Department of Computer Engineering

Academic Term II : 22-23

Class: B.E (Comp A), SemVI **Subject Name: Artificial Intelligence**

Student Name: **Roll No:**

Practical No:	**5**
Title:	**Tic Tac Toe Problem solving by using Adversarial Search approach.**
Date of Performance:	
Date of Submission:	

Rubrics for Evaluation:

Sr. No	**Performance Indicator**	**Excellent**	**Good**	**Below Average**	**Marks**
1	On time Completion & Submission (01)	01 (On Time)	NA	00 (Not on Time)	
2	Logic/Algorithm Complexity analysis(03)	03(Correct)	02(Partial)	01 (Tried)	
3	Coding Standards (03): Comments/indention/Naming conventions Test Cases /Output	03(All used)	02 (Partial)	01 (rarely followed)	
4	Post Lab Assignment (03)	03(done well)	2 (Partially Correct)	1(submitted)	
Total					

Signature of the Teacher :

Experiment No: 5

Title: Tic Tac Toe Problem solving by using Adversarial Search approach.

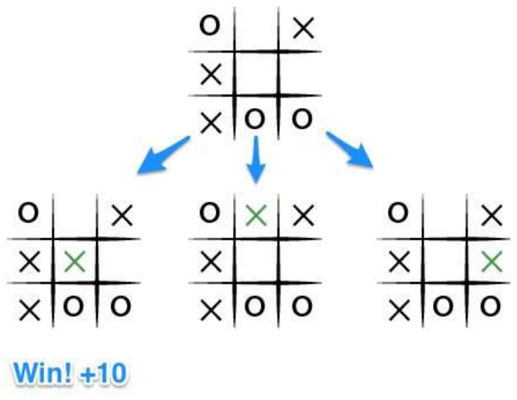
Objective: To write develop an AI for Tic Tac Toe problem that never loses

Theory:

The minimax search is especially known for its usefulness in calculating the best move in two player games where all the information is available, such as chess or tic tac toe. It consists of navigating through a tree which captures all the possible moves in the game, where each move is represented in terms of loss and gain for one of the players. It follows that this can only be used to make decisions in zero-sum games, where one player's loss is the other player's gain. Theoretically, this search algorithm is based on von Neumann's minimax theorem which states that in these types of games there is always a set of strategies which leads to both players gaining the same value and that seeing as this is the best possible value one can expect to gain, one should employ this set of strategies Working:

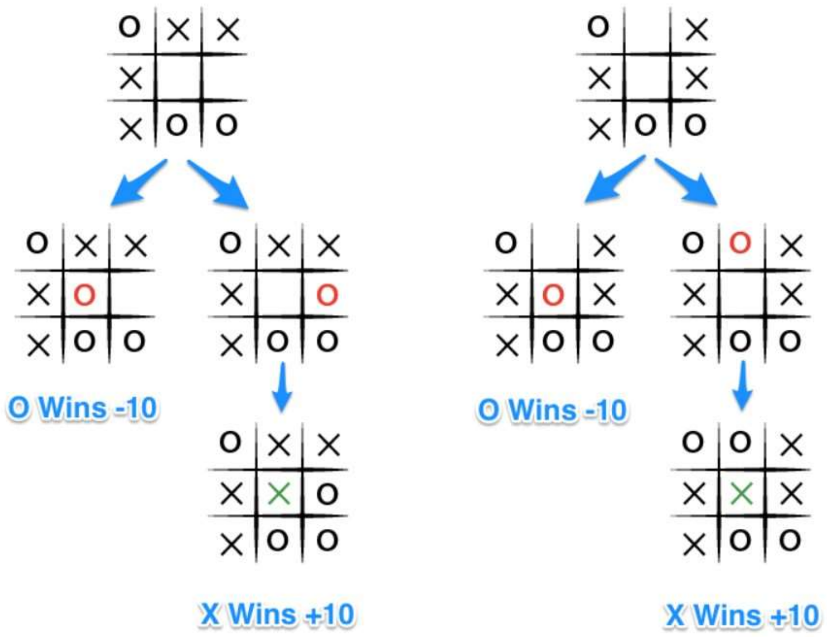
To apply this, let's take an example from near the end of a game, where it is X's turn. X's goal here, obviously, is to *maximize* his game score.

