Vishwakarma Institute of Technology
Department of Computer Science Engineering (CS)

Name	Vaishnavi Vijay Arthamwar
Division	CS-A
Batch	B3
Roll No.	16
PRN No.	12220242
Department	Computer
Subject	Artificial Intelligence
Assignment No.	1

Assingment No. 1

Title: Implementation of AI and Non-AI technique by implementing any two player game

Non-AI:

Approach 1:

Data Structure BOARD: The Tic-Tac-Toe game consists of a nine-element vector called BOARD; it represents 1 to 9 in 3 rows. we use 2 for a blank, 3 for an X and 5 for an O.

Data Structure TURN: An integer variable called TURN indicates 1 for the first move and 9 for the last.

The algorithm consists of three sub procedures: –

MAKE2: Returns 5 if the center square is blank(i.e., if BOARD[5]=2, then return 5); otherwise, it returns any blank non corner square, i.e., 2, 4, 6 or 8.

POSSWIN(p): Returns 0 if player p cannot win on the next move and otherwise returns the number of the square that gives a winning move. It operates by checking, one at a time, each of the rows, columns and diagonals. It checks each line using products of values of its squares together. A product 3*3*2 = 18 gives a win for X, 5*5*2=50 gives a win for O, and the winning move is the holder of the blank.

GO(n): Makes a move to square n. It sets BOARD[n] to 3 if TURN is odd or 5 if TURN is even. It also increments TURN by 1.

Code:

```
#include <iostream>
#include <ctime>
#include <random>
using namespace std;

int board[9] = {2, 2, 2, 2, 2, 2, 2, 2, 2};
void show_board()
{

    cout << "    "
         << "     |    "
         << "     |    " << endl;
    cout << "    " << board[0] << "    |    " << board[1] << "    |    " <<
board[2] << endl;
    cout << "-------------------------" << endl;
    cout << "    "
         << "     |    "
```

```
         << "    |   " << endl;
    cout << "   " << board[3] << "   |   " << board[4] << "   |   " <<
board[5] << endl;
    cout << "------------------------" << endl;
    cout << "   "
         << "    |   "
         << "    |   " << endl;
    cout << "   " << board[6] << "   |   " << board[7] << "   |   " <<
board[8] << endl;
    cout << endl;
}
void Go(int index, int turn)
{
    board[index] = (turn % 2 != 0) ? 3 : 5;
}

int posswin(int player)
{
    if (player == 3)
    {
        for (int i = 0; i < 9; i += 3)
        {
            if (board[i] * board[i + 1] * board[i + 2] == 18)
            {
                if (board[i] == 2)
                    return i;
                if (board[i + 1] == 2)
                    return i + 1;
                if (board[i + 2] == 2)
                    return i + 2;
            }
        }

        for (int i = 0; i < 3; i++)
        {
            if (board[i] * board[i + 3] * board[i + 6] == 18)
            {
                if (board[i] == 2)
                    return i;
                if (board[i + 3] == 2)
                    return i + 3;
                if (board[i + 6] == 2)
                    return i + 6;
            }
        }

        if (board[0] * board[4] * board[8] == 18)
        {
```

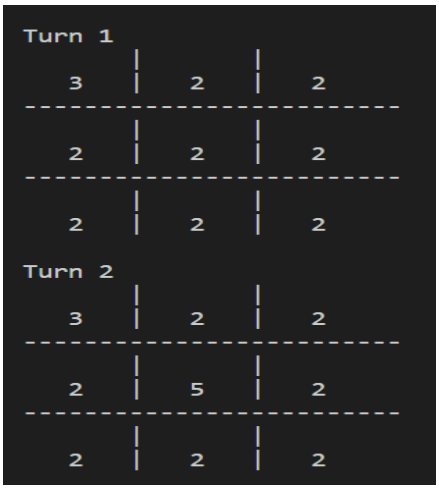
```
            if (board[0] == 2)
                return 0;
            if (board[4] == 2)
                return 4;
            if (board[8] == 2)
                return 8;
        }

        if (board[2] * board[4] * board[6] == 18)
        {
            if (board[2] == 2)
                return 2;
            if (board[4] == 2)
                return 4;
            if (board[6] == 2)
                return 6;
        }
    }
    if (player == 5)
    {
        for (int i = 0; i < 9; i += 3)
        {
            if (board[i] * board[i + 1] * board[i + 2] == 50)
            {
                if (board[i] == 2)
                    return i;
                if (board[i + 1] == 2)
                    return i + 1;
                if (board[i + 2] == 2)
                    return i + 2;
            }
        }

        for (int i = 0; i < 3; i++)
        {
            if (board[i] * board[i + 3] * board[i + 6] == 50)
            {
                if (board[i] == 2)
                    return i;
                if (board[i + 3] == 2)
                    return i + 3;
                if (board[i + 6] == 2)
                    return i + 6;
            }
        }

        if (board[0] * board[4] * board[8] == 50)
        {
```

