Bansilal Ramnath Agarwal Charitable Trust's

Vishwakarma Institute of Technology Pune-37

(Anautonomous Institute of Savitribai Phule Pune University)

Department of Computer Engineering

Division	C
Batch	2
GR-no	12111180
Rollno	17
Name	Divya Mahajan

Title: Implementation of AI and Non-AI Tic Tac Toe.

Description:

Terminal conditions:

- Player1 wins
- Player2 wins
- Tie
- Board is full

Implementation of Tic Tac Toe without AI:

A 3x3 grid is commonly used to depict the board game tic tac toe. Each grid cell has three possible states: empty, player sign (often an 'X' for player 1 and an 'O' for player 2) occupied, or invalid if the cell does not fit within the 3x3 grid. A 2D array is used to represent the game state.

The Tic Tac Toe minimax algorithm is described as follows (With AI):

A 3x3 grid, same as defined above, is used here.

The algorithm rates each terminal state after it has been reached depending on the results. For instance, the score is +1 if player 1 wins, -1 if player 2 wins, and 0 if the game is a tie.

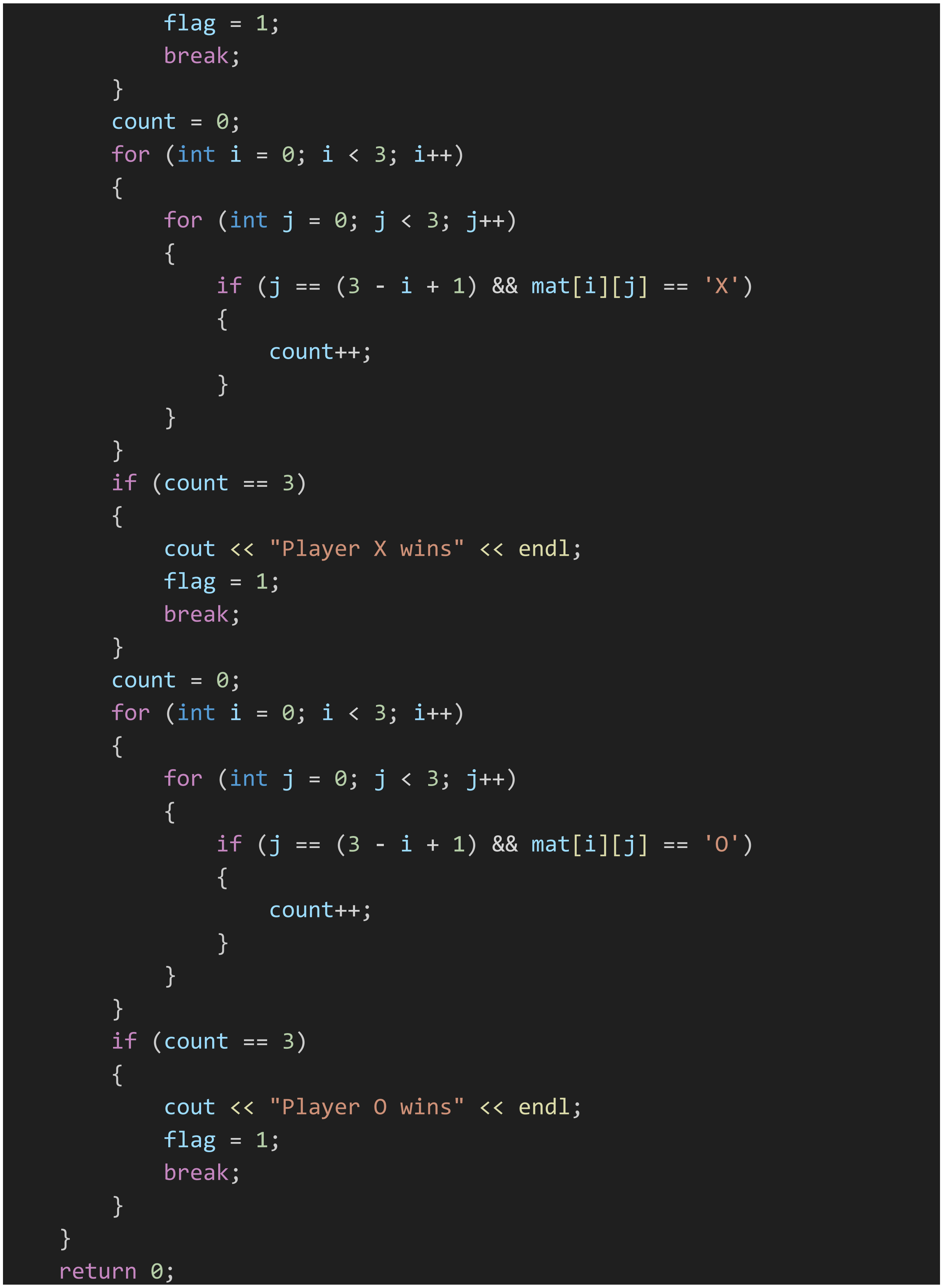
The algorithm propagates the results back up the recursive chain once it reaches a terminal state. It selects the highest possible score for player 1's turn and the lowest possible score for player 2's turn for each level of the recursive call. The original call, the best move for the current player, is reached after further iterations of this process.

