Bansilal Ramnath Agarwal Charitable Trust's
VISHWAKARMA INSTITUTE OF TECHNOLOGY – PUNE

ARTIFICIAL INTELLIGENCE

Division	A
Batch	2
GR-no	12311493
Roll no	54
Name	Atharva Kangralkar

Assignment 1:

Implementation of AI and Non-AI techniques by implementing any two-player game

1. **Non-AI Technique(Using 2-3-5 Strategy):**

Description:-

- **Code:**

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <time.h>

// Magic Square Numbers (corresponds to board positions)
// 8 1 6
// 3 5 7
// 4 9 2
const int MAGIC_SQUARE[9] = {8, 1, 6, 3, 5, 7, 4, 9, 2};
const int WINNING_SUM = 15;  // Sum needed to win
const char PLAYER = 'X';     // Human player symbol
const char COMPUTER = 'O';   // Computer symbol
const char EMPTY = ' ';      // Empty cell symbol

// Function to print the current game board
void showBoard(char board[]) {
    printf("\n");
    printf(" %c | %c | %c \n", board[0], board[1], board[2]);
    // printf("---+---+---\n");
    printf(" %c | %c | %c \n", board[3], board[4], board[5]);
    // printf("---+---+---\n");
    printf(" %c | %c | %c \n", board[6], board[7], board[8]);
    // printf("\n");
}

// Check if a position is empty
bool isPositionEmpty(char board[], int position) {
    return board[position] == EMPTY;
}

// Place a symbol (X or O) on the board
void placeSymbol(char board[], int position, char symbol) {
    board[position] = symbol;
}

// Get all magic numbers for a player's positions
void getMagicNumbers(char board[], char playerSymbol, int numbers[], int
*count) {
    *count = 0;
```

```
    // Check each board position
    for(int i = 0; i < 9; i++) {
        if(board[i] == playerSymbol) {
            // Store the magic number for this position
            numbers[*count] = MAGIC_SQUARE[i];
            (*count)++;
        }
    }
}

// Check if any combination of three numbers sums to 15
bool checkWin(int numbers[], int count) {
    // Need at least 3 numbers to win
    if(count < 3) return false;

    // Try all possible combinations of three numbers
    for(int i = 0; i < count - 2; i++) {
        for(int j = i + 1; j < count - 1; j++) {
            for(int k = j + 1; k < count; k++) {
                if(numbers[i] + numbers[j] + numbers[k] == WINNING_SUM) {
                    return true;
                }
            }
        }
    }
    return false;
}

// Find a winning move or blocking move
int findBestMove(char board[], int playerNumbers[], int playerCount, char
symbol) {
    // Try each empty position on the board
    for(int pos = 0; pos < 9; pos++) {
        if(isPositionEmpty(board, pos)) {
            // Check if adding this position's magic number creates a sum of 15
            for(int i = 0; i < playerCount; i++) {
                for(int j = i + 1; j < playerCount; j++) {
                    int sum = playerNumbers[i] + playerNumbers[j] +
MAGIC_SQUARE[pos];
                    if(sum == WINNING_SUM) {
```

```
                    return pos;
                }
            }
        }
    }
    return -1;  // No winning move found
}

// Make computer's move
int makeComputerMove(char board[], int computerNumbers[], int
humanNumbers[],
                     int computerCount, int humanCount) {
    int position;

    // First, try to win
    position = findBestMove(board, computerNumbers, computerCount,
COMPUTER);
    if(position != -1) {
        return position;
    }

    // If can't win, try to block player
    position = findBestMove(board, humanNumbers, humanCount,
PLAYER);
    if(position != -1) {
        return position;
    }

    // If no winning or blocking move, make random move
    do {
        position = rand() % 9;
    } while(!isPositionEmpty(board, position));

    return position;
}

int main() {
    // Initialize game
    char board[9] = {EMPTY, EMPTY, EMPTY, EMPTY, EMPTY,
```

```
EMPTY, EMPTY, EMPTY, EMPTY};
    int humanNumbers[5] = {0};    // Store player's magic numbers
    int computerNumbers[5] = {0}; // Store computer's magic numbers
    int humanCount = 0, computerCount = 0;
    int moveCount = 0;
    int position;

    // Seed random number generator for computer's random moves
    srand(time(NULL));

    // Show game instructions
    printf("Welcome to Tic Tac Toe!\n");
    printf("You are X, Computer is O\n");
    printf("Enter positions (1-9) as shown below:\n");
    printf("1 2 3\n4 5 6\n7 8 9\n\n");

    // Main game loop
    while(moveCount < 9) {
        // Player's turn
        showBoard(board);
        printf("Your turn. Enter position (1-9): ");
        scanf("%d", &position);
        position--; // Convert to 0-based index