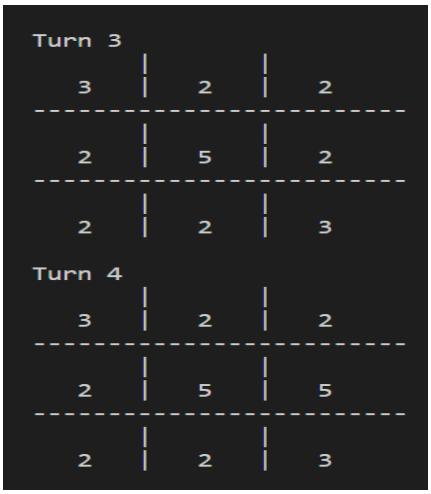
```
            if (board[0] == 2)
                return 0;
            if (board[4] == 2)
                return 4;
            if (board[8] == 2)
                return 8;
        }

        if (board[2] * board[4] * board[6] == 50)
        {
            if (board[2] == 2)
                return 2;
            if (board[4] == 2)
                return 4;
            if (board[6] == 2)
                return 6;
        }
    }

    return -1;
}
int Make2()
{
    if (board[5] == 2)
    {
        return 5;
    }
    else
    {
        int options[] = {2, 4, 6, 8};
        int index = rand() % 4; // Generates a random index (0 to 3)
        return options[index];
    }
}
bool check_winner(int player)
{
    for (int i = 0; i < 9; i += 3)
    {
        if (board[i] * board[i + 1] * board[i + 2] == player * player *
player)
            return true;
    }

    for (int i = 0; i < 3; i++)
    {
        if (board[i] * board[i + 3] * board[i + 6] == player * player *
player)
            return true;
```

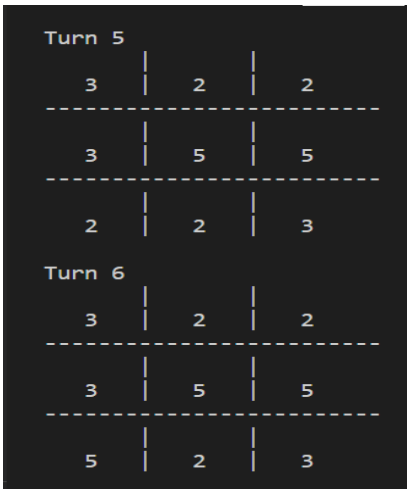
```
    }

    if (board[0] * board[4] * board[8] == player * player * player)
        return true;

    if (board[2] * board[4] * board[6] == player * player * player)
        return true;

    return false;
}

int main()
{
    show_board();
    int turn = 1;
    while (turn <= 9)
    {
        if (turn == 1)
        {
            cout << "Turn " << turn << endl;
            Go(0, turn);
        }
        else if (turn == 2)
        {
            cout << "Turn " << turn << endl;
            Go((board[4] == 2) ? 4 : 1, turn);
        }
        else if (turn == 3)
        {
            cout << "Turn " << turn << endl;
            Go((board[8] == 2) ? 8 : 2, turn);
        }
        else if (turn == 4)
        {
            cout << "Turn " << turn << endl;
            Go((posswin(3) != -1) ? posswin(3) : Make2(), turn);
        }
        else if (turn == 5)
        {
            cout << "Turn " << turn << endl;
            Go((posswin(3) != -1) ? posswin(3) : ((posswin(5) != -1) ?
posswin(5) : (board[7] == 2) ? 7: 3),turn);
        }
        else if (turn == 6)
        {
            cout << "Turn " << turn << endl;
            Go((posswin(5) != -1) ? posswin(5) : ((posswin(3) != -1) ?
posswin(3) : Make2()), turn);
```

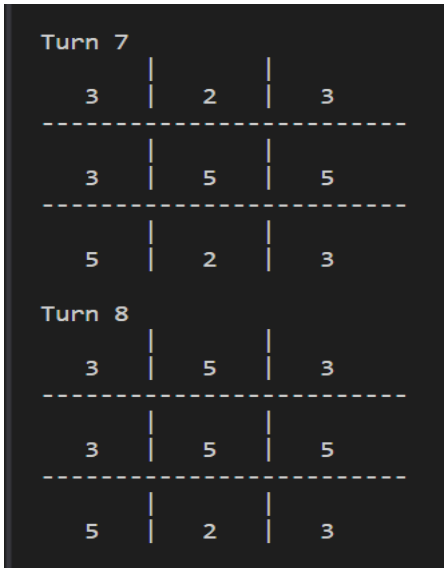
```
        }
        else if (turn == 7 || turn == 9)
        {
            cout << "Turn " << turn << endl;
            Go((posswin(3) != -1) ? posswin(3) : ((posswin(5) != -1) ?
posswin(5) : Make2()), turn);
        }
        else if (turn == 8)
        {
            cout << "Turn " << turn << endl;
            Go((posswin(5) != -1) ? posswin(5) : ((posswin(3) != -1) ?
posswin(3) : Make2()), turn);
        }

        show_board();

        if (check_winner(3))
        {
            cout << "Player X wins!" << endl;
            break;
        }
        else if (check_winner(5))
        {
            cout << "Player O wins!" << endl;
            break;
        }
        else if (turn == 9)
        {
            cout << "It's a draw!" << endl;
        }

        turn++;
    }

    return 0;
}
```

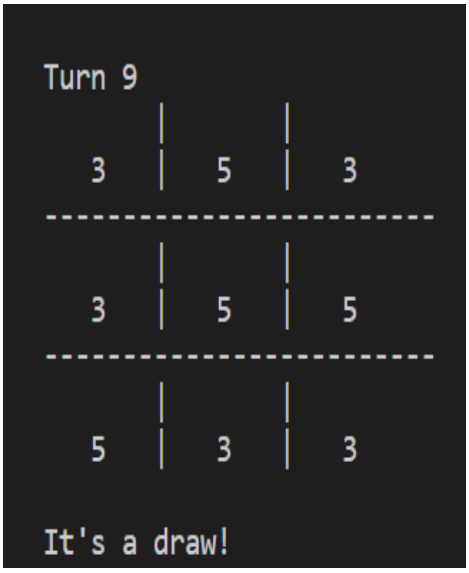
Output:











Approach 2:

Magic Square

Data Structure BOARD: It is identical to Program 2 except for the one change in the board representation. The numbering of the BOARD produces a magic square: all row, columns and diagonals sum up to 15. It simplifies the process of checking of a possible win.

