VISHWAKARMA INSTITUTE OF TECHNOLOGY

COMPUTER ENGINEERING

Name: Abhinav Mahajan

Division: TY-C

Roll No: 15

Subject: Artificial Intelligence (AI)

LAB ASSIGNMENT NO – 1

Implementation of AI and Non-AI techniques by implementing Tic-Tac-Toe Game.

1. Non-AI Technique:

Description:

A 2D array with 3 rows and 3 columns, is used to represent the board game tic tac toe. Each grid cell has three possible states: empty, player sign (either 'X' of 'O') occupied, or invalid if the cell does not fit within the 3x3 grid.

Code:

```
#include<bits/stdc++.h>
using namespace std;

int main(){
    char matrix[3][3];
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            matrix[i][j] = '*';
        }
    }
    while (true) {
        int x, y;
        while (true) {
            int flag = 0;
            cout << "Player One" << endl;
```

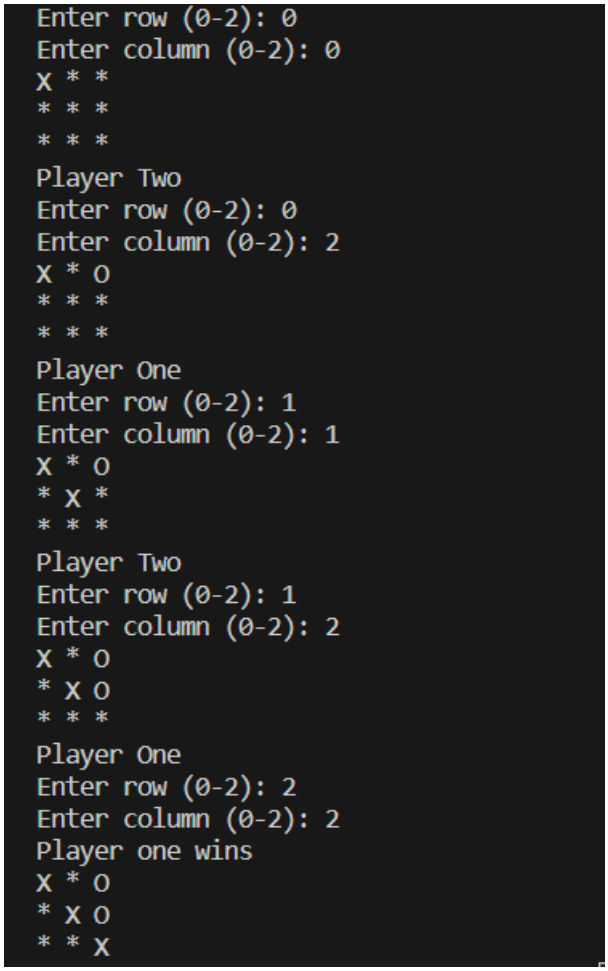
```
            cout << "Enter row (0-2): ";
            cin >> x;
            cout << "Enter column (0-2): ";
            cin >> y;
            if (matrix[x][y] == '*' && x <= 2 && y <= 2)
            {
                matrix[x][y] = 'X';
                flag = 1;
            }
            if (flag == 1)
            {
                break;
            }
            else
            {
                cout << "Error Enter Valid coordinates" << endl;
            }
        }
        int countx = 0;
        int county = 0;
        int countd = 0;
        for (int i = 0; i < 3; i++) {
            if (matrix[x][i] == 'X') {
                countx++;
            }
        }
        for (int i = 0; i < 3; i++) {
            if (matrix[i][y] == 'X') {
                county++;
            }
        }
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (i == j && matrix[i][j] == 'X')
                {
                    countd++;
                }
            }
        }
        if (matrix[0][2] == 'X' && matrix[1][1] == 'X' && matrix[2][0] ==
'X') {
            cout << "Player one wins" << endl;
            break;
        }
        if (countx == 3 || county == 3 || countd == 3) {
            cout << "Player one wins" << endl;
            break;
        }
```

```
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        cout << matrix[i][j] << ' ';
    }
    cout << endl;
}
int x1, y1;
while (true) {
    int flag = 0;
    cout << "Player Two " << endl;
    cout << "Enter row (0-2): ";
    cin >> x1;
    cout << "Enter column (0-2): ";
    cin >> y1;
    if (matrix[x1][y1] == '*' && x1 <= 2 && y1 <= 2)
    {
        matrix[x1][y1] = 'O';
        flag = 1;
    }
    if (flag == 1)
    {
        break;
    }
    else
    {
        cout << "Error Enter Valid coordinates" << endl;
    }
}
countx = 0;
county = 0;
countd = 0;

for (int i = 0; i < 3; i++)
{
    if (matrix[x1][i] == 'O')
    {
        countx++;
    }
}
for (int i = 0; i < 3; i++)
{
    if (matrix[i][y1] == 'O')
    {
        county++;
    }
}
for (int i = 0; i < 3; i++)
{
```

```
            for (int j = 0; j < 3; j++)
            {
                if (i == j && matrix[i][j] == 'O')
                {
                    countd++;
                }
            }
        }
        if (countx == 3 || county == 3 || countd == 3)
        {
            cout << "Player two wins" << endl;
            break;
        }
        if (matrix[0][2] == 'O' && matrix[1][1] == 'O' && matrix[2][0] ==
'O')
        {
            cout << "Player two wins" << endl;
            break;
        }
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                cout << matrix[i][j] << ' ';
            }
            cout << endl;
        }
    }
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            cout << matrix[i][j] << ' ';
        }
        cout << endl;
    }
    return 0;
}
```

Output:



2. AI Technique:

Description:

The grid used here is 3x3, as previously defined.

