Experiment No.: 04

Title: Implementation of Adversarial Search Algorithm (Min-Max)

Batch: **Roll No.:** **Experiment / Assignment / Tutorial No.: 04**

Aim: Implementation of Adversarial algorithm-Min-Max for Tic-Tac-Toe Game

Resources needed: C/ C++/ JAVA/ Python

Theory

The games form a separate category of multi-agent competitive environment. The solution steps cannot be taken in parallel, the steps cannot be undone. Because of all these problems normal informed or uninformed searching techniques will not help the agent to choose the actions at every instant. Fortunately, as the game rules are limited, that gives a limited a branching factor that helps to build the state-space. We need a methodology that will traverse this state-space, will not make a wrong move, and will help the agent to win the game in minimum time, using minimum space and time. That gives rise to find something similar to heuristic function and an algorithm that works on it.

Methodology: Just like the other problems in AI, the games are also considered the problems. The game playing agents define the game as a kind of search problem with the following components:

The initial state: identifies the board position and the player to move

A successor function: returns a list of (move, state) pairs, each indicating a legal move and the resulting state

Terminal test: Determines when the game is over Utility function: Numeric value for terminal states E.g. Chess +1, -1, 0

E.g. Backgammon +192 to -192

The initial state and the legal moves on each side define the game tree for the given game.

In the game tree, each level labeled with player to move. Each level represents a ply i.e. half a game turn. Thus it represents what happens with competing agents. Min-Max algorithm works on it well.

Min-Max Algorithm: this algorithm also uses the game tree where each level has the additional information as:

Max if player wants to maximize utility(Agent)

Min if player wants to minimize utility(opponent)

Also, every node is labeled with minimax value which is nothing but the Utility of node assuming players play optimally. Here, MAX wants to maximize utility, but knows MIN is trying to prevent that. So, MAX wants a strategy for maximizing utility assuming MIN will do best to minimize MAX's utility.

The Min-Max value can be calculated as:

Terminal

$$\text{Minimax-Value}(n) = \begin{cases} \text{Utility}(n) \\ \max_{s \in Succesors\ (n)} \text{Minimax-Value}(s) \\ \min_{s \in Successors\ (n)} \text{Minimax-Value}(s) \end{cases}$$

Max

Min

Thus it calculates the Min-Max value for each node recursively. Generally the first move is played by the Agent. Thus the agent starts in the initial state and then goes on following the branch that gives it the best utility values and same activity is also performed by opponent. i.e. at every turn, it chooses the path that gives the minimum utility value. The modes on Max(agent) ply are called Max nodes and otherwise Min nodes. For the multiplayer games, the first ply is Max(agent), then player 1, player 2, player n.. and so on.

<u>Implementation details:</u>

Implementation starts with a game tree stored in memory. Utility function is designed depending on the kind of game in discussion. The zero-sum games have utility values as:
Utility value: 1- if the agent wins
0- if game is a draw
-1- if opponent wins the game

The non-zero sum games are assigned the utility values on similar lines, where each node is labeled a utility value from the agent's point of view. It goes to the leaf nodes and assigns them the utility values. These values are backed up to their parents by examining whose ply it was: choosing the Max if it were Agent-ply and Min if it were Opponent-ply.

Procedure:

1. Implement two players, 1. AiPlayer and 2. HuPlayer [AI and Human player] for tic-tac-toe game.

2. For AiPlayer implement Minmax algorithm. [For simplicity first consider start state as given in the figure 2 below. Once program is working fine with this start state then change the start state to blank game board.]