8	**3**	**4**
1	**5**	**9**
6	**7**	**2**

The algorithm:

- In addition to marking the BOARD as moves are made, keep a list for each player, of squares in which he/she has played.
- To check a possible win for one player, consider each pair of squares owned by that player and compute the difference between 15 and sum of the two squares. If the difference is <0 (negative) or > 9, then the original two squares are not collinear and so can be ignored. Otherwise, if the square representing the difference is blank, a move there will produce win - or check if the opponent is winning, block his/her win.
- Since, no player can have more than 4 squares at a time, there will be many fewer squares examined using this scheme than using approach in Approach 1.
- This shows how the choice of representation can have impact on the efficiency on problem-solving program.

Code:

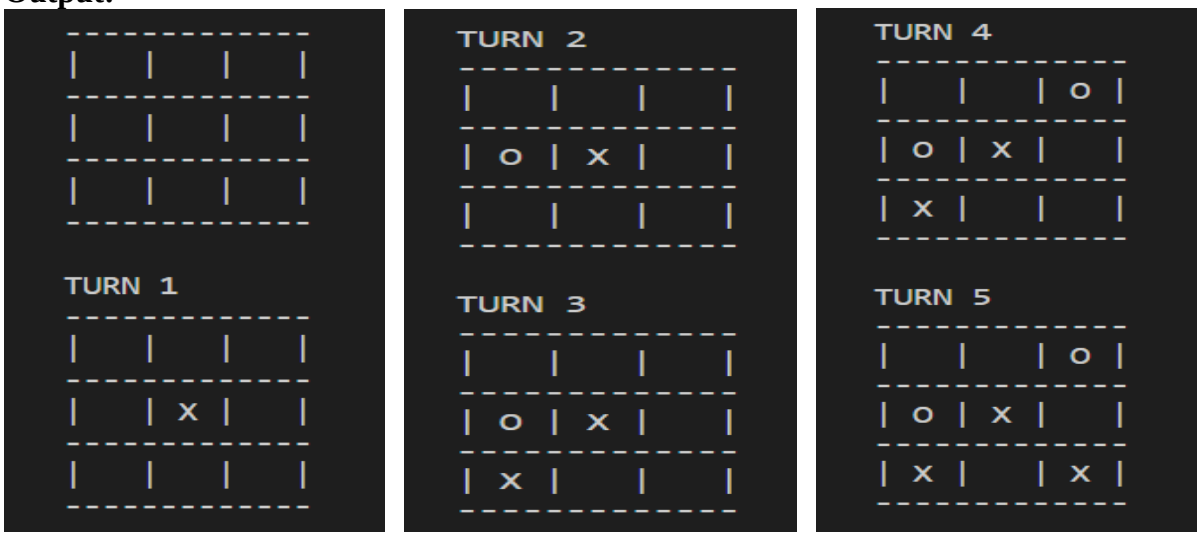
```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>
#include <algorithm>

using namespace std;

vector<int> V1;
vector<int> V2;
int BOARD[] = {8, 3, 4, 1, 5, 9, 6, 7, 2};

int MAKE(); // Function declaration

int POSSWIN(int turnXorO); // Function declaration

void GO(int turn) {
    int value3 = POSSWIN(3);
    int value5 = POSSWIN(5);

    if (turn % 2 == 1) {
        if (!(find(V1.begin(), V1.end(), 5) != V1.end() || find(V2.begin(),
V2.end(), 5) != V2.end())) {
            V1.push_back(5);
        } else if (value3 != -1 && (!(find(V1.begin(), V1.end(), value3) !=
V1.end() || find(V2.begin(), V2.end(), value3) != V2.end()))) {
            V1.push_back(value3);
        } else if (value5 != -1 && (!(find(V1.begin(), V1.end(), value5) !=
V1.end() || find(V2.begin(), V2.end(), value5) != V2.end()))) {
            V1.push_back(value5);
        } else {
            V1.push_back(MAKE());
        }
```