        // Check if move is valid
        if(position < 0 || position > 8) {
            printf("Invalid position! Please enter 1-9.\n");
            continue;
        }
        if(!isPositionEmpty(board, position)) {
            printf("Position already taken! Try again.\n");
            continue;
        }

        // Make player's move
        placeSymbol(board, position, PLAYER);
        getMagicNumbers(board, PLAYER, humanNumbers, &humanCount);
        moveCount++;

        // Check if player won
```

```
        if(checkWin(humanNumbers, humanCount)) {
            showBoard(board);
            printf("Congratulations! You win!\n");
            break;
        }

        // Check for draw
        if(moveCount == 9) {
            showBoard(board);
            printf("It's a draw!\n");
            break;
        }

        // Computer's turn
        printf("\nComputer's turn...\n");
        position = makeComputerMove(board, computerNumbers,
humanNumbers,
                                    computerCount, humanCount);
        placeSymbol(board, position, COMPUTER);
        getMagicNumbers(board, COMPUTER, computerNumbers,
&computerCount);
        moveCount++;

        // Check if computer won
        if(checkWin(computerNumbers, computerCount)) {
            showBoard(board);
            printf("Computer wins!\n");
            break;
        }
    }

    return 0;
}
```

Screenshots/Output:

```
Welcome to Tic Tac Toe!
You are X, Computer is O
Enter positions (1-9) as shown below:
1 2 3
4 5 6
7 8 9



   |   |
   |   |
   |   |
Your turn. Enter position (1-9): 4

Computer's turn...

   |   |
 X |   |
   | O |
Your turn. Enter position (1-9): 3

Computer's turn...

   |   | X
 X |   | O
   | O |
Your turn. Enter position (1-9): 2
```

```
Computer's turn...

 O | X | X
 X |   | O
   | O |
Your turn. Enter position (1-9): 1
Position already taken! Try again.

 O | X | X
 X |   | O
   | O |
Your turn. Enter position (1-9): 5

Computer's turn...

 O | X | X
 X | X | O
 O | O |
Your turn. Enter position (1-9): 9

 O | X | X
 X | X | O
 O | O | X
It's a draw!
```

2. **AI Technique(Using minmax):**

Description:-

- **Code:**

```
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>

#define EMPTY ' '
#define PLAYER_X 'x'
#define PLAYER_O 'o'
#define SIZE 3

// Function prototypes
void print_board(char board[SIZE][SIZE]);
char check_winner(char board[SIZE][SIZE]);
void convert_move(int move, int* row, int* col);
int minimax(char board[SIZE][SIZE], int depth, bool is_maximizing);
void best_move(char board[SIZE][SIZE]);

// Function to print the board
void print_board(char board[SIZE][SIZE]) {
    printf("-(%d) -(%d) -(%d)\n", 1, 2, 3);
    printf("-(%d) -(%d) -(%d)\n", 4, 5, 6);
    printf("-(%d) -(%d) -(%d)\n\n", 7, 8, 9);

    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            if (board[i][j] == EMPTY) {
                printf(" -");
            } else {
                printf("%c(%d)", board[i][j], i * 3 + j + 1);
            }
            if (j < SIZE - 1) printf(" ");
        }
        printf("\n");
    }
    printf("\n");
}

// Function to convert 1-9 move to board indices
```

```
void convert_move(int move, int* row, int* col) {
    move--;  // convert to 0-based index
    *row = move / 3;
    *col = move % 3;
}

// Function to check if there's a winner or a tie
char check_winner(char board[SIZE][SIZE]) {
    // Check rows
    for (int i = 0; i < SIZE; i++) {
        if (board[i][0] != EMPTY && board[i][0] == board[i][1] &&
board[i][1] == board[i][2]) {
            return board[i][0];
        }
    }

    // Check columns
    for (int j = 0; j < SIZE; j++) {
        if (board[0][j] != EMPTY && board[0][j] == board[1][j] &&
board[1][j] == board[2][j]) {
            return board[0][j];
        }
    }

    // Check diagonals
    if (board[0][0] != EMPTY && board[0][0] == board[1][1] &&
board[1][1] == board[2][2]) {
        return board[0][0];
    }
    if (board[0][2] != EMPTY && board[0][2] == board[1][1] &&
board[1][1] == board[2][0]) {
        return board[0][2];
    }

    // Check for a tie
    bool is_tie = true;
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            if (board[i][j] == EMPTY) {
                is_tie = false;
```

