# CS580 Introduction to Artificial Intelligence

# Project 2: Square in 4 with the Minimax Algorithm and Alpha–Beta Pruning

**Introduction:**

In this project we will explore the GUI design and minmax, alpha beta pruning algorithms for connect - 4 game. Most of the games use adversarial search techniques to reach the goal.

Searches in which 2 or more players with conflicting goals are searching the same search space to find a solution are called adversarial searches. In these types of searches tracing the movement of the enemy is important.

Min – max and alpha beta pruning is one of the adversarial search techniques implemented on the connect-4 game to connect 4 discs of the same color in a square. 2 players, an agent and the player explore the search space to connect 4 discs, we apply the min max algorithm with a max depth of 5 to efficiently make a move and win the game.

In this report we will investigate the background that focuses on the algorithm implementation, proposed approach that provides information about the strategy to assign points for the "deep" of the minimax algorithm. A small experiment is performed with different depths from 1 - 5 to arrive at conclusion.

**Background:**

**Explanation of Min-Max Algorithm:**

It is a recursive back tracking algorithm; it makes use of depth limited search strategy.

Leaf nodes are the terminal nodes, the terminal nodes are assigned with a heuristic value which determine if it is a win, lose or draw for the maximizing player, The heuristic value is a score measuring the favorability of the node for the maximizing player. Hence nodes resulting in a favorable outcome, such as a win, for the maximizing player have higher scores than nodes more favorable for the minimizing player.

Based on this value the maximizer tries to pick the highest possible value and assumes that the minimizer plays optimally and minimizes the value picked by the maximizer, an alternation of turns occurs and finally it picks the best state to make a move.

Pseudo code of Depth – Limited minmax 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
```

The time complexity for this algorithm is exponential O ($e^m$), if the search space is too large it might be difficult to compute the final state, hence we can prune the search tree that reduce the possible states to optimize the min max algorithm this pruning technique is called as the alpha-beta pruning technique.

**Explanation of Alpha-Beta pruning technique:**

Some of the branches of the decision tree are useless and the same value is achieved in the current node even if these branches are not visited so these branches can be removed from the search tree.

**Pseudo code Alpha beta pruning algorithm:**

```
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

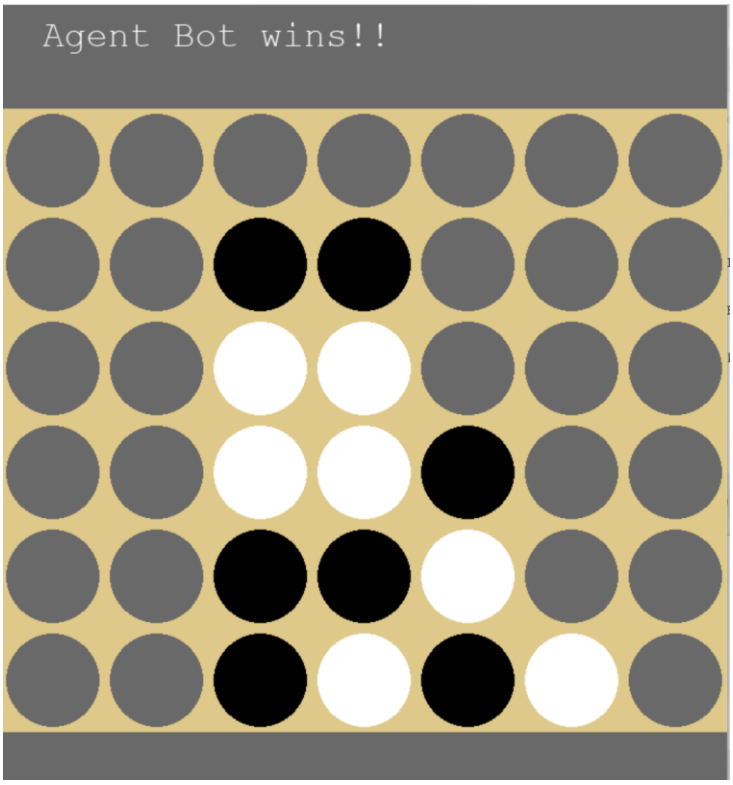
The algorithm maintains two values, alpha and beta, alpha represent the minimum score that the maximizing player is assured of and beta represent the maximum score that the minimizing player is assured of.