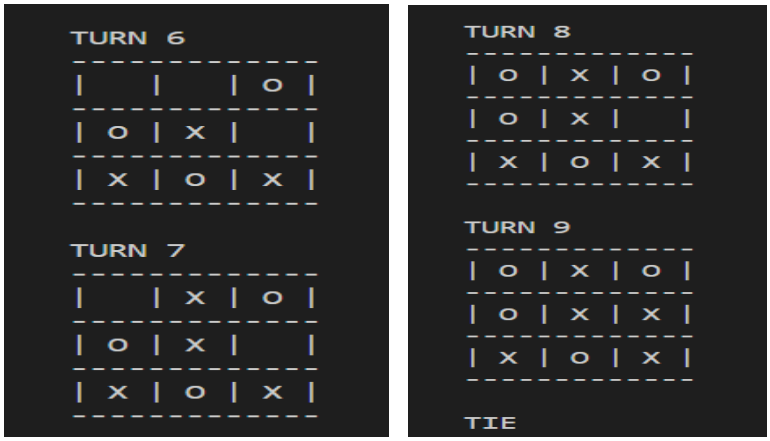
```
    } else {
        if (!(find(V1.begin(), V1.end(), 5) != V1.end() || find(V2.begin(),
V2.end(), 5) != V2.end())) {
            V2.push_back(5);
        } else if (value5 != -1 && (!(find(V1.begin(), V1.end(), value5) !=
V1.end() || find(V2.begin(), V2.end(), value5) != V2.end()))) {
            V2.push_back(value5);
        } else if (value3 != -1 && (!(find(V1.begin(), V1.end(), value3) !=
V1.end() || find(V2.begin(), V2.end(), value3) != V2.end()))) {
            V2.push_back(value3);
        } else {
            V2.push_back(MAKE());
        }
    }
}

int MAKE() {
    if (!(find(V1.begin(), V1.end(), 1) != V1.end() || find(V2.begin(),
V2.end(), 1) != V2.end()))
        return 1;
    else if (!(find(V1.begin(), V1.end(), 6) != V1.end() || find(V2.begin(),
V2.end(), 6) != V2.end()))
        return 6;
    else if (!(find(V1.begin(), V1.end(), 2) != V1.end() || find(V2.begin(),
V2.end(), 2) != V2.end()))
        return 2;
    else if (!(find(V1.begin(), V1.end(), 4) != V1.end() || find(V2.begin(),
V2.end(), 4) != V2.end()))
        return 4;
    else if (!(find(V1.begin(), V1.end(), 7) != V1.end() || find(V2.begin(),
V2.end(), 7) != V2.end()))
        return 7;
    else if (!(find(V1.begin(), V1.end(), 3) != V1.end() || find(V2.begin(),
V2.end(), 3) != V2.end()))
        return 3;
    else if (!(find(V1.begin(), V1.end(), 8) != V1.end() || find(V2.begin(),
V2.end(), 8) != V2.end()))
        return 8;
    else
        return 9;
}

int POSSWIN(int turnXorO) {
    if (turnXorO == 3) {
        for (int i = 0; i < V1.size(); i++) {
            for (int j = i + 1; j < V1.size(); j++) {
                int sum = 15 - (V1[i] + V1[j]);
                if (sum > 0 && sum <= 9) {
```

```
                    return sum;
                }
            }
        }
        return -1;
    } else {
        for (int i = 0; i < V2.size(); i++) {
            for (int j = i + 1; j < V2.size(); j++) {
                int sum = 15 - (V2[i] + V2[j]);
                if (sum > 0 && sum <= 9) {
                    return sum;
                }
            }
        }
        return -1;
    }
}

void WIN(int turn) {
    if (turn % 2 == 1) {
        for (int i = 0; i < V1.size(); i++) {
            for (int j = i + 1; j < V1.size(); j++) {
                for (int k = j + 1; k < V1.size(); k++) {
                    int sum = V1[i] + V1[j] + V1[k];
                    if (sum == 15) {
                        cout << "\n\tWIN X";
                        exit(0);
                    }
                }
            }
        }
    } else {
        for (int i = 0; i < V2.size(); i++) {
            for (int j = i + 1; j < V2.size(); j++) {
                for (int k = j + 1; k < V2.size(); k++) {
                    int sum = V2[i] + V2[j] + V2[k];
                    if (sum == 15) {
                        cout << "\n\tWIN O";
                        exit(0);
                    }
                }
            }
        }
    }
}

void PRINT(string msg) {
    cout << "\n\t" << msg;
```

```
    for (int i = 0; i < 9; i++) {
        if (i % 3 == 0)
            cout << "\n\t-------------\n\t|";

        if (V1.size() > 0 && find(V1.begin(), V1.end(), BOARD[i]) != V1.end())
            cout << " X |";
        else if (V2.size() > 0 && find(V2.begin(), V2.end(), BOARD[i]) !=
V2.end())
            cout << " O |";
        else {
            cout << "   |";
        }
    }
    cout << "\n\t-------------\n";
}

int main() {
    srand(time(NULL)); // Seed for random number generation

    V1.resize(0);
    V2.resize(0);

    for (int i = 0; i < 9; i++) {
        if (i % 3 == 0)
            cout << "\n\t-------------\n\t|";

        cout << "   |";
    }
    cout << "\n\t-------------\n";

    for (int i = 1; i <= 9; i++) {
        GO(i);
        PRINT("TURN " + to_string(i));
        WIN(i);
    }
    cout << "\n\tTIE" << endl;

    return 0;
}
```

Output:





Approach 3:

AI Technique using Minimax Algorithm

Tic-Tac-Toe:

• **Data Structure BOARDPOSITION:** The structure contains a nine-element vector, a list of board positions that could result from the next move and a number representing an estimation of how the board position leads to an ultimate win for the player to move.

• **The algorithm looks** ahead to make a decision on the next move by deciding which the most promising move or the most suitable move at any stage would be, selects the same and assign the rating of the best move to the current position.

• To decide which of a set of BOARD positions is best, do the following for each of them:

– See if it is a win, If so, call the best by giving it the highest possible rating.

– Otherwise, consider all the moves the opponent could make next. See which of them is the worst for us(by recursively calling this procedure). Assume the opponent will make that move. Whatever rating that move has , assign it to the node we are considering.

– The best node is then the one with the highest rating.

• The algorithm will look ahead all possible moves to find out a sequence that leads to a win , if possible, in the shortest time.