Code:

- Non-AI

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

```

```cpp
bool isBoardFull(vector<vector<char>> &board)
{
    for (auto &row : board)
    {
        for (char cell : row)
        {
            if (cell == '_')
                return false;
        }
    }
    return true;
}

int main()
{
    vector<vector<char>> mat(3, vector<char>(3, '_'));
    char player = 'X';
    int flag = 0;
    while (flag == 0)
    {
        int row;
        int col;
        cout << "Player " << player << "'s turn. Enter row and column
(0-2): ";
        cin >> row >> col;
        if (row >= 0 && row < 3 && col >= 0 && col < 3 &&
mat[row][col] == '_')
        {
            mat[row][col] = player;
            if (player == 'X')
                player = 'O';
            else
                player = 'X';
        }
        else
        {
            cout << "Invalid move. Try again!" << endl;
            continue;
        }
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                cout << mat[i][j] << " ";
            }
            cout << endl;
        }

        if (isBoardFull(mat))
        {
            cout << "It's a draw!" << endl;
            break;
        }
        for (int i = 0; i < 3; i++)
        {
            flag = 0;
            if (mat[i][0] == 'X' && mat[i][1] == 'X' && mat[i][2] ==
'X')
            {
                cout << "Player X wins" << endl;
                flag = 1;
                break;
            }
            if (mat[i][0] == 'O' && mat[i][1] == 'O' && mat[i][2] ==
'O')
            {
                cout << "Player O wins" << endl;
                flag = 1;
                break;
            }
            if (mat[0][i] == 'X' && mat[1][i] == 'X' && mat[2][i] ==
'X')
            {
                cout << "Player X wins" << endl;
                flag = 1;
                break;
            }
            if (mat[0][i] == 'O' && mat[1][i] == 'O' && mat[2][i] ==
'O')
            {
                cout << "Player O wins" << endl;
                flag = 1;
                break;
            }
        }
        int count = 0;
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                if (i == j && mat[i][j] == 'X')
                {
                    count++;
                }
            }
        }
        if (count == 3)
        {
            cout << "Player X wins" << endl;
            flag = 1;
            break;
        }
        count = 0;
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                if (i == j && mat[i][j] == 'O')
                {
                    count++;
                }
            }
        }
        if (count == 3)
        {
            cout << "Player O wins" << endl;
```



```
            flag = 1;
            break;
        }
        count = 0;
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                if (j == (3 - i + 1) && mat[i][j] == 'X')
                {
                    count++;
                }
            }
        }
        if (count == 3)
        {
            cout << "Player X wins" << endl;
            flag = 1;
            break;
        }
        count = 0;
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                if (j == (3 - i + 1) && mat[i][j] == 'O')
                {
                    count++;
                }
            }
        }
        if (count == 3)
        {
            cout << "Player O wins" << endl;
            flag = 1;
            break;
        }
    }
    return 0;
```

