QUIZ 2 REPORT

Muhammad Daffa Ashdaqfillah (5025211015)
Salsabila Fatma Aripa (5025211057)
Yusna Millaturrosyidah (5025211254)

DESIGN & ANALYSIS OF ALGORITHM
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2023

About :

This document is the documentation for the **Tic Tac Toe Project**, which is part of the **Quiz 2 Project** for the **Design and Analysis of Algorithms Course**.

- Leveling :
    - **Level 1: Two Player Game**

      In this level, two players take turns placing X's and O's on the game board until someone wins or the game ends in a draw. The game board is displayed on the screen, and players input their moves via the command line.

    - **Level 2: Player vs Bot**

      In this level, the player goes up against a computer-controlled opponent. The player can choose to go first or second. If the player goes first, they input their moves via the command line as in Level 1. If the player goes second, the computer will make the first move, and subsequent moves are made by the player and the computer in turn.

    - **Level 3: Player vs AI (Unbeatable)**

      This level introduces an AI opponent that is more difficult to beat than the bot in Level 3. The AI opponent uses an algorithm called minimax to determine the best move to make based on the current state of the game board.

- How to play :

  To play the game, select the desired level from the main menu. In Levels 1, players input their moves via the command line. In Levels 2 and 3, the game board is displayed on the screen, and players input their moves using the keyboard.

  In Level 2 and 3, the player can choose to go first or second.

  The game ends when a player wins or the game ends in a draw. Players can choose to play again or exit the game after each match.

Design :

Tic Tac Toe is a two player game where they take turns placing their symbols (X or O) on the available squares on a 3x3 game board. The goal is to reach a row, column, or diagonal consisting of three of the same symbol. In implementing this Tic Tac Toe game, we use the C++ programming language. There are several functions that we created for this game, including the functions `mainScreen()`, `gameBoard()`, `init()`, `result()`, `minimax()`, `bestMove()`, dan ` main()`.

- The `mainScreen()` function displays the main screen of the Tic Tac Toe game to the user. Here is the explanation of the function:
    1. Displays the game title "Tic Tac Toe" in the middle of the screen.
    2. Displays a selection of available game levels:
        - Level 1: Two player game. Users will play against other players.
        - Level 2: Game against bots. Users will play against bots whose steps are randomly generated.
        - Level 3: Game against AI. Users will play against an AI that uses the minimax algorithm to make the best move decisions.
    3. Provides blank line spacing to separate level selections and other parts of the screen.
    4. Displays a horizontal line as a separation between the level selection and the rest of the screen.

By displaying this information, the `mainScreen()` function gives the user an overview of the game levels available in Tic Tac Toe. Users can select the desired game level to start playing.

- The `gameBoard()` function is used to display the Tic Tac Toe game board to the screen, including the position of the 'X' and 'O' marks that have been selected by the player. Here is the explanation of the function:
    1. Displays the title "Tic Tac Toe" in the middle of the screen.
    2. Displays the positions of the 'X' and 'O' marks on the game board as a 3x3 matrix:
        - Row 1: Column 1, Column 2, Column 3
        - Row 2: Column 1, Column 2, Column 3
        - Row 3: Column 1, Column 2, Column 3

        Each matrix element represents the mark it is in that position on the game board.
    3. Displays the horizontal separator between the game board rows.
    4. Displays the information of the players involved in the game:
        - Name P1: 'X'
        - Name P2: 'O'
    5. Provides blank line spacing to separate parts of the game board from the rest of the screen.
    6. Displays a horizontal line as a separation between the game board and the rest of the screen.

By displaying this information, the gameBoard() function shows the user the position of the marks on the game board, as well as showing the players acting as 'X' and 'O'. Users can view the game status and continue the game by entering their steps into the boxes provided on the game board.

- The `init()` function is used to initialize the Tic Tac Toe game board with a dot ('.') character. Here is the explanation of the function:
    1. Iterate through the rows and columns of the game board using a for loop.
    2. At each iteration, set the value of the elements in row i and column j position on the game board (arr) to be a dot character ('.').
    3. After the loop has finished executing, the game board is filled with a dot character ('.') in each square, indicating that the square is empty and no marks have been placed in it.

By initializing it this way, the init() function prepares the game board empty before the game starts. This allows the player to start the game with a clean game board and without any marks.

- The `result()` function is used to check the result of playing Tic Tac Toe on the game board (arr). Here is the explanation of the function:
    1. Perform row checks: Iterate through each row on the game board.
        - If all three squares in the same row have the same value (X or O) and are not a period character ('.'), then returns that character's value as a winner.
    2. Perform column checks: Iterate through each column on the game board.
        - If all three squares in the same column have the same value (X or O) and are not a period character ('.'), then returns that character's value as a winner.
    3. Performs main diagonal check: Checks whether the three squares on the main diagonal have the same value (X or O) and not a dot character ('.').
        - If yes, returns that character's value as a winner.
    4. Performs secondary diagonal check: Checks whether three squares on the secondary diagonal have the same value (X or O) and not a dot character ('.').
        - If yes, returns that character's value as a winner.
    5. Checking to determine the draw:
        - Iterate through all the squares on the game board.
        - If there is at least one empty box (worth the dot character '.'), then change the check variable to false.
        - If all boxes are filled in (no dot character '.'), then return the character 'T' which indicates a draw.
    6. If there is no winner or draw, returns a '-' character indicating that the game is still in progress.