• It attempts to maximize a likelihood of winning, while assuming that opponent will try to minimize that likelihood.

Heuristic:

Heuristics Function H= P(X wins) - P(O wins)

H=0 is DRAW.

X chooses Maximum value of H.

O chooses Minimum value of H.

How the heuristic works in this program:

- Winning: If the AI can win in the current state, it assigns a high positive score.
- Losing: If the human player can win in the current state, it assigns a high negative score.
- Draw: If the game ends in a draw, it assigns a neutral score (usually 0).

- Minimax Algorithm: The minimax algorithm recursively explores possible moves by both players, simulating the game until it reaches a terminal state (win, lose, or draw). At each step, it selects the move that leads to the maximum score for the AI player (maximizing player) and the minimum score for the human player (minimizing player), assuming both players play optimally.

Code:

```
#include <stdio.h>
#include <limits.h>
char board[9] = {'1', '2', '3', '4', '5', '6', '7', '8' ,'9'};
char ai = 'O';
char human = 'X';


void displayBoard() {
    printf("\n---------------------TIC-TAC-TOE--------------------------\n");
    printf("\n---------------------AI VS YOU--------------------------\n\n");
   printf("     |     |     \n");
    printf("  %c  |  %c  |  %c \n", board[0], board[1], board[2]);

    printf("_____|_____|_____\n");
    printf("     |     |     \n");

    printf("  %c  |  %c  |  %c \n", board[3], board[4], board[5]);

    printf("_____|_____|_____\n");
```

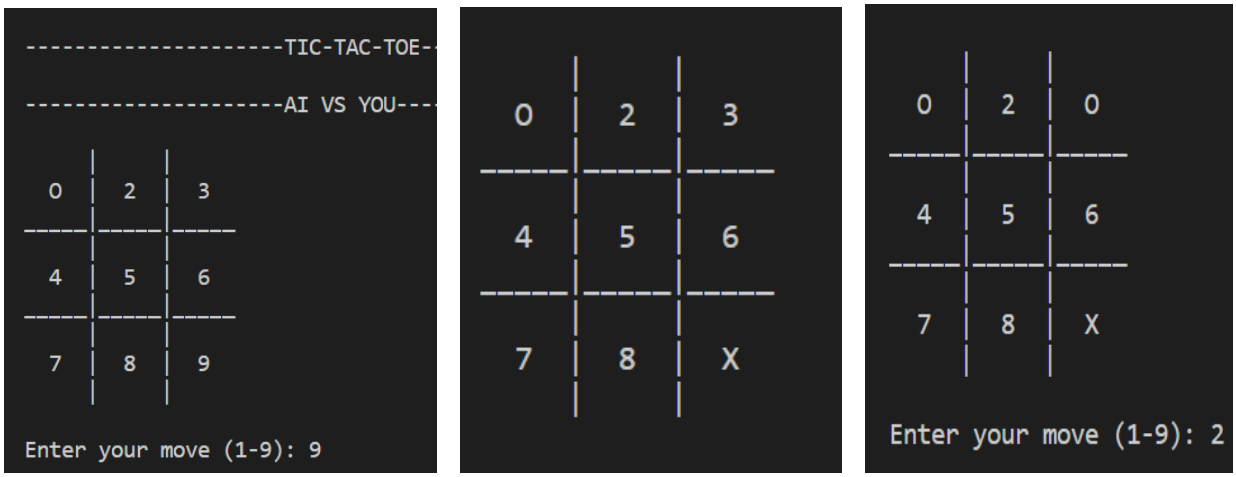
```
    printf("     |     |     \n");

    printf("  %c  |  %c  |  %c \n", board[6], board[7], board[8]);

    printf("     |     |     \n\n");

}


char checkWinner() {
    //checks for the winning condition and returns the player who wins
    for (int i = 0; i < 9; i += 3) {
        if (board[i] == board[i + 1] && board[i + 1] == board[i + 2] ) {
            return board[i];
        }
    }


    for (int i = 0; i < 3; i++) {
        if (board[i] == board[i + 3] && board[i + 3] == board[i + 6] ) {
            return board[i];
        }
    }


    if (board[0] == board[4] && board[4] == board[8] ) {
        return board[0];
    }


    if (board[2] == board[4] && board[4] == board[6] ) {
        return board[2];
    }

    //checks if the game is a draw if it is then returns 't'
    int draw = 1;
    for (int i = 0; i < 9; i++) {
        if (board[i] != 'X' && board[i] != 'O') {
            draw = 0;
            break;
        }
    }
    if (draw) {
        return 'D';
    }

    //means moves are pending
    return 'Y';
}
```

