Member	Percentage
Nur Azizah 5025211252 DAA D	**33.33% -** Game Logic**,** Implementing DFS algorithm
Andina Safitri Innayah 5025221204 DAA G	**33.33% -** Perform analysis, testing, and wrote the final report.
Aulia Daffa Rahmani 5025221205 DAA D	**33.33% -** Game logic(Handle turn and move validation), adding report in github

TIC TAC TOE

PROJECT DESIGN

About

The Tic Tac Toe game is a classic two-player game where the goal is to form a straight line (horizontal, vertical, or diagonal) with three X's or O's on a 3x3 grid. This project creates a single-player version where the player competes against a bot with three difficulty levels: Easy (random moves), Medium (heuristic blocking), and Hard (DFS-based Minimax).

Design Overview

- The board is a 3x3 grid, initialized as a 2D character array (arr[3][3]) filled with '.' indicating empty cells.

- Player vs. Bot setup:
  - Player marks with 'X'.
  - Bot marks with 'O'.

- User interface is console-based with functions:
  - mainScreen(): Displays the main screen.
  - gameBoard(): Displays the current game board and status.
  - init(): Initializes the game board.

- Game control:
  - User chooses if the bot starts first.
  - User selects bot difficulty (0=Easy, 1=Medium, 2=Hard).

Bot Design

- Easy Bot: Random move selection.

- Medium Bot:
    - First, tries to win in one move.
    - If not possible, blocks the player’s immediate win.
    - Otherwise, makes a random move.
- Hard Bot: Implements the Minimax algorithm to choose the optimal move, making it unbeatable.

ALGORITHM ANALYSIS

Game Mechanics

- Win Condition Checking (result()):
    - Checks rows, columns, and diagonals for identical non-empty marks.
    - Returns 'X' (player win), 'O' (bot win), 'T' (draw), or '-' (game continues).

Bot Logic

Easy Bot

- Randomly selects an empty cell using rand() % 3.
- Simple but not strategic.

Medium Bot

- Uses tryMove() to check for immediate winning moves (for bot) or blocking moves (against player).
- If neither is available, falls back to botMoveEasy().

Hard Bot (Minimax)

- Recursive minimax search:
    - Terminal States: Bot win (+10), player win (-10), draw (0).
    - Bot Turn: Maximizes score by simulating placing 'O' in empty cells.
    - Player Turn: Minimizes score by simulating placing 'X'.
- Time Complexity:
    - The minimax algorithm explores up to O(9!) game states, but practical branching is reduced since the board fills quickly.
- Space Complexity: O(n), where n is the recursion depth (maximum 9).