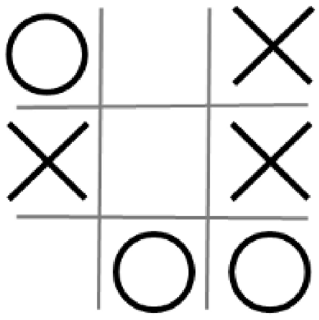

Figure 2: sample start state

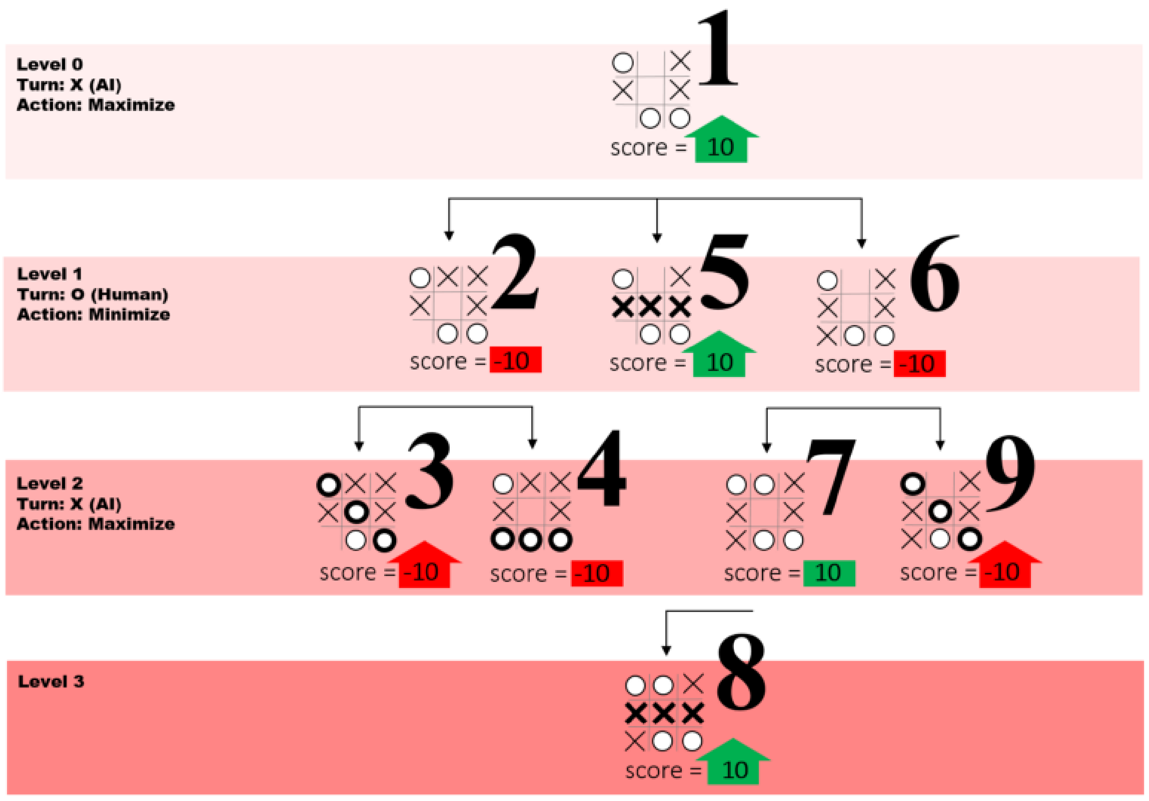

Figure 3 Minimax function call by function call

Results: (Document of program with output)

Questions:

1. Game playing is often called as an

a) Non-adversial search

b) Adversial search

c) Sequential search

d) None of the above

2. What are the basic requirements or need of AI search methods in game playing?

a) Initial State of the game

b) Operators defining legal moves

c) Successor functions

d) Goal test

e) Path cost

Outcomes:

Conclusion:

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

- How to make your Tic Tac Toe game unbeatable by using the minimax algorithm: https://www.freecodecamp.org/news/how-to-make-your-tic-tac-toe-game-unbeatable-by-using-the-minimax-algorithm-9d690bad4b37/#:~:text=A%20Minimax%20algorithm%20can%20be,on%20each%20available%20spot%20(recursion)
- Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, 2ndEdition, Pearson Publication
- Elaine Rich, Kevin Knight, Artificial Intelligence, Tata McGraw Hill, 1999.