After a terminal state has been reached, the algorithm assigns a rating to each terminal state based on the outcomes. For instance, if player 1 wins, the score is +1; if player 2, the score is -1; and if the game is tied, the score is 0.

Once it reaches a terminal state, the algorithm propagates the outcomes back up the recursive chain. For each level of the recursive call, the highest possible score is chosen for player 1's turn and the lowest possible score is chosen for player 2's turn. Further iterations of this process lead to the original call, which is the best action for the current player.

Code:

```
#include<bits/stdc++.h>
```

```
using namespace std;

int BOARD_SIZE = 3;
char PLAYER_X = 'X';
char PLAYER_O = 'O';
char EMPTY_CELL = ' ';

void printBoard(vector<vector<char>>& board){
    for(int i = 0; i < 3; i++){
        for(int j = 0; j < 3; j++){
            cout<<board[i][j]<<" ";
        }
        cout<<endl;
    }
}

bool checkWin(vector<vector<char>>& board, char player) {
    for (int i = 0; i < BOARD_SIZE; i++) {
        if (board[i][0] == player && board[i][1] == player && board[i][2]
== player)
            return true;
        if (board[0][i] == player && board[1][i] == player && board[2][i]
== player)
            return true;
    }

    if (board[0][0] == player && board[1][1] == player && board[2][2] ==
player)
        return true;
    if (board[0][2] == player && board[1][1] == player && board[2][0] ==
player)
        return true;

    return false;
}

bool isBoardFull(vector<vector<char>>& board) {
    for (int i = 0; i < BOARD_SIZE; i++) {
        for (int j = 0; j < BOARD_SIZE; j++) {
            if (board[i][j] == EMPTY_CELL)
                return false;
        }
    }
    return true;
}

int evaluateBoard(vector<vector<char>>& board) {
    if (checkWin(board, PLAYER_X))
```

```
        return 10;
    if (checkWin(board, PLAYER_O))
        return -10;
    return 0;
}

int minimax(vector<vector<char>>& board, int depth, bool isMaximizer) {
    int score = evaluateBoard(board);

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

    if (isBoardFull(board))
        return 0;

    if (isMaximizer) {
        int bestScore = -1000;
        for (int i = 0; i < BOARD_SIZE; i++) {
            for (int j = 0; j < BOARD_SIZE; j++) {
                if (board[i][j] == EMPTY_CELL) {
                    board[i][j] = PLAYER_X;
                    bestScore = max(bestScore, minimax(board, depth + 1,
!isMaximizer));
                    board[i][j] = EMPTY_CELL;
                }
            }
        }
        return bestScore;
    } else {
        int bestScore = 1000;
        for (int i = 0; i < BOARD_SIZE; i++) {
            for (int j = 0; j < BOARD_SIZE; j++) {
                if (board[i][j] == EMPTY_CELL) {
                    board[i][j] = PLAYER_O;
                    bestScore = min(bestScore, minimax(board, depth + 1,
!isMaximizer));
                    board[i][j] = EMPTY_CELL;
                }
            }
        }
        return bestScore;
    }
}

pair<int, int> findBestMove(vector<vector<char>>& board) {
    int bestScore = -1000;
    pair<int, int> bestMove = {-1, -1};
```

```
    for (int i = 0; i < BOARD_SIZE; i++) {
        for (int j = 0; j < BOARD_SIZE; j++) {
            if (board[i][j] == EMPTY_CELL) {
                board[i][j] = PLAYER_X;
                int moveScore = minimax(board, 0, false);
                board[i][j] = EMPTY_CELL;

                if (moveScore > bestScore) {
                    bestScore = moveScore;
                    bestMove = {i, j};
                }
            }
        }
    }

    return bestMove;
}

int main() {
    vector<vector<char>> board(BOARD_SIZE, vector<char>(BOARD_SIZE,
EMPTY_CELL));

    bool isPlayerXTurn = true;

    while (!checkWin(board, PLAYER_X) && !checkWin(board, PLAYER_O) &&
!isBoardFull(board)) {
        printBoard(board);

        if (isPlayerXTurn) {
            int row, col;
            cout << "Enter row (0-2): ";
            cin >> row;
            cout << "Enter column (0-2): ";
            cin >> col;

            if (row >= 0 && row < BOARD_SIZE && col >= 0 && col <
BOARD_SIZE && board[row][col] == EMPTY_CELL) {
                board[row][col] = PLAYER_O;
                isPlayerXTurn = false;
            } else {
                cout << "Invalid move. Try again!" << endl;
            }
        } else {
            cout << "AI is making a move..." << endl;
            pair<int, int> aiMove = findBestMove(board);
            board[aiMove.first][aiMove.second] = PLAYER_X;
            isPlayerXTurn = true;
        }
```

```
    }

    printBoard(board);

    if (checkWin(board, PLAYER_X))
        cout << "AI wins!" << endl;
    else if (checkWin(board, PLAYER_O))
        cout << "You win!" << endl;
    else
        cout << "It's a draw!" << endl;

    return 0;
}
```

Output: