Name : Harsh Doshi

Roll No. : 03

Batch : CS-B B1

PRN : 12110143

**Assignment 1**

**Objective:**

This assignment aims to implement both a non-AI two-player TIC-TAC-TOE game and an AI-driven TIC-TAC-TOE game utilizing the minimax algorithm. The non-AI game will provide a traditional gaming experience, allowing two players to take turns making moves on the TIC-TAC-TOE board. The AI version will showcase the application of the minimax algorithm, where the AI, represented by 'X,' aims to make optimal moves to win the game against the opponent, represented by 'O.'

**Theory:**

**Non-AI Game:**

* The non-AI TIC-TAC-TOE game is implemented in C, providing a simple and intuitive interface.
* The game features a 3x3 board, and players, represented by 'X' and 'O,' take turns to place their symbols on the board.
* After each move, the program checks for a winner by examining rows, columns, and diagonals for matching symbols.
* The game continues until a player wins or the board is full, resulting in a draw.

**AI Game:**

* The AI-driven TIC-TAC-TOE game employs the minimax algorithm, a recursive decision-making algorithm widely used in two-player turn-based games.
* The AI, represented by 'X,' aims to maximize its chances of winning by exploring all possible moves and outcomes up to a specified depth using recursion.
* The opponent, represented by 'O,' aims to minimize the AI's chances of winning.
* The game displays the board after each move, along with the calculated number of variations explored by the minimax algorithm.
* The AI makes decisions based on the maximization of its chances of winning while considering the opponent's attempts to minimize its chances.

**AI and Non AI two player game : TIC-TAC-TOE**

**Non – AI Game :**

Code:

#include <stdio.h>

void printBoard(char board[3][3]) {

    for (int i = 0; i < 3; i++) {

        for (int j = 0; j < 3; j++) {

            printf(" %c ", board[i][j]);

            if (j < 2) {

                printf("|");

            }

        }

        printf("\n");

        if (i < 2) {

            printf("-----------\n");

        }

    }

    printf("\n");

}

char checkWinner(char board[3][3]) {

    // Check rows, columns, and diagonals for a winner

    for (int i = 0; i < 3; i++) {

        if (board[i][0] == board[i][1] && board[i][1] == board[i][2] && board[i][0] != ' ') {

            return board[i][0];  // Row win

        }

        if (board[0][i] == board[1][i] && board[1][i] == board[2][i] && board[0][i] != ' ') {

            return board[0][i];  // Column win

        }

    }

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

        return board[0][0];  // Diagonal win

    }

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

        return board[0][2];  // Diagonal win

    }

    return ' ';

}

int isBoardFull(char board[3][3]) {

    for (int i = 0; i < 3; i++) {

        for (int j = 0; j < 3; j++) {

            if (board[i][j] == ' ') {

                return 0; // Board is not full

            }

        }

    }

    return 1; // Board is full

}

void ticTacToe() {

    char board[3][3] = {{' ', ' ', ' '}, {' ', ' ', ' '}, {' ', ' ', ' '}};

    char current\_player = 'X';

    while (1) {

        printBoard(board);

        int row, col;

        printf("Player %c, enter row (0-2): ", current\_player);

        scanf("%d", &row);

        printf("Player %c, enter column (0-2): ", current\_player);

        scanf("%d", &col);

        if (0 <= row && row < 3 && 0 <= col && col < 3 && board[row][col] == ' ') {

            board[row][col] = current\_player;

            char winner = checkWinner(board);

            if (winner != ' ') {

                printBoard(board);

                printf("Player %c wins!\n", winner);

                break;

            } else if (isBoardFull(board)) {

                printBoard(board);

                printf("It's a tie!\n");

                break;

            } else {

                current\_player = (current\_player == 'X') ? 'O' : 'X';

            }

        } else {

            printf("Invalid move. Try again.\n");

        }

    }

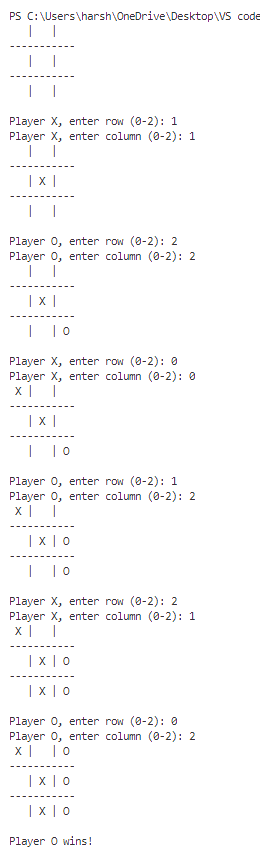
}

int main() {

    ticTacToe();

    return 0;

