# Playing Connect4 with Minimax & Alpha-Beta Pruning

Sai Manogyana T (G01377339)

**Introduction:**

This report takes an elaborate look at the approach used to solve the Connect4 game using AI implemented with the Minimax algorithm and the corresponding experimental results obtained. Since the minimax algorithm can be slow if the depth of the tree is large, alpha-beta pruning is also performed to cut down on how deep the tree can go.

Minimax algorithm is an approach used in adversarial search. Adversarial search is a kind of search used in competitive environments, in which two or more agents have conflicting goals. Each agent in this type of search tries to play optimally, making the best move it can make in its current state.

Connect4 game is a two-player game, in which one player is the user and the other player is the AI. For this implementation, we consider a player to have won the game if his pieces form a square. It is a perfect-information, The AI for this game is developed in such a way that it looks into the future through the usage of the minimax algorithm and chooses the best move possible for its current state. For a game with many possibilities, it becomes difficult to store all of these possible states in the memory, hence making alpha-beta pruning a useful technique to cut down on the memory usage and time complexity as well.

Background section of the report illustrates in detail the rules of Connect4 game and the winning move, along with the details of minimax and alpha-beta pruning algorithms. Proposed approach section of this report looks at the way this algorithm is implemented in detail and mentions the reasoning behind why some parameters were taken the way they were taken. Then, the experimental results section provides information about the different results obtained in 35 runs of the game with different depths of the minimax tree. Then, the report draws the major conclusions.

**Background:**

Minimax is a backtracking algorithm that is used in decision making and game theory to find the optimal move for a particular player, assuming that each player (opponent and current player) plays optimally.

The minimax algorithm computes the minimax decision from the current state. Since this is a backtracking based algorithm, it tries all possible moves, then backtracks and makes a decision.

Template of Minimax algorithm:

```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, depth − 1, FALSE))
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth − 1, TRUE))
        return value
```

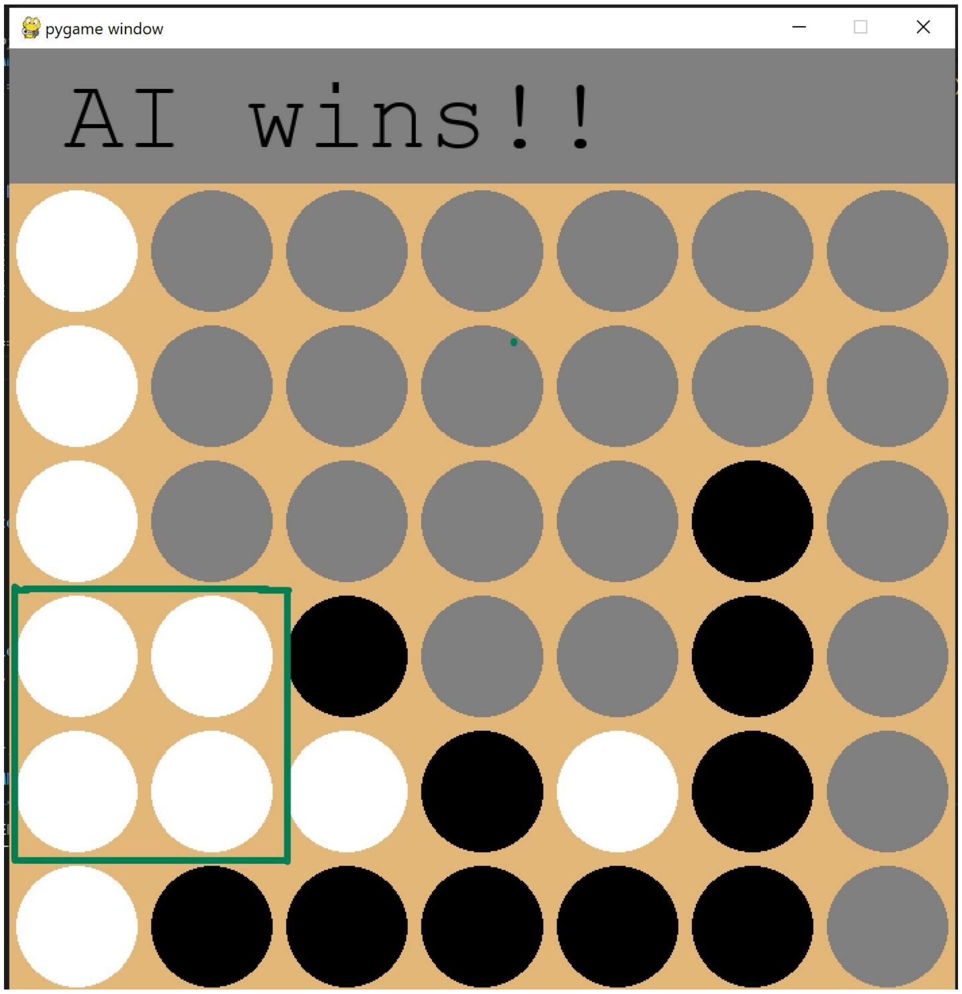
(Source: Wikipedia)