Initially, alpha is negative infinity and beta is positive infinity, i.e. both players start with their worst possible score. Whenever the maximum score that the minimizing player (i.e. the "beta" player) is assured of becomes less than the minimum score that the maximizing player (i.e., the "alpha" player) is assured of (i.e. beta < alpha), the maximizing player need not consider further descendants of this node, as they will never be reached in the actual play.

**Connect 4 game:**

Connect-Four is a two-player game in which players alternately place pieces on a vertical board 7 columns across and 6 rows high. Each player uses pieces of a particular color and the goal is to be the first to obtain four pieces in a horizontal, vertical, or diagonal line. Because the board is vertical, pieces inserted in a given column always drop to the lowest unoccupied row of that column. As soon as a column contains 6 pieces, it is full and no other piece can be placed in the column.

Both players begin with 21 identical pieces, and the first player to achieve a line of four connected pieces wins the game, the sample figure below shows how agent BOT has won the game by connecting 4 identical colors diagonally




**Proposed Approach:**

The minimax algorithm and alpha beta pruning approach is applied on the connect 4 game. The following is a high level overview on how to apply the minimax algorithm to the game. Define a game board with dimensions 6 rows and 7 columns The following functions are implemented for the connect 4 game

The function is_valid_location takes the current game board as input and a column number and returns True if the column is a valid move (i.e., the column is not full), and False otherwise.

The get_next_open_row function that takes as input the current game board and a column number and returns the row number of the next open space in that column.

The Drop_piece function takes input the current game board, a row and column number, and a game piece, and updates the game board by dropping the piece in the specified row and column.

The winning_move function that takes as input the current game board and a game piece and returns True if the game piece has won the game, and False otherwise.

The score_Position function that takes as input the current game board and a game piece and returns a score that represents the strength of the current board position for that player. The score can be calculated based on the number of connected pieces, the number of possible winning lines, or other heuristics.

The evaluate_board function takes the current game board as input and returns the score of the board position for the current player.

Firstly, we need to check for all the board states possible to make a move and choose the best possible state that will result in an efficient move for the AI.

Initially we make a call to the minmax algorithm - **minimax (node, depth, maximizing Player)** using the following parameters, the node is the current board state, with fixed depth of 5, assuming the maximizing player to be true if the player is AI and false otherwise.

Terminal nodes are the board states at a depth of 5 and these determine if the AI has won the game or lost the game or when no more possible moves are left. If a state is a win situation for the AI then this terminal state is assigned with a maximum possible heuristic value which is 100000000000000, else if it is not a winning move for the AI then it gets assigned with a minimum possible heuristic value – 10000000000000.

When the AI (maximizing player) has its turn it takes a max value from the descendants and assumes that the opponent plays optimally thus the opponent is responsible for minimizing the score. This process alternates for each layer and finally when a depth of 0 is arrived of all the available heuristic values the best value is chosen.

Alpha beta pruning technique is applied to prune the branches that are of no use to the maximizer function (i.e) pruning occurs whenever the alpha value is greater than or equal to beta, thus further exploring of the nodes is not required. Thus, the algorithms time complexity is increased.

**Template of the min-max algorithm used in the connect 4 game:**

```
function minimax(board, depth, alpha, beta maximizingPlayer) is
   if depth = 0 or node is a terminal node then
      return the heuristic value of nodes
   if maximizingPlayer then
      value := −∞
      col = random column
      for all the valid col's on the board
         by dropping the piece in each and every column
         find new_score := minimax(board, depth-1, alpha, beta, False)[1]
      if the new_score > value
         value = will be the new_score which is the maximum score
         column = col that is the best
      alpha is assigned with max(alpha,value)
      if alpha>=beta
       break without exploring the further branches of the search tree    // pruning occurs
       return value, col
```

```
else (* minimizing player *)
    value := ∞
   col = random column
   for all the valid col's on the board
      by dropping the piece in each and every column
      find new_score := minimax(board, depth-1, alpha, beta, True)[1]
  if the new_score < value
      value = will be the new_score which is the minimizing score
      column = col that is the best
  beta is assigned with min(beta,value)
  if alpha>=beta
   break without exploring the further branches of the search tree   // pruning occurs
   return value, col
```

**Experimental Results:**

We play against the agent to do an experiment. Where the player always makes the best move. In each game we will select each of the different columns [1–7] and each of the different values for deep [1–5]. We will Record who is the winner in each of the 35 different plays.