By checking like this, the result() function determines the game result based on the current state of the game board.

- The `minimax()` function is an implementation of the Minimax algorithm in the Tic Tac Toe game. Here is the explanation of the code:
    1. Check the current game result by calling the result() function:
        - If the result of the game is player 'O' winning, returns a positive score of 1.
        - If the result of the game is player 'X' winning, returns a negative score of -1.
        - If the result of the game is a draw, returns a score of 0.
    2. If the player currently running is player 'O' (isMaximizing = true):
        - Initialize the bestScore variable with a value of -1000 as a very low initial score.
        - Iterate through each square on the game board.
        - If the box is empty (worth the dot character '.'), simulate the move of player 'O' on the box:
            - Defines the box with the character 'O'.
            - Recursively calls the minimax() function with depth increments (depth+1) and switches the current player to the opposing player (isMaximizing = false).
            - Returns the score from the recursive call.
            - Returns the best score (max value) of all possible moves of player 'O'.
    3. If the player currently running is player 'X' (isMaximizing = false):
        - Initialize the bestScore variable with a value of 1000 as a very high initial score.
        - Iterate through each square on the game board.
        - If the square is empty (worth the dot character '.'), simulate player 'X' move on the square:
            - Defines the box with the character 'X'.
            - Recursively calls the minimax() function with increased depth (depth+1) and switches the current player to the opposing player (isMaximizing = true).
            - Returns the score from the recursive call.
            - Returns the best score (min value) of all possible moves of player 'X'.

Using the Minimax algorithm, the `minimax()` function tries all possible moves for each player in turn and chooses the best move based on the resulting score.

- The `bestMove()` function is used to determine the best move to be taken by player 'O' using the Minimax algorithm. Here is the explanation of the code:

    1. Initialize the bestScore variable with a very low initial value (-1000).
    2. Iterate through each square on the game board.
    3. If the box is empty (worth a dot character '.'), then:
        - Simulate player 'O' move by assigning the square with character 'O'.

- Calls the minimax() function with an initial depth of 0 and sets isMaximizing to false.
- Returns the score from the minimax() call.
- Restore the box to its original state by setting the box back to the dot '.' character.
- If the score obtained is greater than the bestScore, update the bestScore with that score and note the position of the (x, y) box.

4. Set squares with position (x,y) with character 'O', which is the best move for player 'O' to take.

Using the Minimax algorithm, the bestMove() function tries all possible moves of player 'O' by calling minimax() for each possible move, and then selecting the move with the highest score for player 'O'.

- The `main()` function: Sets the flow of the game. Contains a main loop that allows the user to play repeatedly. In each iteration, level selection, initialization of the game board, processing of player and AI/Bot steps, and checking of game results are carried out.

Through the use of a control structure and looping, this code allows players to play Tic Tac Toe against other players, random bots, or AI that uses the minimax algorithm. The user is given the option to play again or exit the program after each game.

Using this approach, players can interact with the game of Tic Tac Toe and enjoy the experience of playing against other players or dealing with intelligent AI strategies.

Implementation :

a. Minimax Algorithm :

The minimax algorithm is a well-known algorithm used in game theory and decision-making problems. It is a recursive function that simulates all possible moves that both the AI and the player can make, and assigns a score to each move based on the likelihood of winning the game. The function then chooses the move with the highest score for the AI.

In Tic Tac Toe, the algorithm is used in Level 3 to determine the best move for the AI opponent. The algorithm starts by checking if the current state of the game board is a winning state for either the AI or the player. If it is, it returns a score of 1 for a win, -1 for a loss, and 0 for a tie.

If the game is not in a winning state, the algorithm creates a tree of all possible moves that the AI and the player can make. The algorithm then recursively calls itself on each of the possible moves, alternating between maximizing the score for the AI and minimizing the score for the player. The score for each move is assigned based on the result of the recursive call, and the move with the highest score is chosen for the AI.

The `minimax()` function in the source code is the implementation of the minimax algorithm. It takes two arguments: depth, which represents the current depth of the game tree, and isMaximizingPlayer, which is a boolean value that represents whether the current player is the AI or the opponent.

The function first checks if the game is in a winning state, by calling the `result()` function. If the game is in a winning state, the function returns a score of 1 for a win, -1 for a loss, and 0 for a tie.

If the game is not in a winning state, the function creates a loop that iterates through all possible moves that can be made on the game board. For each possible move, the function simulates the move by updating the game board and calling itself recursively with the depth parameter increased by 1 and the isMaximizingPlayer parameter negated. The score for each move is then assigned based on the result of the recursive call.