Alpha-beta pruning is a technique used to reduce the number of nodes that are explored in its search tree while executing the minimax algorithm. This technique is helpful because the minimax algorithm, without this pruning, would have to explore $O(b^m)$ nodes where m is the depth of the tree and b is the branching factor
Some of the branches of the decision tree produced by the minimax algorithm are useless, and the same result can be achieved if they were never visited. Alpha-beta pruning deletes useless branches and, in best-case, reduces the exponent to half. As a result, the algorithm becomes more efficient.

Template of Alpha-Beta Pruning:

```
function alphabeta(node, depth, α, β, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, alphabeta(child, depth − 1, α, β, FALSE))
            if value > β then
                break (* β cutoff *)
            α := max(α, value)
        return value
    else
        value := +∞
        for each child of node do
            value := min(value, alphabeta(child, depth − 1, α, β, TRUE))
            if value < α then
                break (* α cutoff *)
            β := min(β, value)
        return value
```
(Source: Wikipedia)

The modified version of the Connect4 game declares a player to be the winner if all his four pieces form a square anywhere on the board. The following picture illustrates the winning condition of the modified Connect4 game:



The part highlighted in green borders shows the square in connect4 game that declares a player the winner of the game.

**Proposed Approach:**

The implementation follows the steps outlined below:

- Takes the user input: Player's name, who plays first; user or AI, board color from 16 options, depth of the minimax search tree [1-5].
- Once the turn is decided, that player places their piece.

- When the AI has to place its piece, it generates all the possible states using drop_piece() function and checks which state produces best move by assigning scores to each state using score_postion() function.
- score_position() assigns scores to each window of size four in the form of a square based on how many pieces of the opponent or how many of its own pieces are present. Playing around with different values of this may produce different results.
- If either of the players win or there are no more valid moves to make (here, it means that there are no more empty locations), then the game is stopped and the game data (total time elapsed and the total number of moves till the end) are displayed.

**Experimental Results:**
The experiment was carried out 35 times in total. Each value of depth from1 to 5 is played 7 times with the initial column in which the piece was put varying from 1 to 7 in each play of the game. In each run, the player that plays first is always the user, not the AI.
The purpose of the experiment is to see how many times the AI has won the game at each “deep” value of the minimax search tree, for each column being the starting point of the game in each iteration.