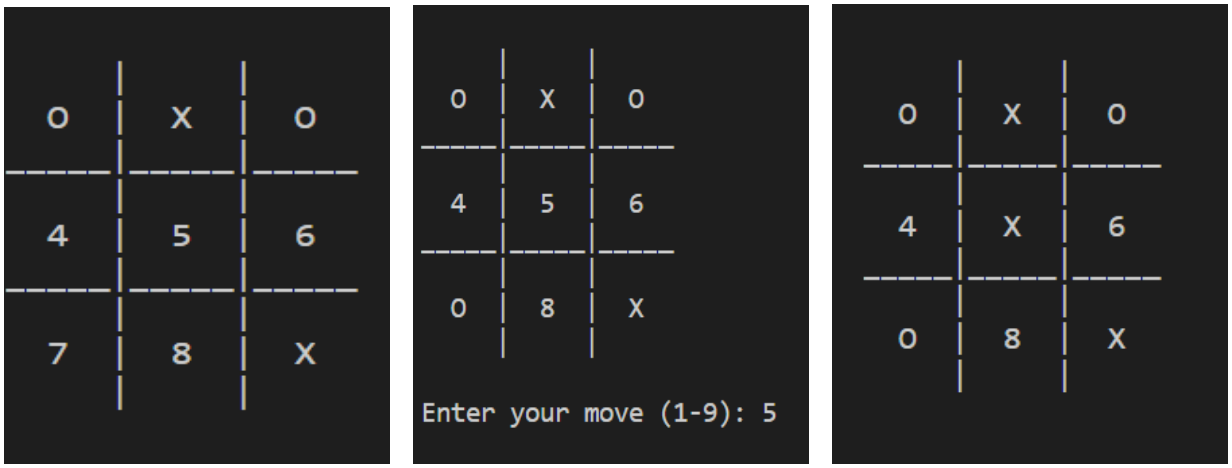
```
int minimax(char player) {
    //checks the result
    char result = checkWinner();

    //base cases
    if (result == ai) {
        return 1;
    } else if (result == human) {
        return -1;
    } else if (result == 'D') {
        return 0;
    }

    int bestScore;
    if (player == ai) {
        bestScore = INT_MIN;
        //puts move for every possible empty place
        for (int i = 0; i < 9; i++) {
            if (board[i] != 'X' && board[i] != 'O') {
                char ch = board[i];
                board[i] = ai;
                //calculates max score
                int score = minimax(human);
                //backtrack and put whtever previous char it was
                board[i] = ch;
                bestScore = (score > bestScore) ? score : bestScore;
            }
        }
    } else {
        //if player is human
        bestScore = INT_MAX;
        for (int i = 0; i < 9; i++) {
            if (board[i] != 'X' && board[i] != 'O') {
                char ch = board[i];
                board[i] = human;
                //calculates min score
                int score = minimax(ai);
                board[i] = ch;
                //if score is less than bestscore update it.
                bestScore = (score < bestScore) ? score : bestScore;
            }
        }
    }

    return bestScore;
}
```

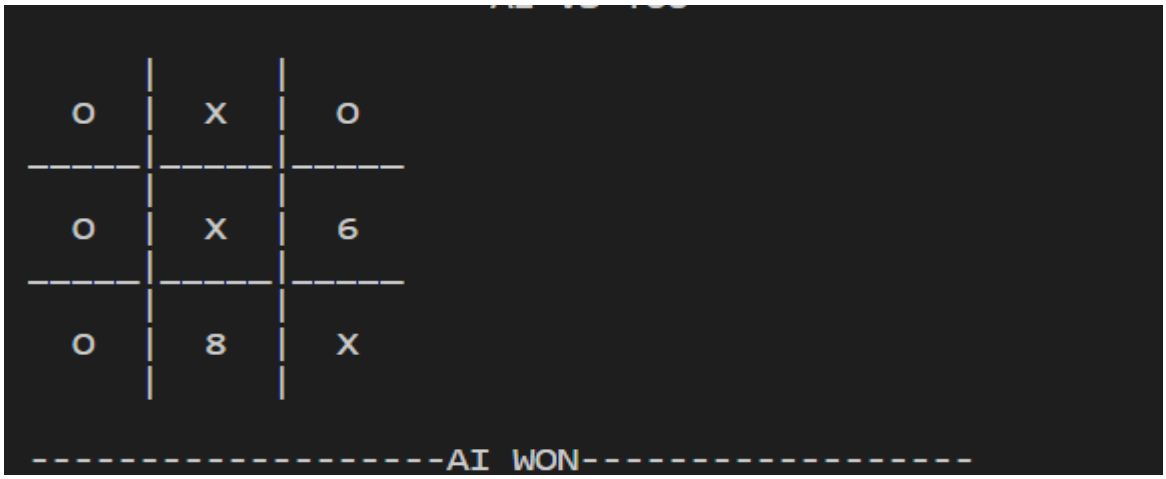
```
void aiMove() {
    //initializes for every ai move
    int bestScore = INT_MIN;
    int bestMove = -1;

    for (int i = 0; i < 9; i++) {
        //puts ai move on every possible empty place
        if (board[i] != 'X' && board[i] != 'O') {
            char ch = board[i];
            board[i] = ai;
            //calculates score for human checks if i put this move here what
will be the human's move
            //then ai's then human's likewise till each reaches base case and
thn returns the maximum score because ai is trying to maximize score
            int score = minimax(human);
            //backtrack's and puts whatever the char was in initial state
            board[i] = ch;

            //if the score we got is max then update the bestscore and
bestmove

            if (score > bestScore) {
                bestScore = score;
                bestMove = i;
            }
        }
    }
    //finally put the bestmove of ai
    board[bestMove] = ai;
}

//puts hum move in grid
void humanMove() {
    int move;

    do {
        printf("Enter your move (1-9): ");
        scanf("%d", &move);

        move--;
        if(board[move] == 'X' || board[move] == 'O'){
            printf("Enter valid number\n\n");
        }
    } while (move < 0 || move >= 9 || board[move] == 'X' || board[move] ==
'O');

    board[move] = human;
}
```

```c
int main() {



    int turn = 0;


    while (1) {
        displayBoard();

        if (checkWinner() != 'Y') {
            if(checkWinner()=='X'){
                printf("--------------------YOU WON------------------\n");
            }
           else if (checkWinner()=='O'){
                printf("-------------------AI WON------------------\n");
           }
        else if (checkWinner()=='D'){
                printf("-------------------MATCH  DRAW----------------------
\n");
           }
            break;
        }

     else{
       if (turn == 0) {
             aiMove();
             turn = 1;
        } else {
            humanMove();
            turn = 0;
        }
    }
    }

    return 0;
}
```

