Lab Assignment-4 Report

I. Introduction

In this Report, I have summarized our experiments, observations, and results while performing experiments given to us in the labs

II. Week 5 - Lab Assignment-4

Learning Objective:
Game Playing Agent — Minimax — Alpha-Beta Pruning

III. Problem Statement :

What is the size of the game tree for Noughts and Crosses? Sketch the game tree.
Read about the game of Nim (a player left with no move losing the game). For the initial configuration of the game with three piles of objects as shown in Figure, show that regardless of the strategy of player-1, player-2 will always win. Try to explain the reason with the MINIMAX value backup argument on the game tree.
Implement MINIMAX and alpha-beta, pruning agents. Report on a number of evaluated nodes for the Noughts and Crosses game tree.
Use recurrence to show that under perfect ordering of leaf nodes, the alpha-beta pruning time complexity is O(bm/2), where b is the effective branching factor and m is the depth of the tree.

A. *Playing Agent*

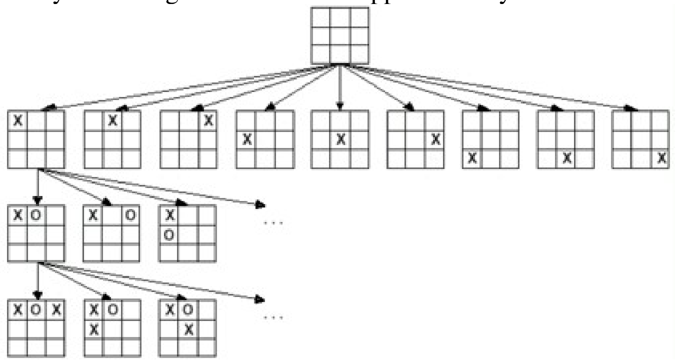
Playing agent in AI refer s to a computer program or sys tem designed to play games such as chess, Go, or video games. It uses various AI techniques such as game theory, reinforcement learning, deci sion making algorithms.

B. *Game Tree for Noughts and Crosses*

In the game of cross and noughts Initially all the places are empty. So, Tree Size = 1 Then there are total 9 possibilities for first move Tree Size = (1 + 9) Then for the next 9 states there are 8 possible moves . So, Tree Size = (1 + 9 + 9x8) Then for the next 9*8 states, there are 7 possible moves. So, Tree Size = (1 + 9 + 9x8 + 9x8x7) And it similarly it continues So, Tree Size = (1 + 9 + 9x8 + 9x8x7 + ... + 9x8x7x6x5x4x3x2x1) Tree Size = 9P0 + 9P1 + 9P2 +9P3 + ... +9P9 (npr = n!/r!) Sum of the series is 9!*e (e = 2.718...) Hence ,Tree Size = 9,86,410 But actually the tree size is less as many states get terminated by winning before even 9 moves so we dont go futher in that matrix So, Actual Size is around 549946.

IV. TIC TAC TOE

Tic-Tae-Toe is a two-player game that is played by marking X's and 0's on a 3X3 board. One who marks 3 X's or 3 0's in a straight line first, wins. As the game starts, P1 has 9 possibilities P2 then has 8 possibilities for each of the 9 possibilities, then 7 for each of the previous 8, and so on. Thus we will get around 10 lakh possibilities and thus a tree size of 10 lakhs. But, the game finishes when one of the two players wins. Thus most of the time we won't traverse all the nodes. Thus finally we will get a tree size of approximately 6 lakh nodes.



V. THE GAME OF NIM

In this game we have some objects arranged in different piles (3 piles in our case) and two players take turn to remove some objects from a pile. Rules A player can remove any number of object but from a single pile. The player left with no object to remove in their turn looses. Nim is considered a game of perfect information, meaning that both players have complete information about the game state at all times, including the number of objects in each pile.

A. *Winning Strategy*

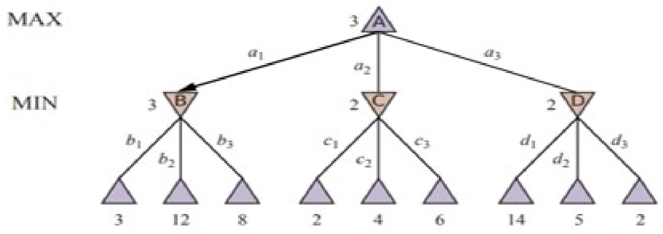
The xor of number of objects in each pile is called the nim sum. eg. if the piles are 1 2 3 nim sum = 1 xor 2 xor 3 hence nim sum = 0 The strategy is that to win this game a player should make the nimsum zero in their turn. eg. Piles: 1 2 4 nimsum = 7 we remove 1 from the third pile Piles: 1 2 3 nimsum = 0 If the nim sum is already zero there is no possibility that we can make a move to make nim sum 0 in a single turn. Hence if the nim sum is zero in the beginning player 2 will always win if played optimally else if the nim sum is non-zero in the beginning then the player who goes first always wins. In the given Question no. 2 the nim sum is initially not zero nim sum(10, 7 , 9) = 4 Hence player 1 can surely win if played optimally.