SOURCE CODE

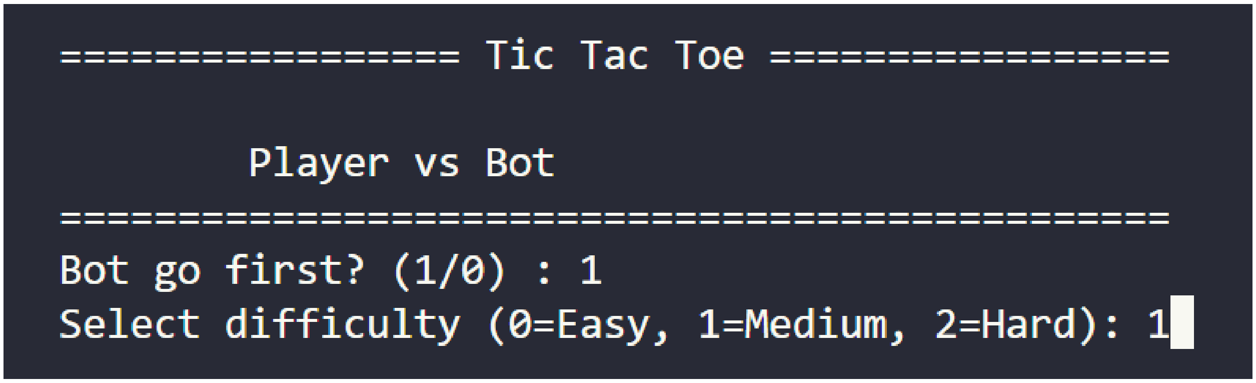
```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>

char arr[3][3];

void init() {
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            arr[i][j] = '.';
```

```
}

void mainScreen() {
    printf("================= Tic Tac Toe =================\n\n");
    printf("\tPlayer vs Bot\n");
    printf("===============================================\n");
}

void gameBoard(const char* P1, const char* P2) {
    printf("=============== Tic Tac Toe ===============\n\n");
    printf("\t\t %c | %c | %c \n", arr[0][0], arr[0][1], arr[0][2]);
    printf("\t\t---+---+---\n");
    printf("\t\t %c | %c | %c \n", arr[1][0], arr[1][1], arr[1][2]);
    printf("\t\t---+---+---\n");
    printf("\t\t %c | %c | %c \n", arr[2][0], arr[2][1], arr[2][2]);
    printf("\n\t %s : 'X'\t %s : 'O'\n", P1, P2);
    printf("===========================================\n");
}

char result() {
    for (int i = 0; i < 3; i++) {
         if (arr[i][0] == arr[i][1] && arr[i][1] == arr[i][2] && arr[i][0] !=
'.')
            return arr[i][0];
         if (arr[0][i] == arr[1][i] && arr[1][i] == arr[2][i] && arr[0][i] !=
'.')
            return arr[0][i];
    }
    if (arr[0][0] == arr[1][1] && arr[1][1] == arr[2][2] && arr[0][0] != '.')
        return arr[0][0];
    if (arr[0][2] == arr[1][1] && arr[1][1] == arr[2][0] && arr[0][2] != '.')
        return arr[0][2];

    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (arr[i][j] == '.')
                return '-';

    return 'T';
}

bool isMovesLeft() {
    for (int i = 0; i < 3; i++)
```

```
        for (int j = 0; j < 3; j++)
            if (arr[i][j] == '.')
                return true;
    return false;
}

int minimax(bool isBotTurn) {
    char res = result();
    if (res == 'O') return +10;
    if (res == 'X') return -10;
    if (res == 'T') return 0;

    if (isBotTurn) {
        int bestScore = -1000;
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (arr[i][j] == '.') {
                    arr[i][j] = 'O';
                    int score = minimax(false);
                    arr[i][j] = '.';
                    if (score > bestScore) bestScore = score;
                }
            }
        }
        return bestScore;
    } else {
        int bestScore = 1000;
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (arr[i][j] == '.') {
                    arr[i][j] = 'X';
                    int score = minimax(true);
                    arr[i][j] = '.';
                    if (score < bestScore) bestScore = score;
                }
            }
        }
        return bestScore;
    }
}

void botMoveHard() {
    int bestScore = -1000;
    int moveX = -1, moveY = -1;

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (arr[i][j] == '.') {
                arr[i][j] = 'O';
                int score = minimax(false);
                arr[i][j] = '.';
                if (score > bestScore) {
                    bestScore = score;
                    moveX = i;
                    moveY = j;
                }
            }
        }
    }
    if (moveX != -1 && moveY != -1)
        arr[moveX][moveY] = 'O';
}

void botMoveEasy() {
    int x, y;
    srand((unsigned)time(NULL));
    while (1) {
        x = rand() % 3;
        y = rand() % 3;
        if (arr[x][y] == '.') {
            arr[x][y] = 'O';
            break;
        }
    }
}

bool tryMove(char player, int* moveX, int* moveY) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (arr[i][j] == '.') {
                arr[i][j] = player;
                if (result() == player) {
                    arr[i][j] = '.';
                    *moveX = i;
                    *moveY = j;
                    return true;
                }
                arr[i][j] = '.';
            }
        }
    }
    return false;
}

void botMoveMedium() {
    int x = -1, y = -1;

    if (tryMove('O', &x, &y)) {
        arr[x][y] = 'O';
        return;
    }
    if (tryMove('X', &x, &y)) {
        arr[x][y] = 'O';
        return;
    }
    botMoveEasy();
}

int main() {
    while (1) {
        system("cls");
        mainScreen();
        init();

        int cnt = 0;
        int turn;
        int difficulty;

        printf("Bot go first? (1/0) : ");
        scanf("%d", &turn);

        printf("Select difficulty (0=Easy, 1=Medium, 2=Hard): ");
        scanf("%d", &difficulty);

        if (turn) cnt++;

        while (1) {
            system("cls");
            gameBoard("Player", "Bot");
            if (cnt % 2 == 0) {
                printf("\n\t     Player your move: \n\t\t    ");
                int x, y;
                scanf("%d %d", &x, &y);
                  if (x < 0 || x > 2 || y < 0 || y > 2 || arr[x][y] != '.')
continue;
                arr[x][y] = 'X';
                cnt++;
            } else {
                printf("\nBot is thinking...\n");
                if (difficulty == 0) botMoveEasy();
                else if (difficulty == 1) botMoveMedium();
                else botMoveHard();
                cnt++;
            }

            char res = result();
            if (res != '-') {
                system("cls");
                gameBoard("Player", "Bot");
                if (res == 'O')
                    printf("\n\t\t  Bot Wins.\n");
                else if (res == 'X')
                    printf("\n\t\tPlayer Wins.\n");
                else if (res == 'T')
                    printf("\n\t\t   Draw.\n");
                break;
            }
        }

        int repeat;
        printf("\n===========================================\n\n");
        printf("\t   Play again? (1/0) : ");
        scanf("%d", &repeat);
        if (repeat == 0) break;
    }
    return 0;
}
```

