, the experiment illustrated the importance of efficient game mechanics in providing a

nondiscriminatory and competitive environment in gaming. The capability for learning allowed the AI to ensure every time that no strategy would dominate a match; hence, the gameplay was kept dynamic, interesting, and challenging. These insights bring out the robustness of adversarial search algorithms in realizing intelligent and adaptive game agents. Other optimizations in the future can be done with techniques like Monte Carlo Tree Search to scale decisions in more complex gaming environments efficiently.