Using the Mini-max Algorithm and Alpha-Beta Pruning to solve Connect-N (based on Connect-4)

Bharathi Subramanian Nagarajan
2110110162, bn871
Computer Science and Engineering
Shiv Nadar Institute of Eminence

Namit Arjaria
2110110344, na168
Computer Science and Engineering
Shiv Nadar Institute of Eminence

Sanjay Sathish
2110110460, ss219
Computer Science and Engineering
Shiv Nadar Institute of Eminence

Shruti Bansal
2110110492, sb398
Computer Science and Engineering
Shiv Nadar Institute of Eminence

Tejaswi M
2110110556, tm988
Computer Science and Engineering
Shiv Nadar Institute of Eminence

***Abstract*—In this project, we implemented a Connect-N AI using the Minimax algorithm with Alpha-Beta pruning, enhancing the classic Connect 4 game by introducing the goal of connecting N discs. The methods uses tree traversal, maximizing and minimizing player strategies, terminal node evaluation, and Alpha-Beta pruning. We explored parameters like width, height, and N, influencing game dynamics, and incorporated heuristics for effective decision-making. Despite some limitations, the artifical intelligence [AI] demonstrates strategic gameplay in Connect-N.**

***Index Terms*—Artificial Intelligence, Minimax, alpha-beta pruning, Connect-N**

I. Introduction

Connect-N is a generalization of the classic game Connect 4, introducing a variation where players aim to connect N dots in a row instead of the traditional four. The game is played on a rectangular grid, typically 7 columns by 6 rows, though the dimensions can be adjusted for different difficulty levels. Two players take turns dropping their colored discs (usually red and yellow) into the columns. The disc falls to the lowest available position within the chosen column.
The primary objective is to be the first to create a horizontal, vertical, or diagonal line of N discs of the same color. The challenge lies in strategically placing discs to block opponents while forming your own winning sequences. The game ends when a player achieves the required connection or when the grid is full, resulting in a draw.

The Minimax algorithm [1] is a decision-making strategy commonly used in two-player turn-based games like Connect-N. Its primary goal is to determine the optimal move for a player by considering all possible future moves and their outcomes. The algorithm operates on a game tree, where each node represents a game state, and the edges represent possible moves. The minimax algorithm generally includes the following features:

A. Tree Traversal

- The algorithm explores the game tree recursively, starting from the current game state.
- At each level of the tree, it alternates between maximizing and minimizing players.

B. Maximizing Player

- The player seeking the maximum outcome (usually the AI) aims to select the move that leads to the highest score or utility.
- The algorithm assigns a score to each possible move and selects the move with the maximum score.

C. Minimizing Player

- The opponent aims to minimize the score or utility for the maximizing player.
- The algorithm assigns a score to each possible move and selects the move with the minimum score.

D. Terminal Nodes

- When the algorithm reaches a terminal node (end of the game or a specified depth in the tree), it evaluates the utility of that game state.

E. Alpha-Beta Pruning

- Alpha-Beta pruning, while not a feature of the Minimax algorithm, is an optimization technique used to reduce the number of nodes evaluated by the Minimax algorithm.
- It maintains two values, alpha (the best value for the maximizing player) and beta (the best value for the minimizing player).
- If at any point during tree traversal, it is determined that further exploration of a subtree is unnecessary (based on alpha and beta values), that subtree is pruned.

The combination of the Minimax algorithm and Alpha-Beta pruning makes Connect-N AI more efficient by avoiding unnecessary computations and focusing on the most promising

moves in the game tree. In the following sections, we will discuss how we have implemented each of the above mentioned features in our project.

II. Methods

Now, we will look into each of the above methods mentioned above and their implementation for the Connect-N problem.

A. Tree Traversal

The algorithm begins by traversing the game tree recursively, starting from the current game state. In the provided code, this is implemented in the *minimax* function. The function explores all possible moves by iterating over valid column locations and simulating the placement of a piece in each column. It alternates between maximizing and minimizing players, delving deeper into the tree until it reaches the specified depth or a terminal node.

Algorithm 1: Minimax Tree Traversal

```
Minimax(board, depth, α, β, maximizingPlayer)
if depth = 0 or IsTerminalNode(board) then
    return (None, EvaluateScore(board, AI_Piece))
if maximizingPlayer then
    return MaximizingPlayer(board, depth, α, β)
else
    return MinimizingPlayer(board, depth, α, β)
```

B. Maximizing Player

The maximizing player, in this case, represents the AI player seeking the highest score or utility. The algorithm assigns a score to each possible move and selects the move with the maximum score. This is achieved by iterating over valid column locations, simulating the placement of an AI piece, and recursively calling the *minimax* function with the updated game state.