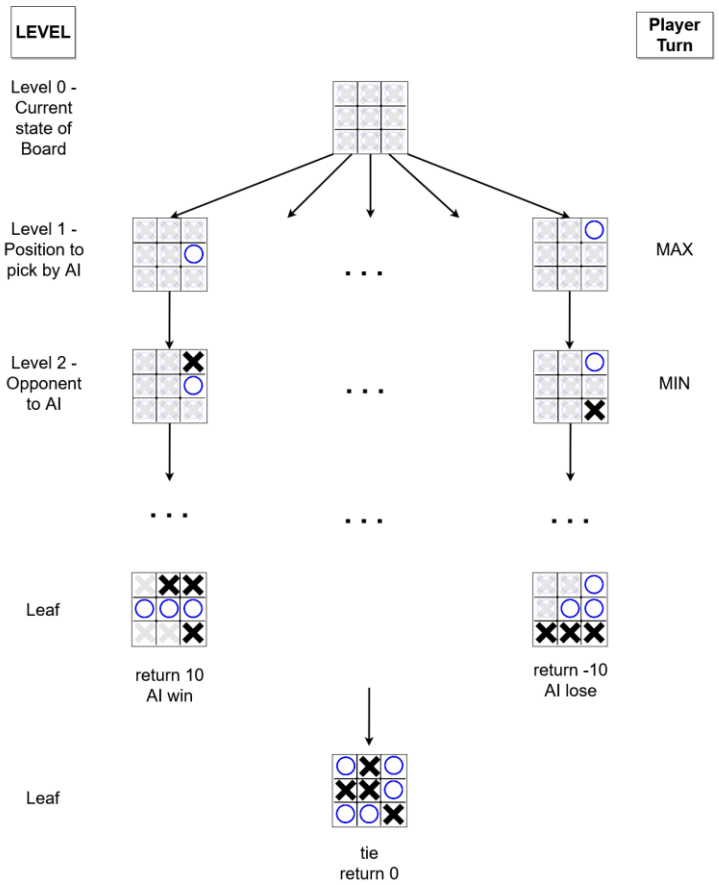
If the isMaximizingPlayer parameter is true, the function chooses the move with the highest score. If the parameter is false, the function chooses the move with the lowest score. The function then undoes the move that was made on the game board and returns the chosen score.

The minimax algorithm uses backtracking to simulate the game board and undo moves that were made during the recursion. This allows it to evaluate all possible moves and determine the best one for the AI to make.

One important aspect of the minimax algorithm is pruning, which is the process of removing nodes from the game tree that are unlikely to lead to the optimal solution. This is done by using alpha-beta pruning, which is a technique that allows the algorithm to skip over branches of the tree that can be proven to be irrelevant to the final result.

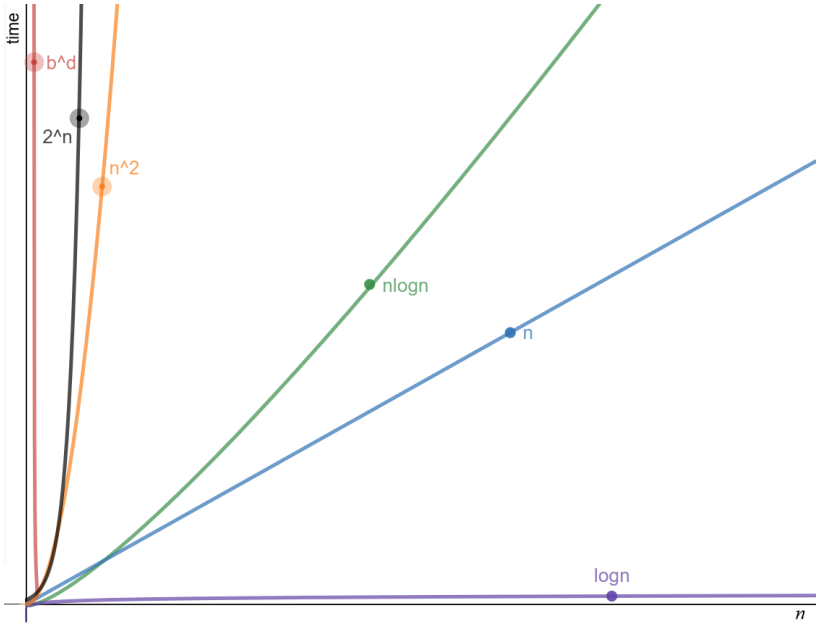
The players are trying to maximize or minimize the others result: The AI is the "Maximizer" who tries to get the highest result per level The opponent is the "Minimizer" who tries to get the lowest result per level.

Tic Tac Toe minimax game illustration :



Complexity Time :

As we traverse all levels of a tree and at each level, we need to generate all permutations of each node using DFS (Depth-First Search), we can conclude that the worst-case scenario is the only case that runs with a complexity of O(V+E) or O(b^d) for DFS (i.e., exploring the entire tree). Basically, the time complexity of the search using the Minimax Algorithm is the same as that of DFS.



b. Random value by time :

In level 2 of the game, where the player is playing against the bot, we use the rand() function to obtain random numbers. However, there is an adjustment to ensure that the generated numbers are truly different each time it is called.

srand(time(0)) is used before calling the rand() function to initialize the random number generator. This is done to ensure that each time the program is run, the random numbers generated by rand() will be different.

The rand() function itself uses an algorithm to generate a sequence of random numbers based on a specific seed value. This seed is the initial value used to start the algorithm. **If the seed is the same, the sequence of random numbers generated will be the same in every program execution**.

By using srand(time(0)), the seed of rand() is set based on the current time. Since time keeps changing, the seed will also change each time the program is run. This produces different sequences of random numbers in each program execution, providing better variation in the random number selection.

