Lab 04

Game Playing in Artificial Intelligence

Submission Guidelines:

- Lab must be submitted in the google classroom.
- Submission other than google classroom won't not be accepted.
- You are required to submit a python (version 3 compatible) file and Notebook (Both) named after Your RollNo. e.g **LXX-XXXX.ipynb** and**LXX-XXXX.py**

Game Playing is an important domain of artificial intelligence. Games don't require much knowledge; the only knowledge we need to provide is the rules, legal moves and the conditions of winning or losing the game.

Both players try to win the game. So, both of them try to make the best move possible at each turn. Searching techniques like BFS(Breadth First Search) are not accurate for this as the branching factor is very high, so searching will take a lot of time. So, we need another search procedures that improve –

- **Generate procedure** so that only good moves are generated.
- **Test procedure** so that the best move can be explored first.

The most common search technique in game playing is **MinMax Search Algorithm**. It is depth-first depth-limited search procedure. It is used for games like **chess and tic-tac-toe**.

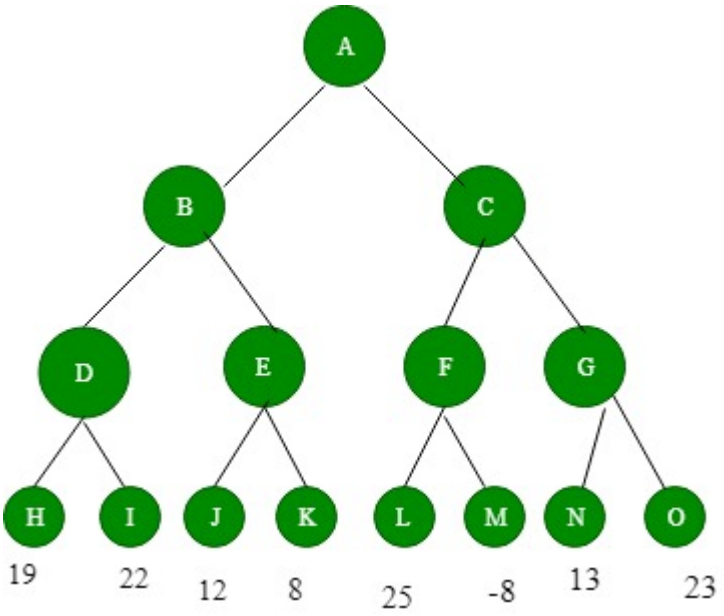
Minimax algorithm uses two functions –

MOVEGEN : It generates all the possible moves that can be generated from the current position.
STATICEVALUATION : It returns a value depending upon the goodness from the view point of wo-player.

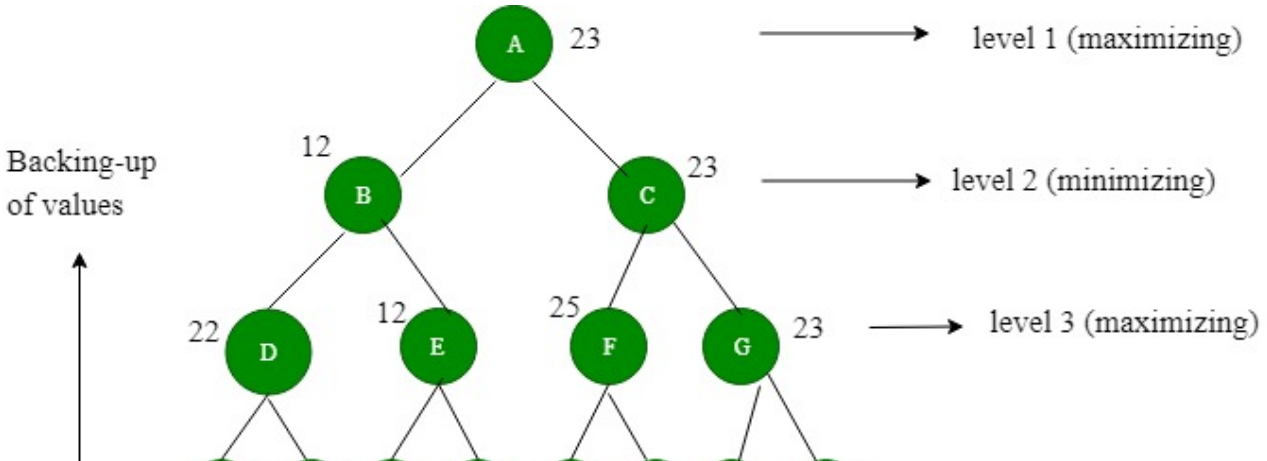
This algorithm is a two player game, so we call the first player as PLAYER1 and second player as PLAYER2. The value of each node is backed-up from its children. For PLAYER1