C. Minimizing Player

The minimizing player, representing the human player, aims to make moves that minimize the AI player's score. The algorithm iterates over valid column locations, simulates the placement of a player piece, and recursively calls the *minimax* function with the updated game state.

D. Check Terminal Nodes

The algorithm checks whether the current game state is a terminal node, indicating the end of the game. A terminal node is reached when there is a winning move for either the AI or the player, or when there are no more valid moves left.

Algorithm 2: Maximizing Player

```
MaximizingPlayer(board, depth, α, β) value ← −∞
column ← RandomChoice(GetValidLocations(board))
forall col in GetValidLocations(board) do
    row ← GetNextOpenRow(board, col)
    b_copy ← CopyBoard(board)
    DropPiece(b_copy, row, col, AI_PIECE)
    new_score ←
      Minimax(b_copy, depth − 1, α, β, False)[1]
    if new_score > value then
        value ← new_score
        column ← col
    α ← Max(α, value)
    if α ≥ β then
        
return (column, value)
```

Algorithm 3: Minimizing Player

```
MinimizingPlayer(board, depth, α, β) value ← ∞
column ← RandomChoice(GetValidLocations(board))
forall col in GetValidLocations(board) do
    row ← GetNextOpenRow(board, col)
    b_copy ← CopyBoard(board)
    DropPiece(b_copy, row, col, PLAYER_PIECE)
    new_score ←
      Minimax(b_copy, depth − 1, α, β, True)[1]
    if new_score < value then
        value ← new_score
        column ← col
    β ← Min(β, value)
    if α ≥ β then
        
return (column, value)
```

E. Alpha-Beta Pruning

Alpha-Beta Pruning is a technique used to reduce the number of nodes evaluated in the minimax algorithm. It maintains two values, alpha and beta, representing the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of, respectively. The algorithm "prunes", or avoids evaluating branches that cannot possibly affect the final decision.

F. Depth of Traversal

The depth upto which we have chosen to let our AI model traverse the game tree is 4.

G. Parameters

The original game, Connect-4, consists of a vertical board with 7 columns and 6 rows. Here, 4 discs of the same color

Algorithm 4: Alpha-Beta Pruning

Minimax($board, depth, \alpha, \beta, maximizingPlayer$)
if $depth = 0$ *or IsTerminalNode(board)* **then**
  return (None, EvaluateScore($board, AI\_Piece$))
if *maximizingPlayer* **then**
  return MaximizingPlayer($board, depth, \alpha, \beta$)
else
  return MinimizingPlayer($board, depth, \alpha, \beta$)
MaximizingPlayer($board, depth, \alpha, \beta$) $value \leftarrow -\infty$
$column \leftarrow$ RandomChoice(GetValidLocations($board$))
forall *col in GetValidLocations(board)* **do**
  $row \leftarrow$ GetNextOpenRow($board, col$)
  $b\_copy \leftarrow$ CopyBoard($board$)
  DropPiece(b_copy, row, col, AI_PIECE)
  $new\_score \leftarrow$ Minimax($b\_copy, depth - 1, \alpha, \beta$, False)[1]
  if $new\_score > value$ **then**
    $value \leftarrow new\_score$
    $column \leftarrow col$
  $\alpha \leftarrow$ Max($\alpha, value$)
  if $\alpha \geq \beta$ **then**
return ($column, value$)
MinimizingPlayer($board, depth, \alpha, \beta$) $value \leftarrow \infty$
$column \leftarrow$ RandomChoice(GetValidLocations($board$))
forall *col in GetValidLocations(board)* **do**
  $row \leftarrow$ GetNextOpenRow($board, col$)
  $b\_copy \leftarrow$ CopyBoard($board$)
  DropPiece(b_copy, row, col, PLAYER_PIECE)
  $new\_score \leftarrow$ Minimax($b\_copy, depth - 1, \alpha, \beta$, True)[1]
  if $new\_score < value$ **then**
    $value \leftarrow new\_score$
    $column \leftarrow col$
  $\beta \leftarrow$ Min($\beta, value$)
  if $\alpha \geq \beta$ **then**
return ($column, value$)

must fall into 4 contiguous positions, either horizontally, vertically, or diagonally, for a player to win. In the generalization, Connect-N, there will be a variable number of rows and columns, as well as a variable number (N) of same-colored discs to be dropped in adjacent places for a winner to be declared.

- Width: The width, which will be the branching factor for this game, will be decided and set according to N, for each game played.
- Height: The height of Connect-N, representing the number of rows, is variable in contrast to the fixed 6 rows in Connect-4. A greater height allows more vertical space for disc placement, impacting the game's complexity and strategic options.
- N: In Connect-N, N signifies the consecutive discs required for victory, initially set at 4 in Connect-4. Adjusting N in Connect-N alters game difficulty, with higher values increasing the challenge by demanding longer sequences for a win. The chosen N significantly shapes gameplay, strategy, and overall game dynamics.