In the context of this game, rand() is used to select random positions on the game board. By using srand(time(0)), the selection of random positions will be more random and unpredictable each time the program is run.

Analysis :

a. Libraries Used

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

Explanation :

`Header file <stdio.h>` : This header file provides standard input/output functions such as printf() and scanf(), which are used in the program to display messages on the screen and receive input from the user.

`Header file <time.h>` : This header file provides functions for manipulating time and date. In this program, this header file is used to generate random values when the computer player (Bot or AI) needs to make a move.

`Header file <stdbool.h>` : This header file introduces the boolean data type (bool) and boolean constants (true and false). The boolean data type is used in this program to store truth values (true or false), which are used in logical conditions and program flow control.

`Header file <stdlib.h>` : This header file provides general utility functions such as dynamic memory allocation (malloc() and free()) and other utility functions. In this program, this header file is used to use functions like rand() to generate random numbers.

`Header file <string.h>` : This header file provides functions for string manipulation, such as strcpy() and strlen(). In this program, this header file is not directly used, so it may not be necessary in the context of this program.

b. char arr[3][3] Function

```
char arr[3][3]= {'.','.','.',
                 '.','.','.',
                 '.','.','.'};
```

Explanation :

The line of code

`char arr[3][3]= {'.','.','.', '.','.','.', '.','.','.'};` initializes a 2-dimensional character array named arr with a size of 3x3. The array is filled with the characters '.', representing empty spaces or placeholders.

This array is used as the game board for the Tic Tac Toe game. Each element in the array corresponds to a cell on the game board. The initial values of all elements are set to '.', indicating that the cells are initially empty.

As the game progresses, the elements in the arr array will be updated to hold the symbols 'X' or 'O' to represent the moves made by the players.

c. void mainScreen() Function

```
void mainScreen(){
    printf("================== Tic Tac Toe ==================\n\n");
    printf("\tLevel 1 : Two Player Game\n");
    printf("\tLevel 2 : Player vs Bot\n");
    printf("\tLevel 3 : Player vs AI\n\n");
    printf("================================================\n");
}
```

Explanation :

The function `**void mainScreen()**` is defined to display the main screen or menu of the Tic Tac Toe game. The main screen provides a clear and concise presentation of the game options, allowing the player to choose the desired level of gameplay.

d. void gameBoard(char* P1, char* P2) Function

```
void gameBoard(char* P1, 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");
}
```

Explanation :

The function `**void gameBoard(char* P1, char* P2)**` is defined to display the current state of the Tic Tac Toe game board. The elements of the arr array are used to populate the game board. Each element represents a cell on the board, and its corresponding value is printed.

The players' names and symbols are displayed below the game board. P1 corresponds to the first player's name, and P2 corresponds to the second player's name. The symbols 'X' and 'O' represent the moves made by the respective players.

By calling this function during the game, the current state of the game board is visually presented to the players, allowing them to see the positions of their moves and the overall progress of the game.

e. void init() Function

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

Explanation :

The function `**void init()**` is defined to initialize or reset the Tic Tac Toe game board to its initial state. It sets all the elements of the arr array to the character '.', indicating empty spaces or placeholders.

The function achieves this by using nested loops to iterate through each element of the 2-dimensional arr array. The outer loop iterates over the rows, and the inner loop iterates over the columns. For each element, the assignment statement arr[i][j]='.' is used to set its value to '.'.

After executing this function, the game board will be reset to its original state, where all the cells are empty. This allows for a fresh start or a new game to be played on the board.

f. char result() Function

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

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

   if (arr[0][0] == arr[1][1] && arr[1][1] == arr[2][2] && arr[2][2]!='.'){
      return arr[0][0];
   }
```

```
  if (arr[0][2] == arr[1][1] && arr[1][1] == arr[2][0] && arr[2][0]!='.'){
     return arr[0][2];
  }

  bool cek = true;
  for(int i=0; i<3; i++){
    for(int j=0; j<3; j++){
      if(arr[i][j] == '.') {
        cek = false;
        break;
      }
    }
  }

  if(cek) return 'T';
  return '-';
}
```

Explanation:
The function **char result()** is defined to determine the current result or outcome of the Tic Tac Toe game. It analyzes the game board represented by the arr array and checks for winning conditions or a draw.

The function first checks for a winning condition in each row by comparing the values of 3 consecutive elements in the same row yakni (0, 1, 2). It iterates through each row using a for loop and checks if the three elements are equal to each other and not equal to '.', indicating that a player has made a move in that row. If a winning condition is found, the corresponding symbol is returned.

Next, the function checks for a winning condition in each column by comparing the values of three consecutive elements in the same column. It uses a similar logic as the row check but iterates through each column using a for loop. If a winning condition is found, the corresponding symbol is returned.

The function then checks for a winning condition in the diagonal from the top-left to the bottom-right. It compares the values of the three elements at positions (0, 0), (1, 1), and (2, 2). If a winning condition is found, the corresponding symbol is returned.

Finally, the function checks for a winning condition in the diagonal from the top-right to the bottom-left. It compares the values of the three elements at positions (0, 2), (1, 1), and (2, 0). If a winning condition is found, the corresponding symbol is returned.

If no winning condition is found after the previous checks, the function proceeds to check if the game board is full, indicating a draw. It uses nested loops to iterate through each element of the arr array. If an empty cell (denoted by '.') is found, the boolean variable cek is set to false, indicating that the board is not full. If the board is full, meaning all cells are filled, the function returns 'T' to represent a draw.

If none of the above conditions are met, the function returns '-', indicating that the game is still ongoing.

g. int minimax(int depth, bool isMaximizing) Function