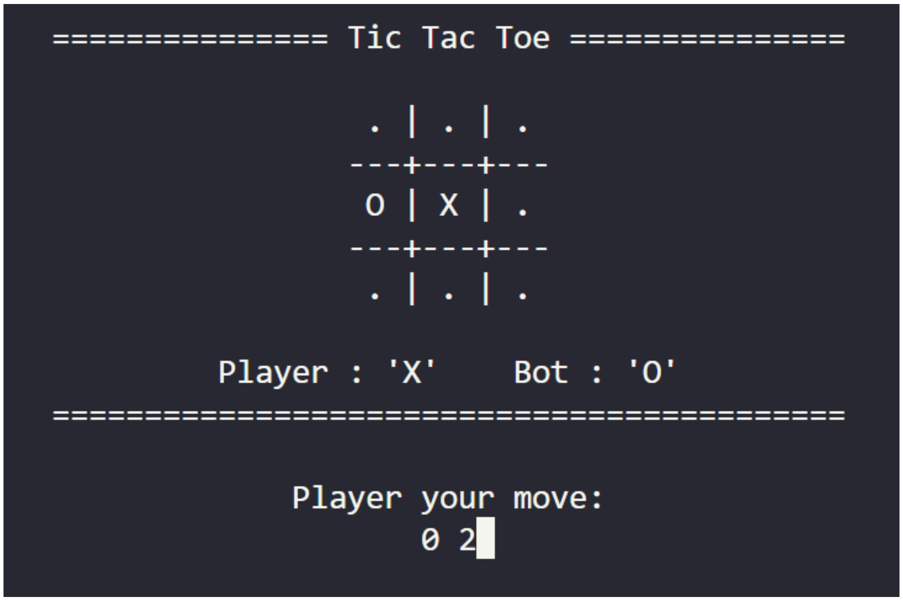
OUTPUT

1. **Intro & select level**

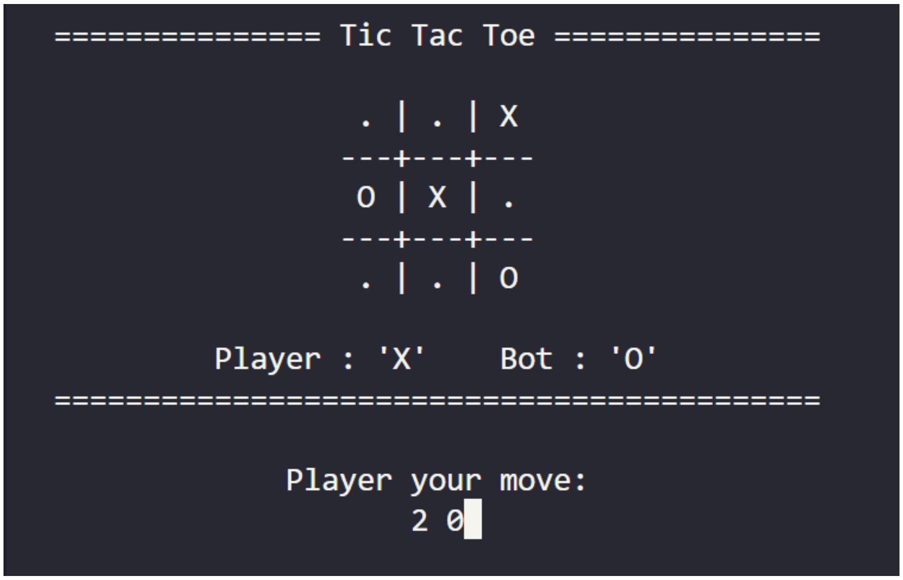


```
================= Tic Tac Toe =================


        Player vs Bot
===============================================
Bot go first? (1/0) : 1
Select difficulty (0=Easy, 1=Medium, 2=Hard): 1
```

