National University of Computer and Emerging Sciences

Lab Manual 05
Artificial Intelligence Lab

Instructor: Mariam Nasim

1. Minmax Search Algorithm:

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.

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.

You can read more about it here: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

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 **AlphaBeta 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.

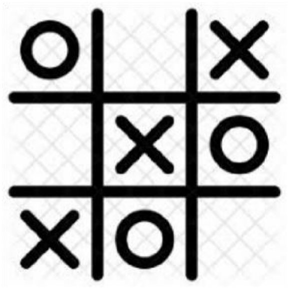
You can learn more about it here:

https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/

Task: TIC-TAC-TOE Game

Tic-tac-toe (also known as 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.



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.

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.

def minimax(board) , given a board, **returns the optimal action** for the current player. The algorithm first checks if the board is terminal, meaning the game is over and no actions can be taken. If not, it then check whose turn it is to take a move. IMPORTANT, if the AI plays for X, it tries to the pick the best out of all minimum values that have been selected from maximum values; likewise, it the AI plays for O, it tries to pick the worst out of all maximum values that have been selected from minimum values.

Tasks to Perform:

Implement all the above functions. Your goal is to make a Tic Tac Toe game, this algorithm faces an opponent that is playing against the machine. For example, in Tic-Tac-Toe, the AI plays against a human; the AI also **knows the complete state** of the game, which is a requirement for the Minimax Algorithm.

1. **def player(board): (3 marks)**
2. **def actions(board): (3 marks)**
3. **def result(board, action): (4) Add proper exceptions**
4. **def winner(board):** (4)
5. **def terminal(board):** (3)
6. **def utility(board):** (3)
7. **def minimax(board)** (10 marks)

Submission Instructions:

- Rename your Jupyter notebook to your “roll number_Name” and download the notebook as .ipynb extension.
- To download the required file, go to File->Download .ipynb
- Only submit the .ipynb file. DO NOT zip or rar your submission file
- Submit this file on Google Classroom under the relevant assignment.
- All outputs should be displayed properly.
- Late submissions will NOT AT ALL be accepted.