```
int minimax(int depth, bool isMaximizing){

  int score;
  char res = result();
  if (res=='O') {
     score = 1;
     return score;
  }
  else if (res=='X') {
     score = -1;
     return score;
  }
  else if (res=='T') {
     score = 0;
     return score;
  }

  if(isMaximizing){
     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';
              score = minimax(depth+1, 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';
                    score = minimax(depth+1, true);
                    arr[i][j]='.';
                    if(score < bestScore){
                        bestScore = score;
                    }
                }
            }
        }
        return bestScore;
    }
}
```

Explanation :
The function **int minimax(int depth, bool isMaximizing)** implements the minimax algorithm, which is a recursive algorithm used to determine the optimal move in a game with perfect information. This function uses the minimax algorithm to recursively evaluate all possible moves and assign scores, ultimately returning the best score for the maximizing player or the worst score for the minimizing player at the current game state.

The function takes two parameters: depth, which represents the current depth in the game tree, and isMaximizing, a boolean flag indicating whether it is the maximizing player's turn or not.

The function first checks the result of the game by calling the result() function. If the result is 'O', it means the maximizing player (represented by 'O') has won, so the function assigns a score of 1 and returns it. If the result is 'X', it means the minimizing player (represented by 'X') has won, so the function assigns a score of -1 and returns it. If the result is 'T', it means the game is a draw, so the function assigns a score of 0 and returns it.

If none of the above conditions are met, the function proceeds to evaluate the possible moves.

If it is the maximizing player's turn (isMaximizing is true), the function initializes bestScore as a very low value (-1000) to track the highest score found. It then iterates through each cell of the game board represented by the arr array. If a cell is empty (denoted by '.'), it makes a move by assigning 'O' to that cell, and then recursively calls the minimax function with depth+1 and isMaximizing set to false to evaluate the opponent's turn.

After the recursive call, the move is undone by resetting the cell to '.'. The returned score is compared with bestScore, and if it is greater, bestScore is updated to the new higher score.

Finally, after iterating through all possible moves, the function returns bestScore, representing the highest possible score for the maximizing player.

If it is the minimizing player's turn (isMaximizing is false), the function follows a similar logic as above, but instead, it initializes bestScore as a very high value (1000) to track the lowest score found. It aims to find the move with the lowest possible score for the minimizing player.

The function iterates through each cell, makes a move with 'X', recursively calls the minimax function with depth+1 and isMaximizing set to true to evaluate the opponent's turn, undoes the move, and updates bestScore if the returned score is lower.

After evaluating all possible moves, the function returns bestScore, representing the lowest possible score for the minimizing player.

h. void bestMove() Function

```
void bestMove(){
  int score;
  int bestScore = -1000, x, y;
  for(int i=0; i<3; i++){
    for(int j=0; j<3; j++){
      if(arr[i][j]=='.'){
        arr[i][j]='O';
        score = minimax(0, false);
        arr[i][j]='.';
        if(score > bestScore){
          bestScore = score;
          x=i; y=j;
        }
      }
    }
  }
  arr[x][y]='O';
}
```

Explanation :

The function **void bestMove()** is used to determine the best possible move for the AI player ('O') in the game. The bestMove() function utilizes the minimax() function to evaluate and select the best move for the AI player ('O') by iterating through all possible moves, assigning scores, and choosing the move with the highest score.

The function starts by declaring and initializing variables score, bestScore, x, and y. bestScore is set to a very low value (-1000) to track the highest score found, while x and y will store the coordinates of the best move.

Next, the function iterates through each cell of the game board represented by the arr array using nested loops. It checks if a cell is empty (denoted by '.'). If the cell is empty, it temporarily assigns 'O' to that cell, indicating the AI player's move.

After making the move, the function calls the minimax() function with a depth of 0 and isMaximizing set to false to evaluate the opponent's turn and obtain a score for the current move.

Once the score is obtained, the move is undone by resetting the cell to '.'. The function then compares the obtained score with the current bestScore. If the score is greater than bestScore, it updates bestScore to the new higher score and records the coordinates of the current move in x and y.

After evaluating all possible moves, the function assigns 'O' to the cell with the best move by using the coordinates stored in x and y. This represents the optimal move for the AI player.

i. main() Function