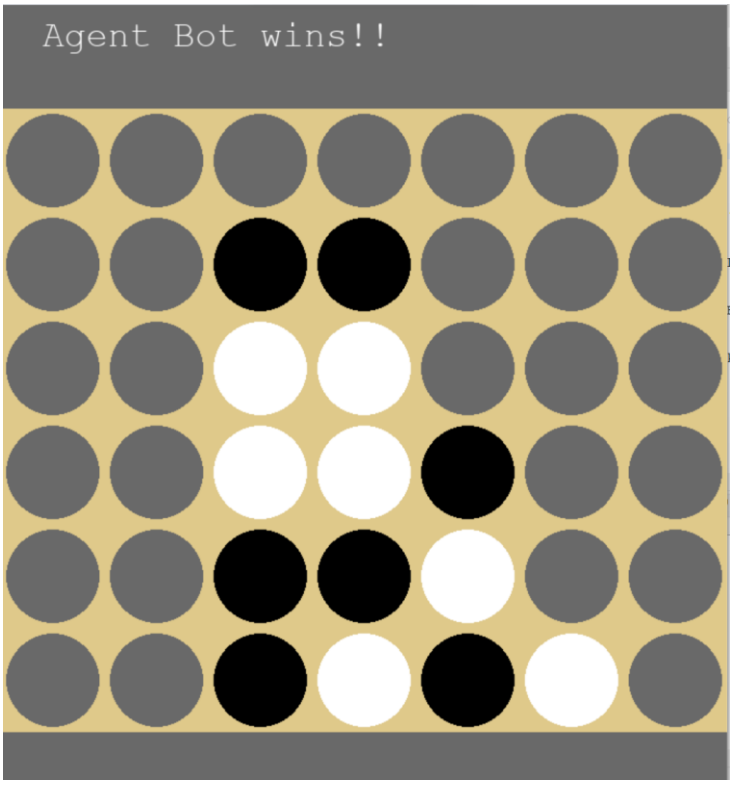
To perform this experiment Consider, for a deep of 4 there are 7 different cols in the board, since the player can make an initial move we release our disc into the first column and then perform subsequent moves by playing optimally and see if the player or the agent wins the game, similarly we consider second column as our first move and determine the result and so on. We repeat the same for 1-5 deep. The depth determine the difficulty level of the algorithm the higher the deep the more difficult the game play is.

**P – represents that the player has won the game**

**A - represents that the Agent has won the game**

**Mov – is the total number of moves to arrive at the result**

| Deep | Col1-Mov | Col2(Mov) | Col3(Mov) | Col4(Mov) | Col5(Mov) | Col6(Mov) | Col7(Mov) |
|---|---|---|---|---|---|---|---|
| **1** | A (12) | P(19) | P(15) | P(17) | P(21) | P(19) | P(11) |
| **2** | A (16) | A(12) | A(16) | P(19) | P(27) | A(20) | A(6) |
| **3** | A (26) | P(15) | A(12) | P(9) | A(30) | P(25) | A{16) |
| **4** | A (12) | A(20) | A(34) | A(14) | P(17) | A(20) | A(14) |
| **5** | A (12) | A(28) | A(28) | A(22) | A(24) | A(18) | A(12) |




**Conclusions:**

It has been inferred from the experiment that the higher the depth the difficulty level for the player was increased to win the game and vice versa if the depth was lower.

Thus, when the depth was 1 the player won in almost all the scenarios, when the depth was 2,3 also it was a bit easy to medium for the player to make a winning move. When the depth was 4 the difficulty level was medium to make a winning move, however when the depth was 5 it was equal to impossible to win the game and the agent won in all the cases on doing an optimal play.

Adversarial search technique has been used on the connect 4 game, it has used min max and alpha beta pruning techniques to compute the moves to be chosen by the AI, there are other possible methods to compute the result however if the search space is large to have an efficient game play, we applied minimax algorithm.