the backed-up value is the maximum value of its children and for PLAYER2 the backed-up value is the minimum value of its children. It provides most promising move to PLAYER1, assuming that the PLAYER2 has make the best move. It is a recursive algorithm, as same procedure occurs at each level.

Before Backing-up Values



After Backing-up values



Alpha-Beta Pruning

1. Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
2. As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning**. This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm**.
3. Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prunes the tree leaves but also entire sub-tree.
4. The two-parameter can be defined as:

    1. **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is **-∞**.
    2. **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is **+∞**.
2. The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

Condition for Alpha-beta pruning:

The main condition which required for alpha-beta pruning is:

α>=β

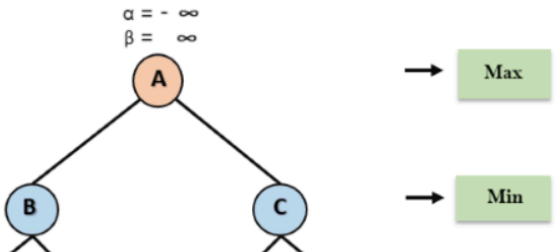
Key points about alpha-beta pruning:

- The Max player will only update the value of alpha.
- The Min player will only update the value of beta.
- While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
- We will only pass the alpha, beta values to the child nodes.

Working of Alpha-Beta Pruning:

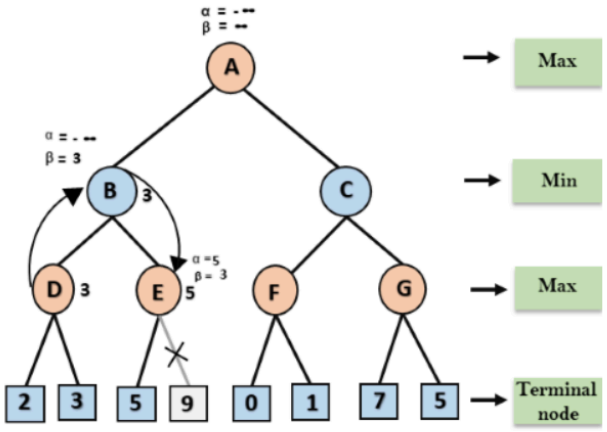
Let's take an example of two-player search tree to understand the working of Alpha-beta pruning

Step 1: At the first step the, Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.



In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α= -∞, and β= 3 will also be passed.

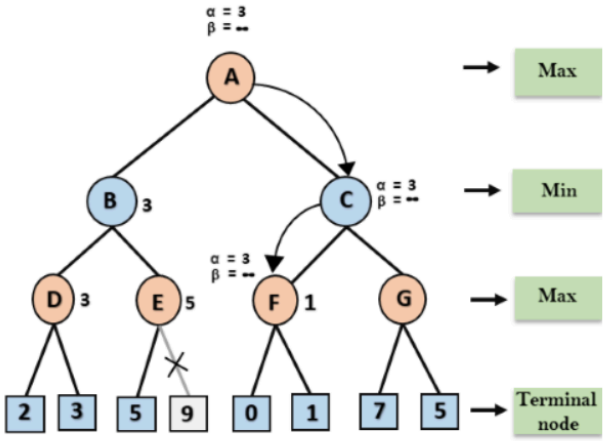
Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