```
int main (){
   while(1){
      system("cls");
      mainScreen();
      int level;
      char res='-';
      printf("\n\t      Select Level : ");
      init();
      scanf("%d", &level);

      int player=1, x, y, cnt=0;
      if(level==1){

         while(1){
            system("cls");
            gameBoard("Player 1", "Player 2");
            printf("\n\t    Player %d your move: \n\t\t    ", player);
            scanf("%d %d", &x, &y);

            if(arr[x][y]=='X' || arr[x][y]=='O' || x<0 || x>2 || y<0 || y>2) continue;

            if(player==2) arr[x][y] = 'O';
```

```
            else arr[x][y] = 'X';

            cnt++;
            res = result();
            if(res!='-') {
                system("cls");
                gameBoard("Player 1", "Player 2");
                if(res == 'O')          printf("\n\t\tPlayer 2 Win.\n");
                else if (res == 'X')    printf("\n\t\tPlayer 1 Win.\n");
                else if (res == 'T')    printf("\n\t\t   Draw.\n");
                break;
            }

            player++;
            player = (player % 2) ? 1 : 2;
        }
    }

    if(level==2){
        int turn;
        system("cls");
        printf("Bot go first? (1/0) : ");
        scanf("%d", &turn);

        if(turn) cnt++;
        while(1){
            system("cls");
            gameBoard("Player", "Bot");
            if(cnt%2==0){
                printf("\n\t     Player your move: \n\t\t    ");
                scanf("%d %d", &x, &y);
                if(arr[x][y]=='X' || arr[x][y]=='O' || x<0 || x>2 || y<0 || y>2) continue;
                arr[x][y] = 'X';
                cnt++;
            }
            else {

                printf("\nBot\n");
                srand(time(0));
                while(1){
                    x = rand()%3;
                    y = rand()%3;
```

```
                if(arr[x][y]=='.') break;
            }
            arr[x][y] = 'O';
            cnt++;
        }
        res = result();
        if(res!='-') {
            system("cls");
            gameBoard("Player", "Bot");
            if(res == 'O')          printf("\n\t\t  Bot Win.\n");
            else if (res == 'X')    printf("\n\t\tPlayer Win.\n");
            else if (res == 'T')    printf("\n\t\t   Draw.\n");
            break;
        }

    }
}

if(level==3){
    int turn;
    system("cls");
    printf("AI go first? (1/0) : ");
    scanf("%d", &turn);

    if(turn) cnt++;
    while(1){
        system("cls");
        gameBoard("Player", "AI");
        if(cnt%2==0){
            printf("\n\t     Player your move: \n\t\t    ");
            scanf("%d %d", &x, &y);
            if(arr[x][y]=='X' || arr[x][y]=='O' || x<0 || x>2 || y<0 || y>2) continue;
            arr[x][y] = 'X';
            cnt++;
        }
        else {
            printf("\nBot\n");
            bestMove();
            cnt++;
        }

        res = result();
```

```
            if(res!='-') {
                system("cls");
                gameBoard("Player", "AI");
                if(res == 'O')          printf("\n\t\t  AI Win.\n");
                else if (res == 'X')    printf("\n\t\tPlayer Win.\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) return 0;
  }
}
```

Explanation :

The int main() function serves as the central component of the Tic Tac Toe game program. It encompasses the main gameplay logic and controls the flow of the game. The function starts by setting up an infinite while loop, ensuring that the game continues until the player decides to exit.

Inside the loop, the console screen is cleared, and the main menu is displayed using the mainScreen() function. The player is then prompted to choose a level of gameplay, and the game board is initialized by calling the init() function, which sets all the cells to '.'.

The subsequent code handles the chosen level differently. For level 1, a two-player game, the loop allows players to take turns entering their moves until the game reaches a conclusion. The game board is displayed using the gameBoard() function, and the players' moves are validated and placed on the board. After each move, the game result is checked using the result() function, and if the game is over, the corresponding message is displayed.

For level 2, player vs. bot, the loop alternates between the player and the bot. The player's move is obtained through input, while the bot's move is randomly generated using the rand() function. The game result is checked after each move, and if the game is over, the appropriate message is displayed.

Level 3, player vs. AI, follows a similar structure to level 2, but instead of random moves for the AI, the bestMove() function is called to determine the optimal move. Again, the game result is checked, and if the game is over, the corresponding message is displayed.

After the game ends, the player is given the option to play again. If they choose to continue, the loop restarts, and if not, the program terminates. This structure ensures that the game remains interactive and allows for multiple rounds of gameplay.

Source Code :

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

char arr[3][3]= {'.','.','.',
                 '.','.','.',
                 '.','.','.'};

void mainScreen(){
   printf("================== Tic Tac Toe ==================\n\n");
   printf("\tLevel 1 : Two Player Game\n");
   printf("\tLevel 2 : Player vs Bot\n");
   printf("\tLevel 3 : Player vs AI\n\n");
   printf("================================================\n");
}

void gameBoard(char* P1, 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");
}

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

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

```
            return arr[i][0];
        }
    }

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

    if (arr[0][0] == arr[1][1] && arr[1][1] == arr[2][2] && arr[2][2]!='.'){
        return arr[0][0];
    }

    if (arr[0][2] == arr[1][1] && arr[1][1] == arr[2][0] && arr[2][0]!='.'){
        return arr[0][2];
    }

    bool cek = true;
    for(int i=0; i<3; i++){
      for(int j=0; j<3; j++){
        if(arr[i][j] == '.') {
          cek = false;
          break;
        }
      }
    }

    if(cek) return 'T';
    return '-';
}

int minimax(int depth, bool isMaximizing){

    int score;
    char res = result();
    if (res=='O') {
        score = 1;
        return score;
    }
    else if (res=='X') {
        score = -1;
```