}

Output :

**AI Game :**

Code :

 #include <stdio.h>

#define X -1

#define O -2

#define MAX\_SIZE 50 //This is the Max size of the Input Buffer

int turn = O;

int board[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};

int variations = 0; //To keep track of no. of variations the ai has seen

void copyBoard(int from[], int to[]){

    for(int i = 0; i < 9; i++){

        to[i] = from[i];

    }

}

//gets line WITHOUT \n

int getl(char s[], int lim){

    int c, i;

    for(i = 0; i < lim-1 && (c = getchar()) != EOF && c != '\n'; i++)

        s[i] = c;

    s[i] = '\0';

    return i;

}

//Converts char[] to int

int bufferToNum(char buffer[]){

    int n = 0;

    for(int i = 0; buffer[i] != '\0'; i++){

        n = 10 \* n + buffer[i] - '0';

    }

    return n;

}

//converts the board numbers to char to display

char boardToChar(int i){

    int a = board[i];

    if (a == X){

        return 'X';

    } else if (a == O){

        return 'O';

    } else {

       return a + '0';

    }

}

//prints board

void printBoard(){

    printf("=============\n| %c | %c | %c |\n-------------\n| %c | %c | %c |\n-------------\n| %c | %c | %c |\n=============\n", boardToChar(0), boardToChar(1), boardToChar(2), boardToChar(3), boardToChar(4), boardToChar(5), boardToChar(6), boardToChar(7), boardToChar(8));

}

//alternates turn

void alternateTurn(){

    if (turn == O){

        turn = X;

    } else if (turn == X){

        turn = O;

    }

}

//returns 1 if draw, return 0 if not a draw

int drawCheck(int l\_board[]){

    for(int i = 0; i < 9; i++){

        if (l\_board[i] == i+1){

            return 0;

        }

    }

    return 1;

}

//returns X if X won and O if O one and 0 if nobody winning

int winCheck(int l\_board[]){

    //Rows

    for (int i = 0; i < 3; i++){

        if (l\_board[3\*i] == l\_board[3\*i + 1] && l\_board[3\*i + 1] == l\_board[3\*i + 2]){

            return l\_board[3\*i];

        }

    }

    //Columns

    for (int j = 0; j < 3; j++){

        if (l\_board[j] == l\_board[3 + j] && l\_board[3 + j] == l\_board[6 + j]){

            return l\_board[j];

        }

    }

    //Diagonal Top Left to Bottom Right

    if (l\_board[0] == l\_board[4] && l\_board[0] == l\_board[8]){

        return l\_board[0];

    }

    //Diagonal Top Right to bottom Left

    if (l\_board[2] == l\_board[4] && l\_board[2] == l\_board[6]){

        return l\_board[2];

    }

    return 0;

}

//1 if nothing is ther and 0 if something was already ther

int putInBoard(int l\_board[], int pos, int newVal){

    if (l\_board[pos] == pos+1){

        l\_board[pos] = newVal;

        return 1;

    } else

    {

        return 0;

    }

}

//X if X win, O if O win, 0 if draw, 1 if nothing

int gameState(int l\_board[]){

    int wc = winCheck(l\_board);

    if (wc == X){

        return X;

    } else if(wc == O){

        return O;

    } else {

        if (drawCheck(l\_board)){

            return 0;

        }

    }

    return 1;

}

void legalMoves(int l\_board[], int output[]){

    for(int i = 0; i < 9; i++){

        if (l\_board[i] == i+1){

            output[i] = 1;

        } else {

            output[i] = 0;

        }

    }

}

int max(int a, int b){

    return a>b ? a : b;

}

int min(int a, int b){

    return a<b ? a : b;

}

//X is ai

int minimax(int l\_board[], int depth, int maximising){

    int gs = gameState(l\_board);

    variations++;

    if (gs == X){

        return 10;

    } else if (gs == O){

        return -10;