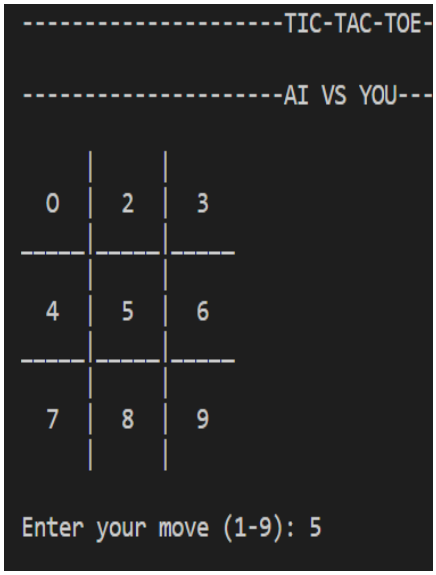
Output: Testcase1:

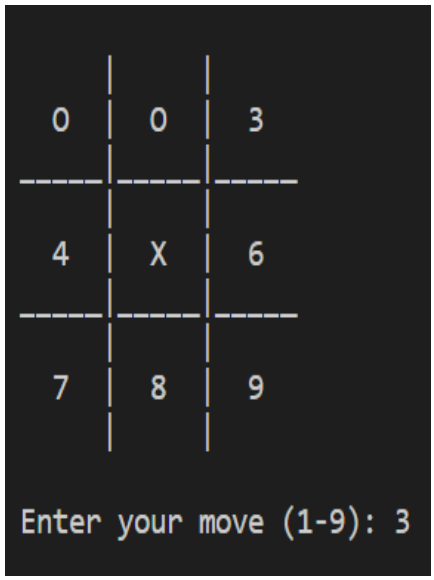


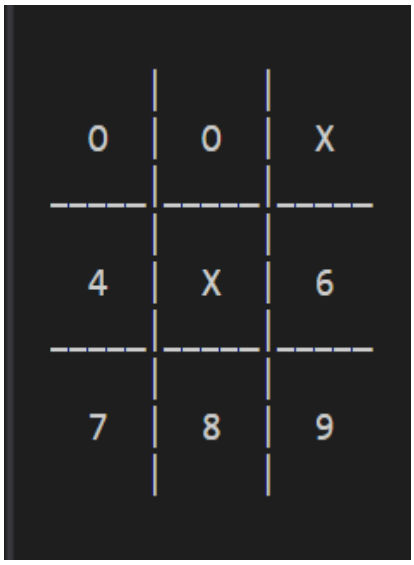


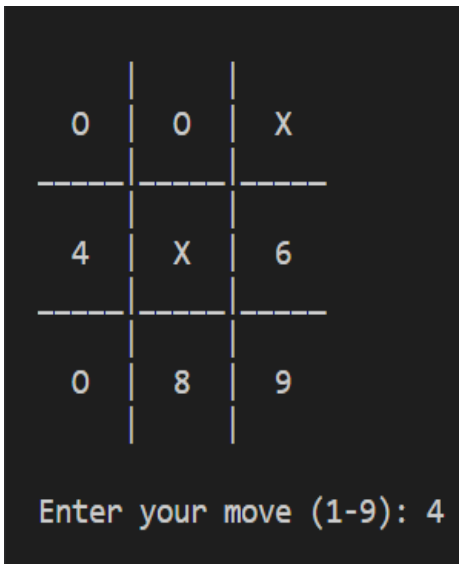


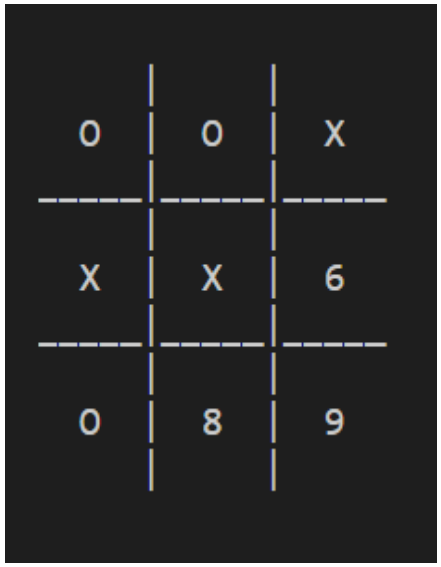
Testcase2: When It's a draw.