```
        return score;
    }
    else if (res=='T') {
        score = 0;
        return score;
    }

    if(isMaximizing){
        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';
                    score = minimax(depth+1, 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';
                    score = minimax(depth+1, true);
                    arr[i][j]='.';
                    if(score < bestScore){
                        bestScore = score;
                    }
                }
            }
        }
        return bestScore;
    }
}

void bestMove(){
    int score;
```

```
    int bestScore = -1000, x, y;
    for(int i=0; i<3; i++){
      for(int j=0; j<3; j++){
        if(arr[i][j]=='.'){
          arr[i][j]='O';
          score = minimax(0, false);
          arr[i][j]='.';
          if(score > bestScore){
             bestScore = score;
             x=i; y=j;
          }
        }
      }
    }
    arr[x][y]='O';
}

int main (){
    while(1){
       system("cls");
       mainScreen();
       int level;
       char res='-';
       printf("\n\t      Select Level : ");
       init();
       scanf("%d", &level);

       int player=1, x, y, cnt=0;
       if(level==1){

          while(1){
             system("cls");
             gameBoard("Player 1", "Player 2");
             printf("\n\t    Player %d your move: \n\t\t    ", player);
             scanf("%d %d", &x, &y);

             if(arr[x][y]=='X' || arr[x][y]=='O' || x<0 || x>2 || y<0 || y>2) continue;

             if(player==2) arr[x][y] = 'O';
             else arr[x][y] = 'X';

             cnt++;
```

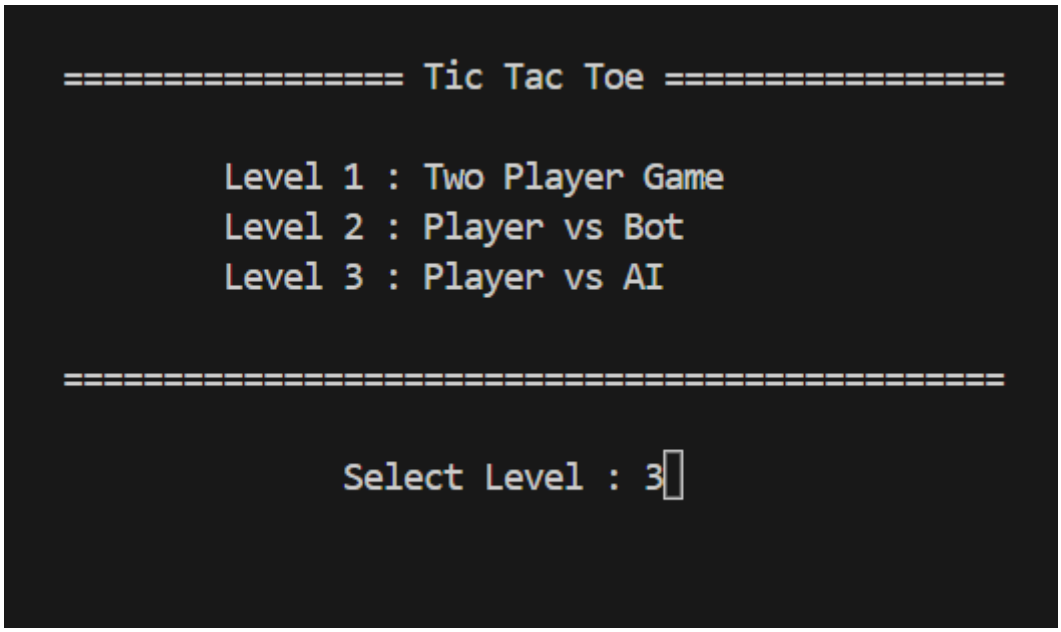
```
        res = result();
        if(res!='-') {
            system("cls");
            gameBoard("Player 1", "Player 2");
            if(res == 'O')          printf("\n\t\tPlayer 2 Win.\n");
            else if (res == 'X')    printf("\n\t\tPlayer 1 Win.\n");
            else if (res == 'T')    printf("\n\t\t   Draw.\n");
            break;
        }

        player++;
        player = (player % 2) ? 1 : 2;
    }
}

if(level==2){
    int turn;
    system("cls");
    printf("Bot go first? (1/0) : ");
    scanf("%d", &turn);

    if(turn) cnt++;
    while(1){
        system("cls");
        gameBoard("Player", "Bot");
        if(cnt%2==0){
            printf("\n\t     Player your move: \n\t\t    ");
            scanf("%d %d", &x, &y);
            if(arr[x][y]=='X' || arr[x][y]=='O' || x<0 || x>2 || y<0 || y>2) continue;
            arr[x][y] = 'X';
            cnt++;
        }
        else {

            printf("\nBot\n");
            srand(time(0));
            while(1){
                x = rand()%3;
                y = rand()%3;
                if(arr[x][y]=='.') break;
            }
            arr[x][y] = 'O';
```

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

    }
}

if(level==3){
    int turn;
    system("cls");
    printf("AI go first? (1/0) : ");
    scanf("%d", &turn);

    if(turn) cnt++;
    while(1){
        system("cls");
        gameBoard("Player", "AI");
        if(cnt%2==0){
            printf("\n\t     Player your move: \n\t\t    ");
            scanf("%d %d", &x, &y);
            if(arr[x][y]=='X' || arr[x][y]=='O' || x<0 || x>2 || y<0 || y>2) continue;
            arr[x][y] = 'X';
            cnt++;
        }
        else {
            printf("\nBot\n");
            bestMove();
            cnt++;
        }

        res = result();
        if(res!='-') {
            system("cls");
            gameBoard("Player", "AI");
```