2. **Choose your position**

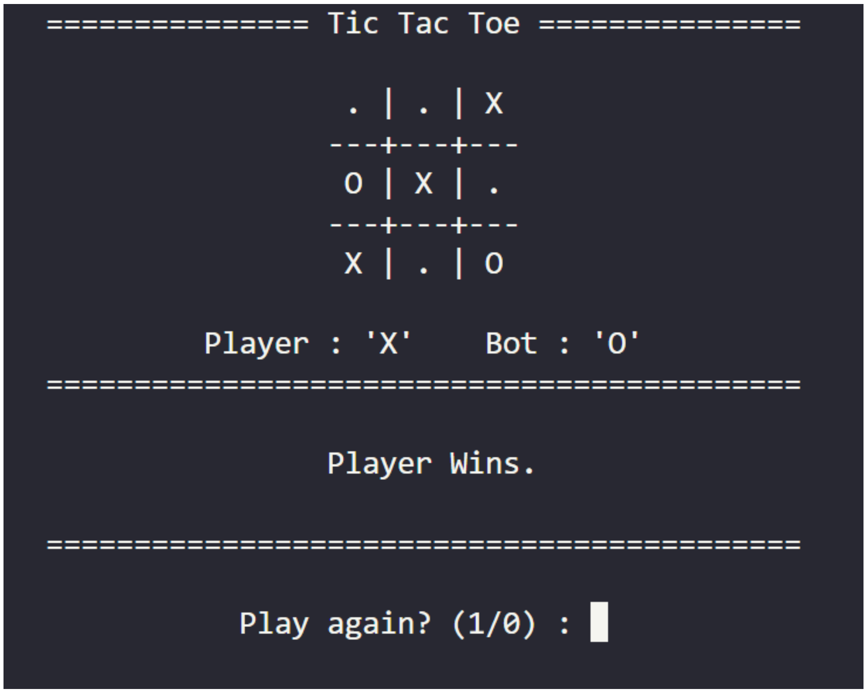


```
=============== Tic Tac Toe ===============

                 . | . | .
                ---+---+---
                 O | X | .
                ---+---+---
                 . | . | .

         Player : 'X'    Bot : 'O'
===========================================

             Player your move:
                    0 2
```



```
=============== Tic Tac Toe ===============

                 . | . | X
                ---+---+---
                 O | X | .
                ---+---+---
                 . | . | O

         Player : 'X'    Bot : 'O'
===========================================

             Player your move:
                    2 0
```

3. **Player wins**



```
=============== Tic Tac Toe ===============

                 . | . | X
                ---+---+---
                 O | X | .
                ---+---+---
                 X | . | O

         Player : 'X'    Bot : 'O'
===========================================

               Player Wins.

===========================================

             Play again? (1/0) :
```

ANALYSIS
Code analysis:

init()

- Purpose: Initializes the game board to '.' (empty).
- Design: Loops over a 3x3 matrix, setting each cell.
- Complexity: O(1) (since it's a constant 3x3 grid).
- Improvement: None needed; it's simple and effective.

mainScreen()

- Purpose: Displays the main menu with a simple header.
- Design: Prints static text using printf.
- Complexity: O(1).
- Improvement: Could include dynamic elements (e.g., showing last scores), but for a console game, this is fine.

gameBoard(const char* P1, const char* P2)

- Purpose: Renders the current game board and player symbols.
- Design: Prints each row of the board and the player markers.
- Complexity: O(1).
- Improvement: Consider clearing the screen more cleanly (e.g., using platform-specific methods instead of system("cls")).

char result()