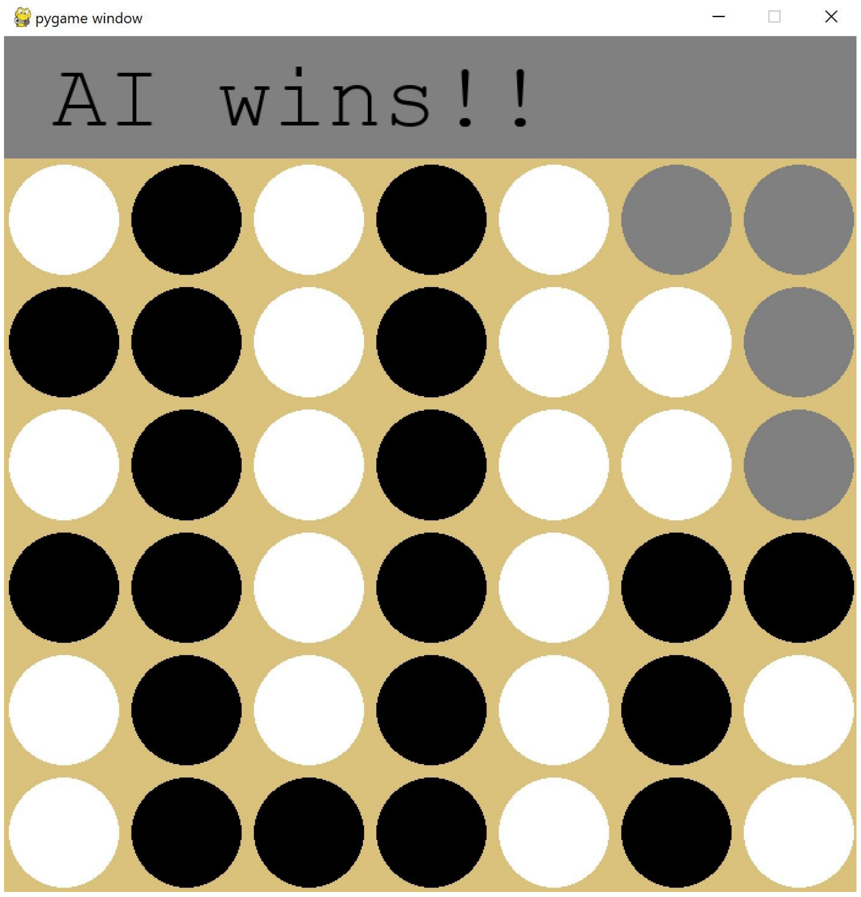
**Figure: Table Summary**

| Play No. | Turn | Starting Column | Deep value of minimax tree | Time elapsed | Total moves | Result |
|---|---|---|---|---|---|---|
| 1 | Player | 1 | 1 | 21.77 | 12 | AI wins |
| 2 | Player | 2 | 1 | 32.55 | 42 | Match Tie |
| 3 | Player | 3 | 1 | 17.78 | 16 | AI wins |
| 4 | Player | 4 | 1 | 13.25 | 8 | AI wins |
| 5 | Player | 5 | 1 | 28.52 | 24 | AI wins |
| 6 | Player | 6 | 1 | 24.06 | 42 | Match Tie |
| 7 | Player | 7 | 1 | 17.98 | 17 | Player wins |
| 8 | Player | 1 | 2 | 38.00 | 42 | Match Tie |
| 9 | Player | 2 | 2 | 29.95 | 20 | AI wins |

| | | | | | | |
|---|---|---|---|---|---|---|
| 10 | Player | 3 | 2 | 17.86 | 12 | AI wins |
| 11 | Player | 4 | 2 | 32.06 | 42 | Match Tie |
| 12 | Player | 5 | 2 | 18.88 | 12 | AI wins |
| 13 | Player | 6 | 2 | 23.39 | 32 | AI wins |
| 14 | Player | 7 | 2 | 26.39 | 34 | AI wins |
| 15 | Player | 1 | 3 | 23.00 | 42 | Match Tie |
| 16 | Player | 2 | 3 | 19.19 | 20 | AI wins |
| 17 | Player | 3 | 3 | 36.89 | 30 | AI wins |
| 18 | Player | 4 | 3 | 22.56 | 42 | Match Tie |
| 19 | Player | 5 | 3 | 28.98 | 42 | Match Tie |
| 20 | Player | 6 | 3 | 26.45 | 32 | AI wins |
| 21 | Player | 7 | 3 | 29.52 | 42 | Match Tie |
| 22 | Player | 1 | 4 | 22.22 | 42 | Match Tie |
| 23 | Player | 2 | 4 | 17.78 | 42 | Match tie |
| 24 | Player | 3 | 4 | 11.61 | 12 | AI wins |
| 25 | Player | 4 | 4 | 33.70 | 42 | Match Tie |
| 26 | Player | 5 | 4 | 32.89 | 42 | Match Tie |
| 27 | Player | 6 | 4 | 30.36 | 42 | Match Tie |
| 28 | Player | 7 | 4 | 18.11 | 42 | Match Tie |
| 29 | Player | 1 | 5 | 25.94 | 42 | Match Tie |
| 30 | Player | 2 | 5 | 19.81 | 40 | AI wins |
| 31 | Player | 3 | 5 | 27.05 | 42 | Match Tie |
| 32 | Player | 4 | 5 | 27.05 | 42 | Match Tie |
| 33 | Player | 5 | 5 | 25.36 | 42 | Match tie |

|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| 34 | Player | 6 | 5 | 22.17 | 40 | AI wins |
| 35 | Player | 7 | 5 | 16.92 | 18 | AI wins |

Relevant Figures:



**Conclusion:**

In conclusion, the usual time of the game ranged from [11.61, 38.00] seconds and the least number of moves needed were 8 whereas the highest were 42, when there was a tie in the match. In addition, it can be seen that out of 35 plays, 18 matches were a tie between the opponent and the AI, 16 matches were a win by the AI, and one match was a win by the player. It also has to be noted that the human player may not always play optimally though he may try to, which becomes advantageous to the AI.