Consider the following state of the game:

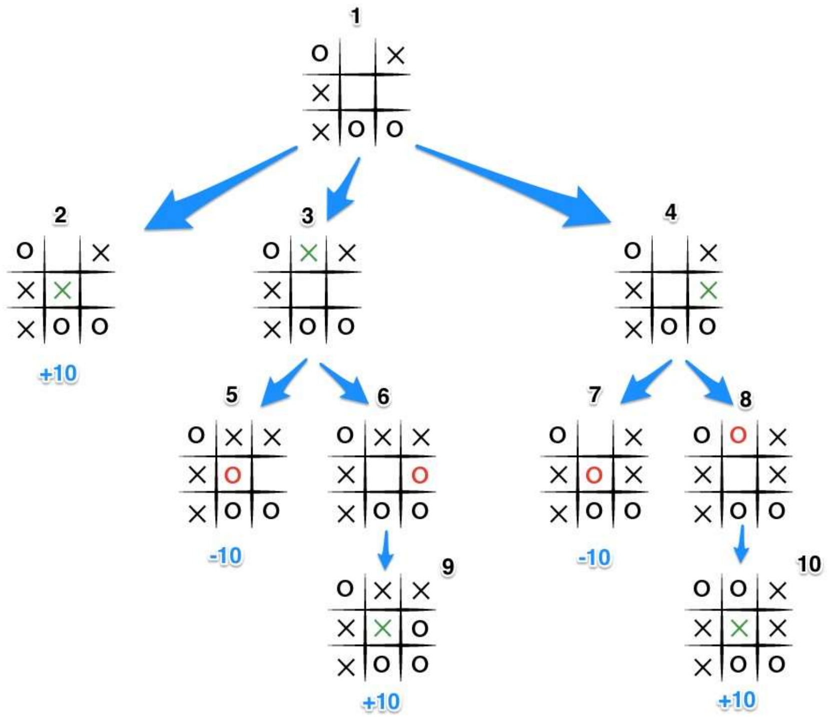


Player X has three possible moves that he can can make. If he chooses the first path then he will win. The other two paths may or may not lead to a win.

Player Xs goal is to maximize the state of the game such that he will win or not lose. So out of the three possible paths he will choose the first. As X wins through the first path, the path is set with a reward of 10 points.But we also need to determine the reward points for the remaining two paths in order for X to make the best possible move. To do this we read these two new states individually and analyze them from O's perpective as it will be Os turn next. Assuming that O is also playing to win this game, but relative to us. Hence now O will be the minimizing player.



O obviously wants to choose the move that results in the worst score for us, it wants to pick a move that would *minimize* our ultimate score. Hence if there is path that leads to O's win and its O's turn the reward for that path will be set to -10. If the path leads to a draw the reward is set to 0. All these reward points have to be passed to the first initial state to enable X to make the best possible move.

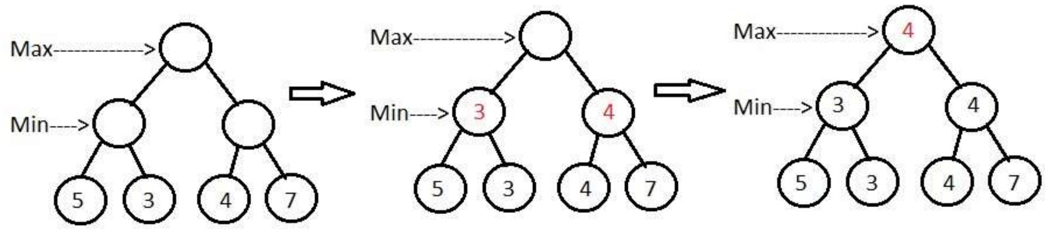


Improving Minimax through alpha beta pruning:

Alpha–beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. It is an adversarial search algorithm used commonly for machine playing of two-player games (Tic-tac-toe, Chess, Connect 4, etc.). It stops evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.
This can be implemented if you are constructing a tree and traversing it to find the best move and not passing the reward points to the initial state.

Algorithm:

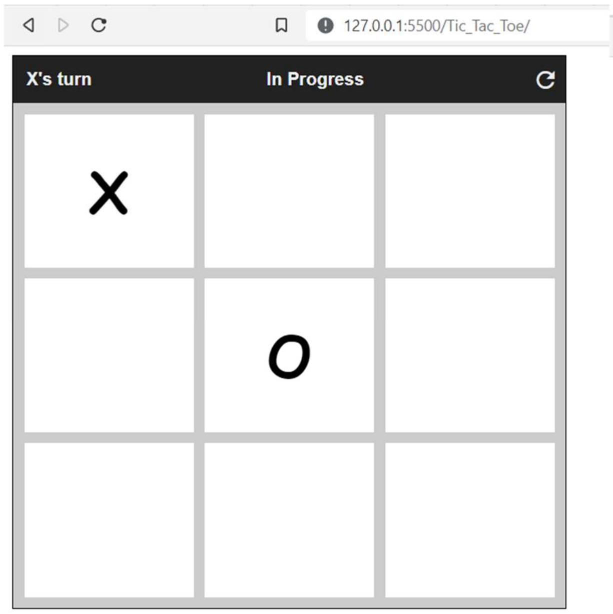
1. Construct the complete game tree
2. Evaluate scores for leaves using the evaluation function
3. Back-up scores from leaves to root, considering the player type: ● For max player, select the child with the maximum score
    ● For min player, select the child with the minimum score
4. At the root node, choose the node with max value and perform the corresponding move