- Purpose: Checks for a win condition (rows, columns, diagonals) or a draw.
- Design:
  - Scans rows and columns for three in a row.
  - Checks two diagonals.
  - If no win, checks for any remaining '.'.
  - Returns 'X', 'O', 'T' (tie), or '-' (continue).
- Complexity: O(1).
- Improvement: Solid as-is; for a 3x3 board, this exhaustive check is acceptable.

bool isMovesLeft()

- Purpose: Checks if any moves are left on the board.
- Design: Scans the board for any '.'.
- Complexity: O(1).
- Improvement: Could integrate this check into result() to avoid redundant loops.

int minimax(bool isBotTurn)

- Purpose: Recursive Minimax algorithm for the hard bot.
- Design:
  - Scores moves as +10 for bot win, -10 for player win, 0 for draw.
  - Recursively explores all possible future moves.
  - Chooses best score for bot and worst for player.
- Complexity: Exponential (O(b^d) where b=9 branches, d=number of empty cells).
- Improvement: Could add alpha-beta pruning to reduce unnecessary branches.

void botMoveHard()

- Purpose: Executes the best move for the bot using Minimax.
- Design:
  - Iterates over all possible moves.
  - Calls minimax to evaluate each move.
  - Selects the move with the best score.
- Complexity: O(b^d), bottlenecked by minimax.
- Improvement: Consider memoization or pruning to speed up.

void botMoveEasy()

- Purpose: Picks a random move for the bot.
- Design: Randomly selects coordinates until an empty cell is found.
- Complexity: O(1) in the best case; O(n) in the worst (if the board is almost full).
- Improvement: Could scan the board once for available moves and select randomly from them for efficiency.

bool tryMove(char player, int* moveX, int* moveY)

- Purpose: Checks if the player can win immediately by placing a marker.
- Design:
  - Tries each empty cell, placing a marker temporarily.
  - Checks if it results in a win.
  - Returns the first found winning move.
- Complexity: O(1).
- Improvement: Efficient for a small board. Could be expanded for larger boards or complex win conditions.

void botMoveMedium()

- Purpose: Medium difficulty bot:
  1. Tries to win.
  2. Blocks the player’s win.
  3. Otherwise, makes a random move.
- Design: Combines tryMove and botMoveEasy.
- Complexity: O(1) since the board is small.
- Improvement: Could prioritize center or corner cells for slightly smarter play.

int main()

- Purpose: The main loop managing:
  - Game initialization.
  - User input (first move, difficulty).
  - Game loop: alternating turns, rendering the board, checking for a result.
  - Replay prompt.

- Design:
  - Uses system("cls") to clear the console screen.
  - Alternates between player and bot based on cnt.
- Complexity: Linear in the number of moves (up to 9).
- Improvement:
  - Replace system("cls") with platform-independent clear screen methods.
  - Validate user input more strictly.
  - Add error handling for non-numeric inputs.

Overall analysis:

Strengths

(+) The game flow is smooth, with clear prompts and a clean display of the board.

(+) Multiple difficulty levels allow players of varying skill levels to enjoy the game.

(+) Hard bot is unbeatable due to the optimal Minimax implementation.

(+) Randomness in easy and medium modes makes gameplay unpredictable.

Weaknesses

(-) No move validation feedback, invalid player moves are silently ignored (e.g., selecting an occupied cell).

(-) No adaptive strategy for bot, the bot plays strictly based on difficulty and does not adapt during the game.

(-) Hardcoded console interface, the user interface is not graphical and limited to the console.

Potential Improvements

- Add error messages for invalid player moves.
- Implement a graphical interface (e.g., using SDL, SFML, or a web-based GUI).
- Include a score tracker across multiple rounds.
- Allow PvP mode (Player vs Player).

Github Repository:

https://github.com/Dancingaroundthelies/EF234405_DAA_Q2_5025211252_Nur-Azizah_5025221204_Andina-Safitri-Innayah.git

“In the name of Allah (God) Almighty, I hereby pledge and sincerely declare that I have completed Quiz 2 independently. I have not committed any form of cheating, plagiarism, or received unauthorized assistance. I accept all consequences should it be proven that I have engaged in cheating and/or plagiarism.”

Surabaya, 29 Mei 2025

Aulia Daffa Rahmani	Andina Safitri Innayah	Nur Azizah
5025221205	5025221204	5025211252