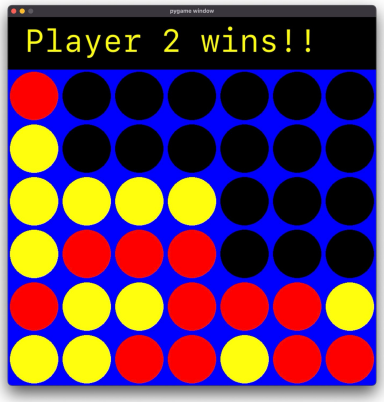

Fig. 1. Screenshot from a trial run of Connect-4 (AI Wins).

H. Heuristics

The evaluation of game states in Connect-N involves heuristics to assess the desirability of a given board configuration. These heuristics guide the AI player in making strategic decisions. The following heuristics are employed:

- **End Cases:** In definitive game outcomes, the heuristic value (h) is set as follows:
  - $h = -\infty$ when the opponent wins.
  - $h = +\infty$ when the AI player wins.
  - $h = 0$ when the board is full, and neither player achieves victory.

  These values act as terminal scores for the minimax algorithm, guiding it toward optimal moves in endgame scenarios.
- **Intermediate Cases:** For non-terminal states, a more nuanced heuristic is applied. The board is sent to the `getHeur()` function, which utilizes a specific formula. Factors such as the number of adjacent pieces in vertical, horizontal, and diagonal orientations. These intermediate heuristic values aid in evaluating and comparing different board configurations during the search process, contributing to the AI's decision-making process.

These heuristics provide a comprehensive framework for the AI to assess the current state of the game and make informed decisions based on the potential outcomes they represent. The combination of endgame scores and intermediate heuristics allows the AI to navigate the game tree efficiently and strategically.

III. Learning Outcomes

A. Current methodology

1) evaluate_window(window, piece):

- This function evaluates a window of consecutive positions in a row, column, or diagonal.
- It assigns a score based on the number of pieces (belonging to either the player or the opponent) in the window.

2) Scoring:

- If the window contains all pieces of the current player (piece), it adds a high score (100).
- If there is one empty space in the window and the rest are filled with the player's pieces, it adds a moderate score (5).
- If there are two empty spaces in the window and the rest are filled with the player's pieces, it adds a lower score (2).
- If the window contains all pieces of the opponent, and one empty space, it subtracts a penalty (4).

3) score_position(board, piece):

- This function calculates the total score for the entire game board by summing up scores for different types of windows.

4) Scoring:

- It gives extra weight to the center column, assuming it's a good strategy to control the center.
- It scores horizontally, vertically, and diagonally using the `evaluate_window` function.

B. Comparing our previous trials

- Connectivity vs. Patterns: The original system focuses on the general connectivity of pieces, while the pattern recognition system looks for specific patterns.
- Weighting Approach: The original system uses variable weights for different types of connections, while the pattern recognition system applies fixed weights for specific patterns.
- Special Case Handling: The original system has provisions for handling certain connections differently, while the pattern recognition system emphasizes scoring based on predefined patterns.
- Center Emphasis: The pattern recognition system puts a specific emphasis on the center column, considering it strategically important.

IV. Conclusion

We have explored the intricacies of Connect-N, an extension of the classic Connect 4 game that introduces strategic depth by aiming to connect N discs instead of four. Our focus has been on implementing an AI player using the Minimax algorithm, a decision-making approach for turn-based games. The Minimax algorithm, along with Alpha-Beta pruning, serves as our AI, providing an efficient way to explore potential moves.

In the methods section, we discussed each aspect of the Minimax algorithm, including tree traversal, maximizing and minimizing player strategies, terminal node evaluation, and the optimization technique of Alpha-Beta pruning. Each component has been presented with detailed pseudocode.

The discussion extended to the parameters influencing game dynamics, such as width, height, and N, illustrating their pivotal role in making the game experience. Additionally, we addressed the incorporation of heuristics, for evaluating board states and guiding the AI's decision-making process.

References

[1] G. Strong, "The minimax algorithm", Trinity College Dublin, 2011.

[2] Brenda Lim Geok San, Yap Jia Xin, Zailan Arabee bin Abdul Salam, and Chanpreet Kaur Dhanoa, "Connect-4 using Alpha-Beta pruning with minimax algorithm," Journal of Applied Technology and Innovation (e -ISSN: 2600-7304) vol. 6, no. 1, (2022) .

[3] Aman Pendyala, https://github.com/amancapy/csd311-material