```
}
```

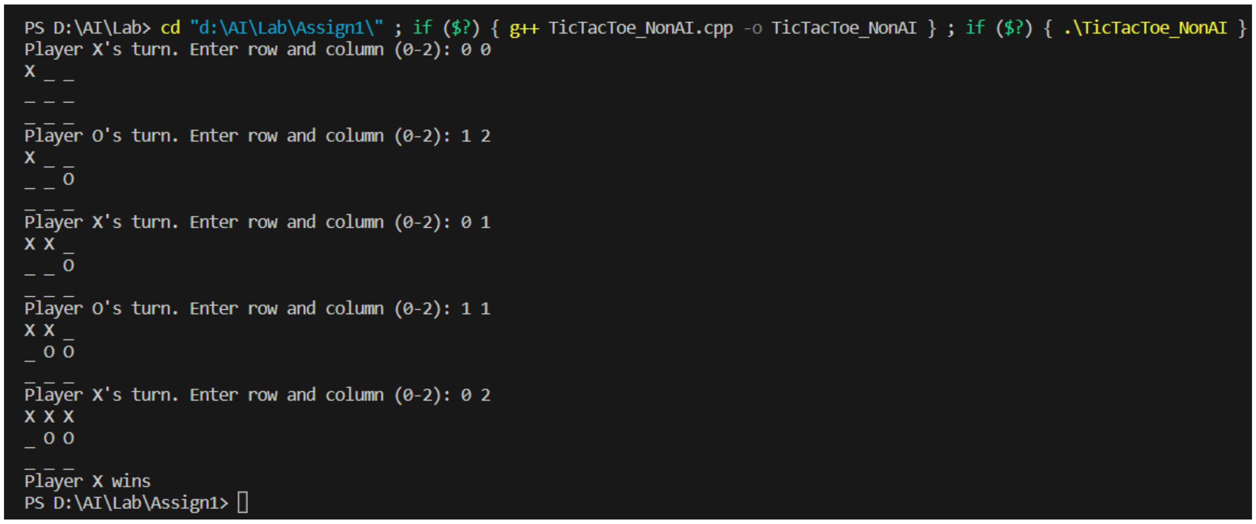
- AI

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

struct Move
{
    int row;
    int col;
};

bool isBoardFull(const vector<vector<char>> &board)
{
    for (const auto &row : board)
    {
        for (const char cell : row)
        {
            if (cell == '_')
                return false;
        }
    }
    return true;
}
char checkWinner(const vector<vector<char>> &board)
{

    for (int i = 0; i < 3; ++i)
    {
        if (board[i][0] != '_' && board[i][0] == board[i][1] &&
board[i][1] == board[i][2])
            return board[i][0];
    }
    for (int i = 0; i < 3; ++i)
    {
        if (board[0][i] != '_' && board[0][i] == board[1][i] &&
board[1][i] == board[2][i])
```

```
            return board[0][i];
    }
    if (board[0][0] != '_' && board[0][0] == board[1][1] &&
board[1][1] == board[2][2])
        return board[0][0];
    if (board[0][2] != '_' && board[0][2] == board[1][1] &&
board[1][1] == board[2][0])
        return board[0][2];
    return '_';
}

int evaluateBoard(const vector<vector<char>> &board)
{
    char winner = checkWinner(board);
    if (winner == 'X')
        return 1;
    else if (winner == 'O')
        return -1;
    else
        return 0;
}

Move findBestMove(vector<vector<char>> &board, char currentPlayer)
{
    int bestScore = currentPlayer == 'X' ? -1000 : 1000;
    Move bestMove;
    Move result;
    if (isBoardFull(board) || checkWinner(board) != '_')
    {
        int score = evaluateBoard(board);
        result.row = score;
        result.col = bestMove.col;
        return result;
    }
    for (int row = 0; row < 3; ++row)
    {
        for (int col = 0; col < 3; ++col)
        {
            if (board[row][col] == '_')
            {
                board[row][col] = currentPlayer;
                if (currentPlayer == 'X')
                {
                    int score = findBestMove(board, 'O').row;
                    if (score > bestScore)
                    {
                        bestScore = score;
                        bestMove.row = row;
                        bestMove.col = col;
                    }
                }
                else
                {
                    int score = findBestMove(board, 'X').row;
                    if (score < bestScore)
                    {
                        bestScore = score;
                        bestMove.row = row;
                        bestMove.col = col;
                    }
                }
                board[row][col] = '_';
            }
        }
    }
    result.row = bestScore;
    result.col = bestMove.col;
    return result;
}

void printBoard(const vector<vector<char>> &board)
{
    for (const auto &row : board)
    {
        for (const char cell : row)
        {
            cout << cell << " ";
        }
        cout << endl;
    }
}

int main()
{
    vector<vector<char>> board(3, vector<char>(3, '_'));
    char currentPlayer = 'X';
    while (true)
    {
        cout << endl;
        printBoard(board);
        if (checkWinner(board) != '_')
        {
            char winner = checkWinner(board);
            if (winner == 'X')
                cout << "Player X wins!" << endl;
            else
                cout << "Player O wins!" << endl;
            break;
        }
        else if (isBoardFull(board))
        {
            cout << "It's a draw!" << endl;
            break;
        }
        if (currentPlayer == 'X')
        {
            cout << "Player X's turn. Enter row and column (0-2): ";
            int row, col;
            cin >> row >> col;
            if (row >= 0 && row < 3 && col >= 0 && col < 3 &&
board[row][col] == '_')
            {
                board[row][col] = 'X';
                currentPlayer = 'O';
            }
            else
            {
                cout << "Invalid move. Try again!" << endl;
            }
        }
        else
        {
            cout << "Player O's turn" << endl;
            Move bestMove = findBestMove(board, 'O');
            board[bestMove.row][bestMove.col] = 'O';
            currentPlayer = 'X';
        }
    }
    cout << endl;
    printBoard(board);
    return 0;
}
```