B. XOR Trick for Game of NIM

From our further search we came across the trick that if "XOR" of Initial configurations of piles is "0" then Player 2 always win if Player 2 plays optimally. And similarly, if "XOR" is non-zero then Player 1 always win if plays optimally.

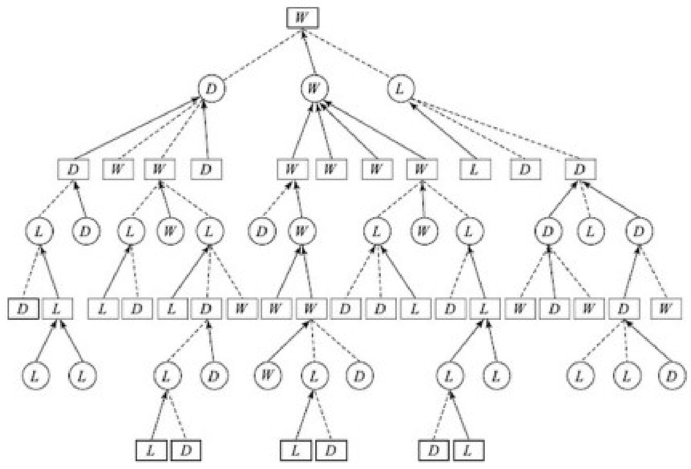
VI. Min Max Algorithm

It is a recursive algorithm that determines the best move using the tree-based approach to evaluate all possible moves. It is a decision-making algorithm that uses DFS where at each turn, the algorithm selects the move with maximum value for the current player and minimum value for the opponent. By alternating maximizing and minimizing the score, the algorithm ultimately finds the best move for both players. Time Complexity : O(b^m)



A. Algorithm

1-Start with the root node of the game tree and evaluate the value of its children.
2-If the node is a maximizing player, choose the move with the maximum value.
3-If the node is a minimizing player, choose the move with the minimum value. If the node is a terminal node, return its value.
4-Repeat the process for the selected child node, alternating between maximizing and minimizing players until a terminal node is reached.
5-The final value returned by the algorithm is the value of the root node, which represents the best move for the current player.

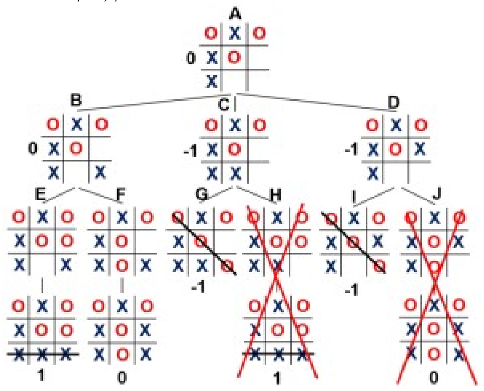


VII. Alpha – Beta Pruning

1-make the decision-making process more efficient by reducing the number of nodes that need to be evaluated.
2-works by pruning unproductive branches of the tree based on the Alpha and Beta values, which represent the best possible moves for maximizing and minimizing players respectively.
3-It starts by initializing the Alpha and Beta values to negative and positive infinity, respectively. Then, it evaluates the children of each node and updates the Alpha and Beta values based on the results.
4-If a branch of the tree is found to be unproductive, it is pruned, and the algorithm moves on to the next branch. This process continues until all nodes have been evaluated or pruned.
Time Complexity : O(b$^m/2$)

A. Alpha – Beta Pruning Time Complexity

The alpha-beta pruning time complexity can be shown using a recurrence relation as follows:T(m) = b * T(m-1) + O(1)
where T(m) is the time complexity of the algorithm at depth m, b is the effective branching factor, and O(1) is the constant time to evaluate each node. Thus, under the perfect ordering of leaf nodes, the alpha-beta pruning time complexity is O(b$^($m/2)).


A
B
C
D
E
F
G
H
I
J
0
0
-1
-1
-1
-1
1
0
1
0