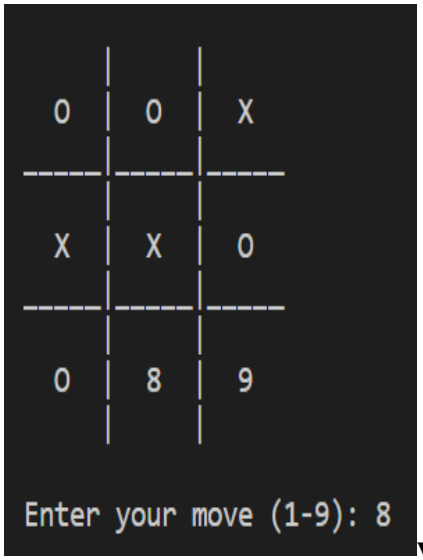






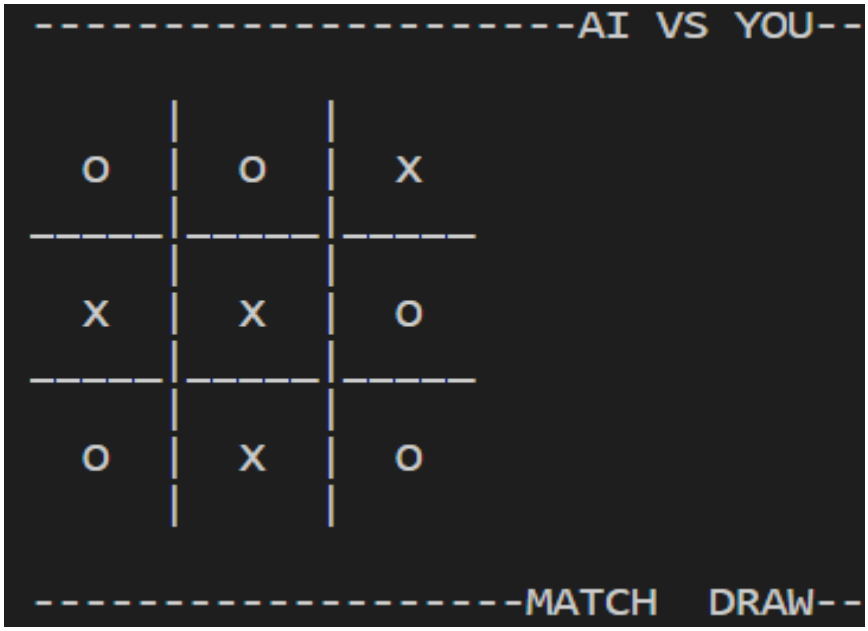




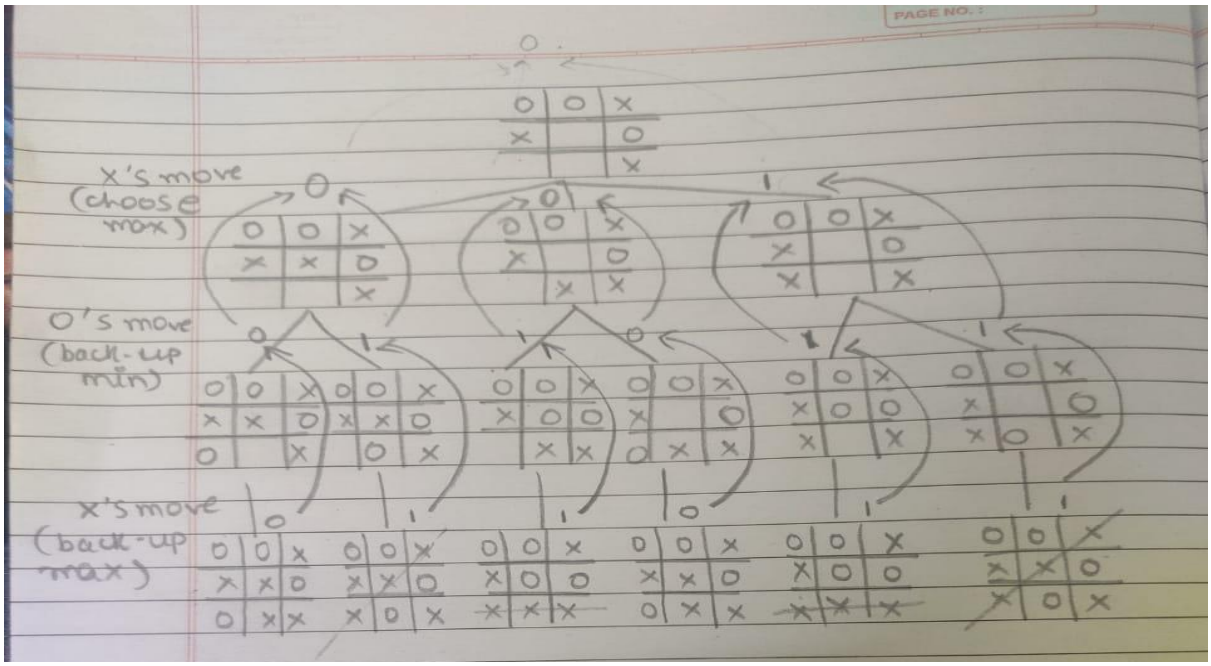

O | O | X
X | X | O
O | 8 | 9
Enter your move (1-9): 8


O | O | X
X | X | O
O | X | 9

v


----------AI VS YOU--
O | O | X
X | X | O
O | X | O
----------MATCH DRAW--

Visualizing Minimax: if there is chance of 'O' to win it will return -1,if there is chance of 'X' to win it will return 1,and if it is draw it will return 0.



Difference between AI and Non-AI Technique

	Non-AI	AI
1	Relies on predefined rules or strategies.	It is based on heuristics.
2	Typically simple logic based on fixed rules.	Utilizes advanced algorithms like minimax with pruning.
3	May not always make the best moves.	Tends to make optimal or near-optimal moves.
4	System keeps on following the defined set of rules to reach the solution.	System learns from its past, overcomes its mistakes and gives more optimal solutions to the problems
5	The chances of winning is possible for both players.	As AI technique uses Heuristic,it always aims to win.