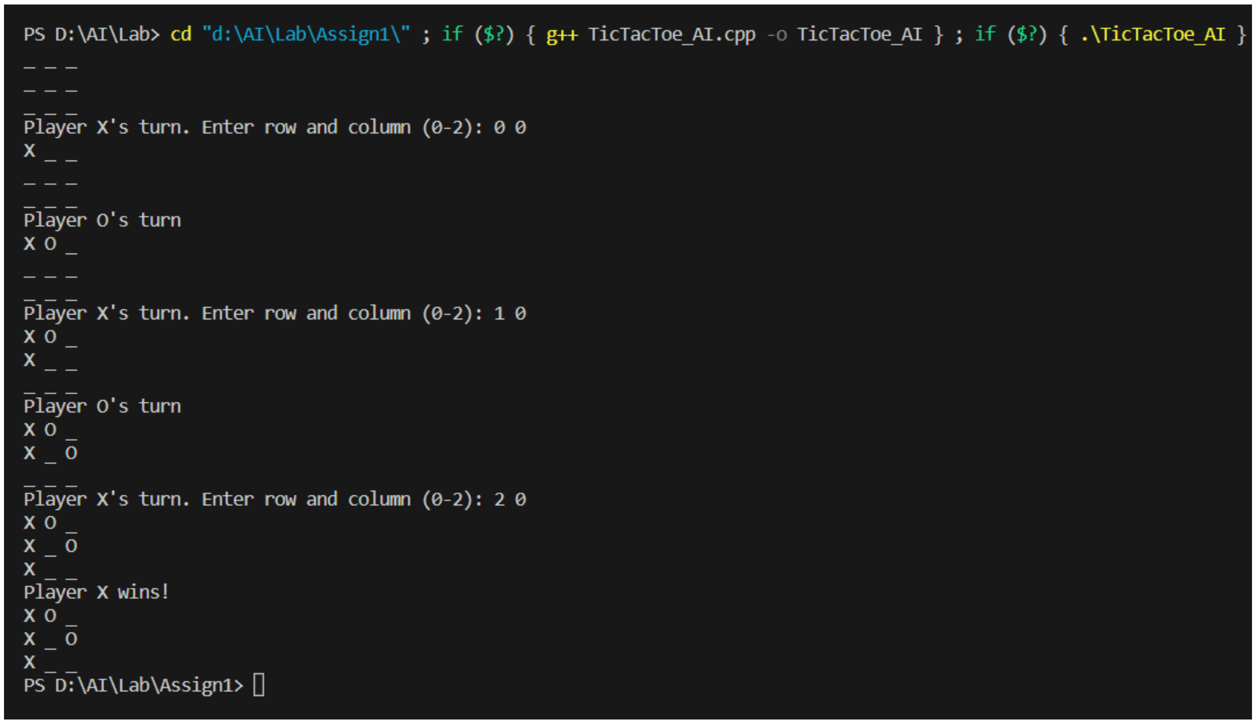
Screenshots/Output:

Non-AI:



```
PS D:\AI\Lab> cd "d:\AI\Lab\Assign1\" ; if ($?) { g++ TicTacToe_NonAI.cpp -o TicTacToe_NonAI } ; if ($?) { .\TicTacToe_NonAI }
Player X's turn. Enter row and column (0-2): 0 0
X _ _
_ _ _
_ _ _
Player O's turn. Enter row and column (0-2): 1 2
X _ _
_ _ O
_ _ _
Player X's turn. Enter row and column (0-2): 0 1
X X _
_ _ O
_ _ _
Player O's turn. Enter row and column (0-2): 1 1
X X _
_ O O
_ _ _
Player X's turn. Enter row and column (0-2): 0 2
X X X
_ O O
_ _ _
Player X wins
PS D:\AI\Lab\Assign1> 
```

AI:



```
PS D:\AI\Lab> cd "d:\AI\Lab\Assign1\" ; if ($?) { g++ TicTacToe_AI.cpp -o TicTacToe_AI } ; if ($?) { .\TicTacToe_AI }
_ _ _
_ _ _
_ _ _
Player X's turn. Enter row and column (0-2): 0 0
X _ _
_ _ _
_ _ _
Player O's turn
X O _
_ _ _
_ _ _
Player X's turn. Enter row and column (0-2): 1 0
X O _
X _ _
_ _ _
Player O's turn
X O _
X _ O
_ _ _
Player X's turn. Enter row and column (0-2): 2 0
X O _
X _ O
X _ _
Player X wins!
X O _
X _ O
X _ _
PS D:\AI\Lab\Assign1> 
```