    } else if (gs == 0){

        return 0;

    }

    if (depth == 0){

        return 0;

    }

    if (maximising){

        //Its AI's Turn so it has to maximise

        int val = -100;

        int legalMovesArr[9];

        legalMoves(l\_board, legalMovesArr);

        for (int i = 0; i < 9; i++){

            if (legalMovesArr[i]){

                int tempBoard[9];

                copyBoard(l\_board, tempBoard);

                putInBoard(tempBoard, i, X);

                val = max(minimax(tempBoard, depth-1, 0), val);

            }

        }

        return val;

    } else {

        int val = 100;

        int legalMovesArr[9];

        legalMoves(l\_board, legalMovesArr);

        for (int i = 0; i < 9; i++){

            if (legalMovesArr[i]){

                int tempBoard[9];

                copyBoard(l\_board, tempBoard);

                putInBoard(tempBoard, i, O);

                val = min(minimax(tempBoard, depth-1, 1), val);

            }

        }

        return val;

    }

}

int ai(int l\_board[], int depth){

    int legalMovesArr[9];

    legalMoves(board, legalMovesArr);

    int val = -100;

    int best\_move = 0;

    for (int i = 0; i < 9; i++){

        if (legalMovesArr[i]){

            int tempBoard[9];

            copyBoard(l\_board, tempBoard);

            putInBoard(tempBoard, i, X);

            int temp = minimax(tempBoard, depth-1, 0);

            if (val <= temp){

                val = temp;

                best\_move = i;

            }

        }

    }

    return best\_move;

}

int main(){

    printBoard();

    int gameOn = 0;

    char buffer[MAX\_SIZE];

    while(!gameOn){

        if (turn == O){

            printf("%c's turn: ", turn == X ? 'X' : 'O');

            getl(buffer, MAX\_SIZE);

            int num = bufferToNum(buffer);

            while (num <= 0 || num > 9){

                printf("Please enter an integer between 1 and 9: ");

                getl(buffer, MAX\_SIZE);

                num = bufferToNum(buffer);

            }

            if (putInBoard(board, num-1, turn)){

                ;

            } else {

                while(!putInBoard(board, num-1, turn)){

                    printf("Something already exists, Please enter a new number: ");

                    getl(buffer, MAX\_SIZE);

                    num = bufferToNum(buffer);

                }

            }

        } else {

            putInBoard(board, ai(board, 8), X);

            printf("Calculated %d variations\n", variations);

            variations = 0;

        }

        printBoard();

        alternateTurn();

        int gs = gameState(board);

        if (gs == X){

            printf("X won!");

            return 0;

        } else if (gs == O){

            printf("O won!");

            return 0;

        } else if (gs == 0){

            printf("Draw!");

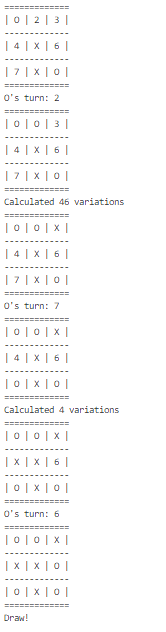
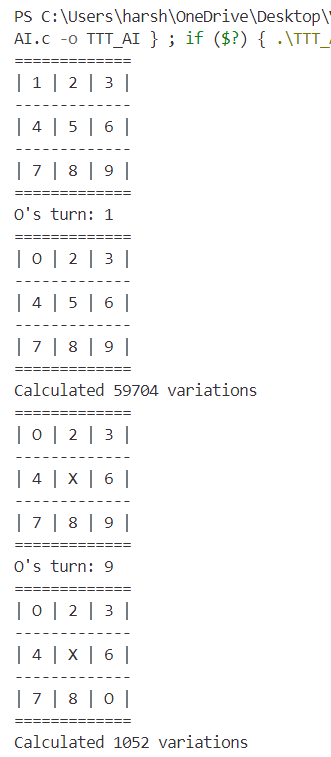
            return 0;

        }

    }

    return 0;

}

Output :

**Conclusion:**

* The non-AI TIC-TAC-TOE game provides a user-friendly interface for two players to enjoy a classic game.
* The AI TIC-TAC-TOE game demonstrates the effectiveness of the minimax algorithm in strategic decision-making, showcasing its ability to explore multiple game variations to determine the optimal move.