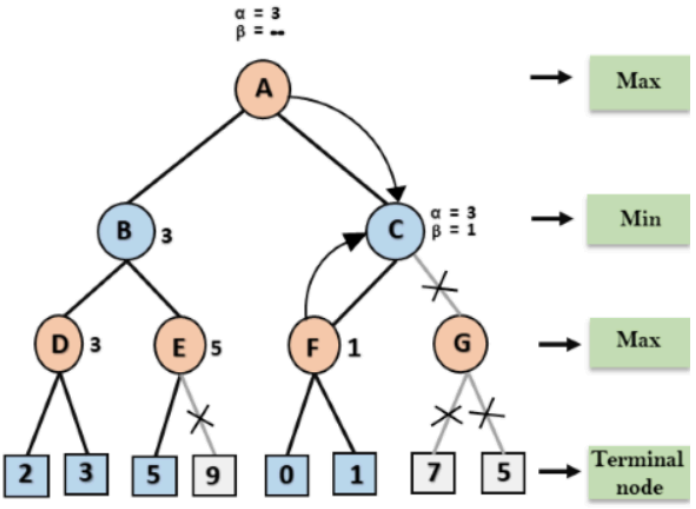
Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.

At node C, α=3 and β= +∞, and the same values will be passed on to node F.

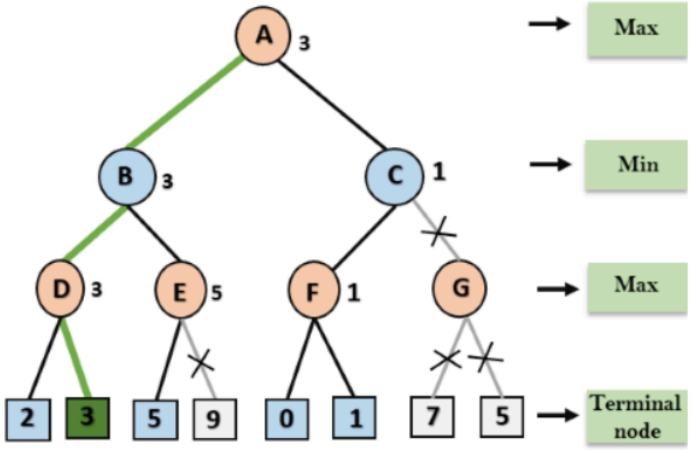
Step 6: At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1.



Step 7: Node F returns the node value 1 to node C, at C α= 3 and β= +∞, here the value of beta will be changed, it will compare with 1 so min (∞, 1) = 1. Now at C, α=3 and β= 1, and again it satisfies the condition α>=β, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.



Step 8: C now returns the value of 1 to A here the best value for A is max (3, 1) = 3. Following is the final game tree which is the showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.

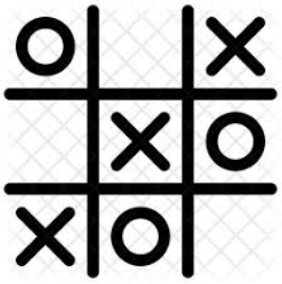


Lab Task

Artificially Intelligent TIC-TAC-TOE Game

Tic-tac-toe (also known as **noughts** and **crosses** or Xs and Os) for two players, X and O, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.

The sample tic tac toe board of order 3 by 3 is shown in the figure below. Suppose player 1 had chosen X and player 2 has chosen 0.



Problem Statement

You need to implement **Min-Max Algorithm** to search optimal solution for each player in Tic-Tac-Toe game. (Algorithm is given at the end for your reference)

Rules of Game :

The simple rules to play the game are:

- Two players can play the game. Each player will choose either 0 or X.
- The winning of a player is dependent on the consecutive 0s or Xs (in either row or column or Diagonal).
- Each player will play alternately. There is no biasness towards or against any player.
- The game will stop when either player has won.
- If no player has won and all the board cells are occupied, the game has been drawn.

Code Provided

def initial_state(): Returns starting state of the board.
def player(board): Returns player who has the next turn on a board.
def actions(board): Returns set of all possible actions (i, j) available on the board.
def result(board, action): Returns the board that results from making move (i, j) on the board.
def winner(board): Returns the winner of the game, if there is one.
def terminal(board): Returns True if game is over, False otherwise.
def utility(board): Returns 1 if X has won the game, -1 if O has won, 0 otherwise.

Task to Perform

1- **def alpha_beta_pruining(board):** Returns the optimal action for the current player on the board.