**Assignment 5**

**Problem Statement:**  
Implement the **Minimax Algorithm** for game playing.

**Theory**

**1. Game Playing in AI**

* **Game playing** is one of the classical problems of Artificial Intelligence.
* Games like **Tic-Tac-Toe, Chess, Checkers** can be modeled as **search problems**.
* Players take turns making moves, and the goal is to maximize one’s winning chances while minimizing the opponent’s.

**2. Minimax Algorithm**

* The **Minimax algorithm** is a decision-making algorithm for two-player games.
* One player is the **Maximizer (AI)**, who tries to maximize the score.
* The other is the **Minimizer (Human)**, who tries to minimize the score.
* The algorithm explores all possible moves using recursion and backtracking.

**Evaluation Function (for Tic-Tac-Toe):**

* +10 → AI (O) wins
* -10 → Human (X) wins
* 0 → Draw or undecided

**Key Concepts:**

* **Terminal states:** Win, lose, or draw.
* **Depth:** Number of moves explored; used to choose faster wins or slower losses.
* **Optimal Strategy:** AI always chooses the best move using minimax.

**Algorithm Steps**

1. Evaluate the current board:
   * Return score if win/loss/draw.
2. If it’s AI’s turn (maximizer):
   * Try all possible moves.
   * Choose the move with maximum score.
3. If it’s Player’s turn (minimizer):
   * Try all possible moves.
   * Choose the move with minimum score.
4. Repeat until the game ends.

**Code (C++ Implementation)**

#include <iostream>

#include <vector>

#include <limits>

using namespace std;

const char PLAYER = 'X'; // Human

const char AI = 'O'; // Computer

// Print the board

void printBoard(vector<vector<char>> &board) {

cout << "\n";

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

cout << (board[i][j] == ' ' ? '.' : board[i][j]) << " ";

}

cout << "\n";

}

cout << "\n";

}

// Check if moves are left

bool movesLeft(vector<vector<char>> &board) {

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

if (board[i][j] == ' ')

return true;

return false;

}

// Evaluate board: +10 if AI wins, -10 if player wins, 0 otherwise

int evaluate(vector<vector<char>> &b) {

// Rows

for (int row = 0; row < 3; row++)

if (b[row][0] == b[row][1] && b[row][1] == b[row][2] && b[row][0] != ' ')

return (b[row][0] == AI) ? 10 : -10;

// Cols

for (int col = 0; col < 3; col++)

if (b[0][col] == b[1][col] && b[1][col] == b[2][col] && b[0][col] != ' ')

return (b[0][col] == AI) ? 10 : -10;

// Diagonals

if (b[0][0] == b[1][1] && b[1][1] == b[2][2] && b[0][0] != ' ')

return (b[0][0] == AI) ? 10 : -10;

if (b[0][2] == b[1][1] && b[1][1] == b[2][0] && b[0][2] != ' ')

return (b[0][2] == AI) ? 10 : -10;

return 0;

}

// Minimax function

int minimax(vector<vector<char>> &board, int depth, bool isMax) {

int score = evaluate(board);

if (score == 10) return score - depth;

if (score == -10) return score + depth;

if (!movesLeft(board)) return 0;

if (isMax) {

int best = numeric\_limits<int>::min();

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

if (board[i][j] == ' ') {

board[i][j] = AI;

best = max(best, minimax(board, depth + 1, false));

board[i][j] = ' ';

}

return best;

} else {

int best = numeric\_limits<int>::max();

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

if (board[i][j] == ' ') {

board[i][j] = PLAYER;

best = min(best, minimax(board, depth + 1, true));

board[i][j] = ' ';

}

return best;

}

}

// Find the best move for AI

pair<int, int> findBestMove(vector<vector<char>> &board) {

int bestVal = numeric\_limits<int>::min();

pair<int, int> bestMove = {-1, -1};

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

if (board[i][j] == ' ') {

board[i][j] = AI;

int moveVal = minimax(board, 0, false);

board[i][j] = ' ';

if (moveVal > bestVal) {

bestMove = {i, j};

bestVal = moveVal;

}

}

return bestMove;

}

// -------------------- MAIN --------------------

int main() {

vector<vector<char>> board(3, vector<char>(3, ' '));

int x, y;

cout << "Tic-Tac-Toe (You = X, Computer = O)\n";

for (int turn = 0; turn < 9; turn++) {

printBoard(board);

if (turn % 2 == 0) {

cout << "Enter your move (row col): ";

cin >> x >> y;

if (x >= 0 && x < 3 && y >= 0 && y < 3 && board[x][y] == ' ')

board[x][y] = PLAYER;

else {

cout << "Invalid move! Try again.\n";

turn--;

}

} else {

cout << "Computer's move...\n";

pair<int, int> bestMove = findBestMove(board);

board[bestMove.first][bestMove.second] = AI;

}

int score = evaluate(board);

if (score == 10) {

printBoard(board);

cout << "Computer Wins!\n";

break;

}

if (score == -10) {

printBoard(board);

cout << "You Win!\n";

break;

}

if (!movesLeft(board)) {

printBoard(board);

cout << "It's a Draw!\n";

break;

}

}

return 0;

}

**Sample Output**

Tic-Tac-Toe (You = X, Computer = O)

. . .

. . .

. . .

Enter your move (row col): 0 0

X . .

. . .

. . .

Computer's move...

X . .

. O .

. . .

Enter your move (row col): 2 0

X . .

. O .

X . .

Computer's move...

X . .

O O .

X . .

Enter your move (row col): 1 2

X . .

O O X

X . .

Computer's move...

X O .

O O X

X . .

Enter your move (row col): 2 1

X O .

O O X

X X .

Computer's move...

X O .

O O X

X X O

Enter your move (row col): 0 2

X O X

O O X

X X O

It's a Draw!

**Conclusion**

* The **Minimax algorithm** was successfully implemented for Tic-Tac-Toe.
* The computer plays optimally and never loses.
* Outcomes are: **AI win or Draw**; human cannot force a win if AI plays correctly.
* This demonstrates how **adversarial search** is applied in game playing.