```
                break;
            }
        }
    }
    if (is_tie) return 'T'; // 'T' represents a tie

    return EMPTY; // No winner yet
}

// Minimax algorithm implementation
int minimax(char board[SIZE][SIZE], int depth, bool is_maximizing) {
    char winner = check_winner(board);

    if (winner == PLAYER_X) return -10 + depth;
    if (winner == PLAYER_O) return 10 - depth;
    if (winner == 'T') return 0;

    if (is_maximizing) {
        int best_score = INT_MIN;
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                if (board[i][j] == EMPTY) {
                    board[i][j] = PLAYER_O;
                    int score = minimax(board, depth + 1, false);
                    board[i][j] = EMPTY;
                    if (score > best_score) best_score = score;
                }
            }
        }
        return best_score;
    } else {
        int best_score = INT_MAX;
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                if (board[i][j] == EMPTY) {
                    board[i][j] = PLAYER_X;
                    int score = minimax(board, depth + 1, true);
                    board[i][j] = EMPTY;
                    if (score < best_score) best_score = score;
                }
```

```
            }
        }
        return best_score;
    }
}

// Function to find the best move for the AI
void best_move(char board[SIZE][SIZE]) {
    int best_score = INT_MIN;
    int move_row = -1, move_col = -1;

    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            if (board[i][j] == EMPTY) {
                board[i][j] = PLAYER_O;
                int score = minimax(board, 0, false);
                board[i][j] = EMPTY;
                if (score > best_score) {
                    best_score = score;
                    move_row = i;
                    move_col = j;
                }
            }
        }
    }

    if (move_row != -1 && move_col != -1) {
        board[move_row][move_col] = PLAYER_O;
    }
}

// Main function to run the game
int main() {
    char board[SIZE][SIZE] = {
        {EMPTY, EMPTY, EMPTY},
        {EMPTY, EMPTY, EMPTY},
        {EMPTY, EMPTY, EMPTY}
    };

    char current_player = PLAYER_X;
```

```
    while (true) {
        printf("Welcome to Tic-Tac-Toe!\n");
        print_board(board);
        char winner = check_winner(board);

        if (winner != EMPTY) {
            if (winner == 'T') {
                printf("It's a tie!\n");
            } else {
                printf("Player %c wins!\n", winner);
            }
            printf("Game Over!\n");
            break;
        }

        if (current_player == PLAYER_X) {
            int move;
            printf("Player x, enter your move (1-9) : ");
            scanf("%d", &move);

            int row, col;
            convert_move(move, &row, &col);

            if (move < 1 || move > 9 || board[row][col] != EMPTY) {
                printf("Invalid move. Try again.\n");
                continue;
            }

            board[row][col] = PLAYER_X;
        } else {
            printf("AI player's turn!\n");
            best_move(board);
        }

        current_player = (current_player == PLAYER_X) ? PLAYER_O :
PLAYER_X;
    }

    return 0;
```

```
}
```

Screenshots/Output:

```
Welcome to Tic-Tac-Toe!
-(1) -(2) -(3)
-(4) -(5) -(6)
-(7) -(8) -(9)

 -  -  -
 -  -  -
 -  -  -

Player x, enter your move (1-9) : 2
Welcome to Tic-Tac-Toe!
-(1) -(2) -(3)
-(4) -(5) -(6)
-(7) -(8) -(9)

 - x(2)  -
 -  -  -
 -  -  -
```

```
AI player's turn!
Welcome to Tic-Tac-Toe!
-(1) -(2) -(3)
-(4) -(5) -(6)
-(7) -(8) -(9)

o(1) x(2)  -
 -   -   -
 -   -   -

Player x, enter your move (1-9) : 5
Welcome to Tic-Tac-Toe!
-(1) -(2) -(3)
-(4) -(5) -(6)
-(7) -(8) -(9)

o(1) x(2)  -
 - x(5)  -
 -   -   -
```

```
AI player's turn!
Welcome to Tic-Tac-Toe!
-(1) -(2) -(3)
-(4) -(5) -(6)
-(7) -(8) -(9)

o(1) x(2)  -
 - x(5)  -
 - o(8)  -

Player x, enter your move (1-9) : 3
Welcome to Tic-Tac-Toe!
-(1) -(2) -(3)
-(4) -(5) -(6)
-(7) -(8) -(9)

o(1) x(2) x(3)
 - x(5)  -
 - o(8)  -
```

```
AI player's turn!
Welcome to Tic-Tac-Toe!
-(1) -(2) -(3)
-(4) -(5) -(6)
-(7) -(8) -(9)

o(1) x(2) x(3)
 - x(5)  -
o(7) o(8)  -

Player x, enter your move (1-9) : 4
Welcome to Tic-Tac-Toe!
-(1) -(2) -(3)
-(4) -(5) -(6)
-(7) -(8) -(9)

o(1) x(2) x(3)
x(4) x(5)  -
o(7) o(8)  -

AI player's turn!
Welcome to Tic-Tac-Toe!
-(1) -(2) -(3)
-(4) -(5) -(6)
-(7) -(8) -(9)

o(1) x(2) x(3)
x(4) x(5)  -
o(7) o(8) o(9)

Player o wins!
Game Over!
```