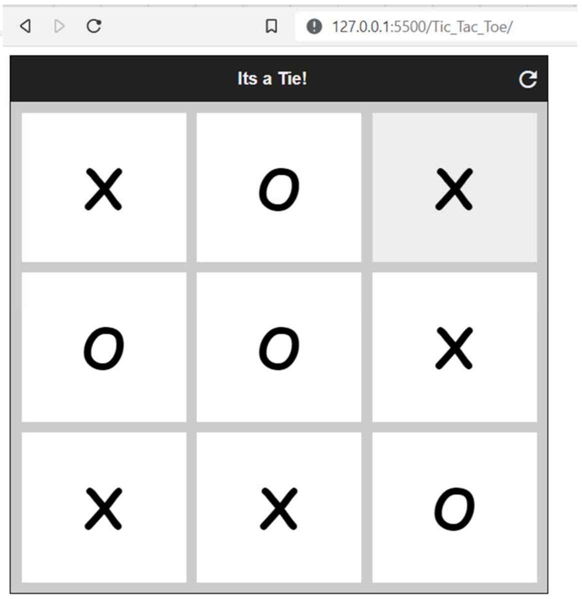


CODE:

```
import Game from "./Game.js";
export default class AI_logic{

    constructor(game)
    {
        //copy instance of game
        this.og_game=Object.assign(Object.create(Object.getPrototype
Of(game)),game);
        this.og_game.board=game.board.slice(0);
        this.og_game.turn=game.turn;

        this.player=this.og_game.turn;//understand who is ai
    }

    makeMove()// return move to make
    {
        let possible_moves=this.find_poss_mov(this.og_game);

        let cpy_game=this.copy_state(this.og_game);
        Array.from(possible_moves.keys()).map(key=>{
            possible_moves.set(key,this.simulate(cpy_game,key));
            cpy_game=this.copy_state(this.og_game);
        });
        console.log(possible_moves);

        let
best_move=this.choose_bestMove(this.player,possible_moves);
        return best_move;
    }

    simulate(game,index)
    {
        let value=0;
        game.makeMove(index);
        if(this.isInProgress(game))
        {
            let map=this.find_poss_mov(game);
            let copy_game=this.copy_state(game);
            Array.from(map.keys()).map(key=>{
                map.set(key,this.simulate(copy_game,key));
                copy_game=this.copy_state(game);
            });
            value=map.get(this.choose_bestMove(game.turn,map));
        }
        else if(this.hasWon(game)){
            if(game.turn==this.player)
                value+=1;
            else
                value-=1;
        }
        return value;
    }

    find_poss_mov(game)
    {
        let map =new Map();
        for(let i=0;i<game.board.length;i++)
        {
            if(game.board[i]==null)
            map.set(i,null);
        }
        return map;
    }

    copy_state(game)
    {
        let copy_state;
        copy_state=Object.assign(Object.create(Object.getPrototypeOf
(game)),game);
        copy_state.board=game.board.slice(0);
        copy_state.turn=game.turn;

        return copy_state;
    }

    choose_bestMove(player,possible_moves)
    {
        let move,best_score;
        if(player==this.player)
            best_score=-Infinity;
        else
            best_score=+Infinity;

        Array.from(possible_moves.keys()).map(key=>
        {
            if(player==this.player &&
possible_moves.get(key)>best_score)
            {
                move=key;
                best_score=possible_moves.get(key);
            }
            else if(player!=this.player &&
possible_moves.get(key)<best_score)
            {
                move=key;
                best_score=possible_moves.get(key);
            }
        });
        return move;
    }

    isInProgress(game)//for tie and in progress
    {
        return !this.hasWon(game) && game.board.includes(null);
    }

    hasWon(game)
    {
        const winningCombos=[
            [0,1,2],
            [3,4,5],
            [6,7,8],
            [0,3,6],
            [1,4,7],
            [2,5,8],
            [0,4,8],
            [2,4,6]
        ];

        for (const comb of winningCombos) {
            const [a,b,c]=comb;

            if(game.board[a] && (game.board[a]===game.board[b] &&
game.board[b]===game.board[c]))
            {
                return comb;
                }
            }
            return null;
        }


}
```

OUTPUT:





Post Lab Assignment:

1. Why is it called minimax algorithm?
2. How Min-Max algorithm is used in solving real life problem? Explain
3. What is minimax vs maximin?
4. What is maximin example?
5. What are the properties of minimax algorithm?