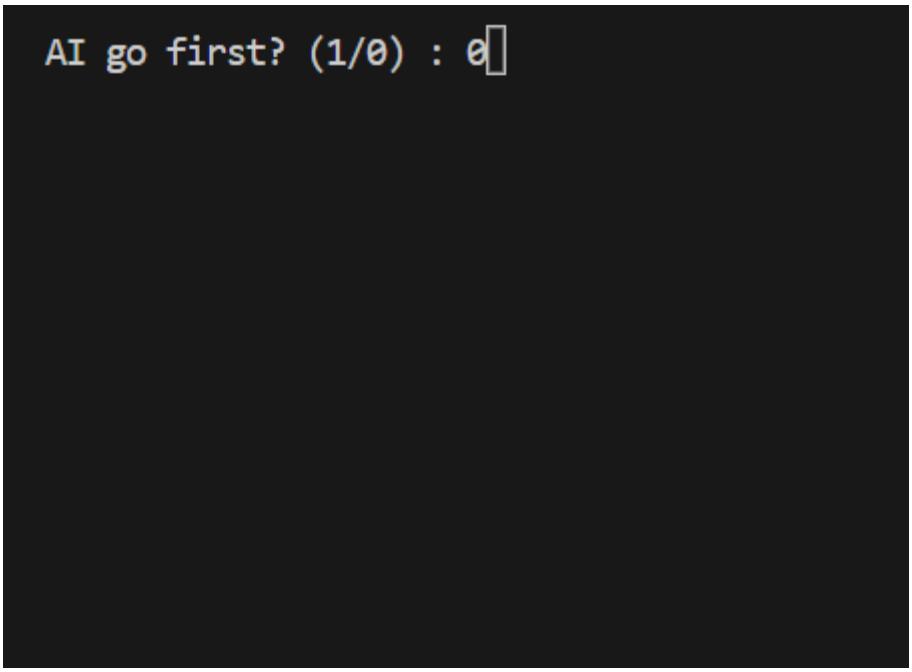
```
            if(res == 'O')          printf("\n\t\t  AI Win.\n");
            else if (res == 'X')    printf("\n\t\tPlayer Win.\n");
            else if (res == 'T')    printf("\n\t\t  Draw.\n");
            break;
        }
      }
    }
    int repeat;
    printf("\n=============================================\n\n");
    printf("\t   Play again? (1/0) : ");
    scanf("%d", &repeat);
    if (repeat == 0) return 0;
  }
}
```

Play Game :

- We can choose the level first, the example we use is level 3



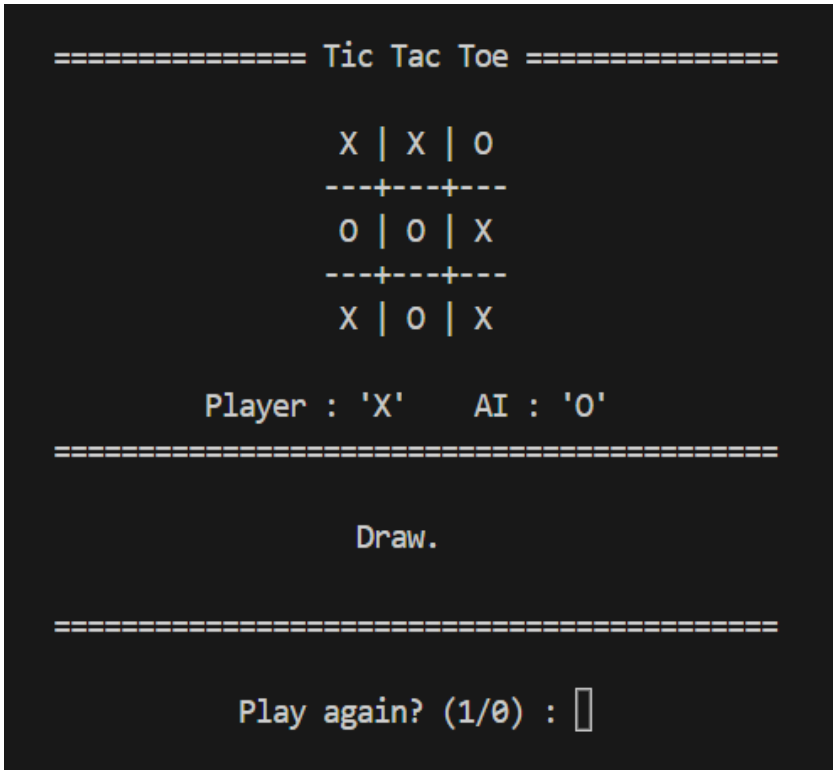
- after selecting the level, then there is the question "AI go first? (1/0)". If the player chooses 1, the AI will start the game first, if it chooses 0 then vice versa.



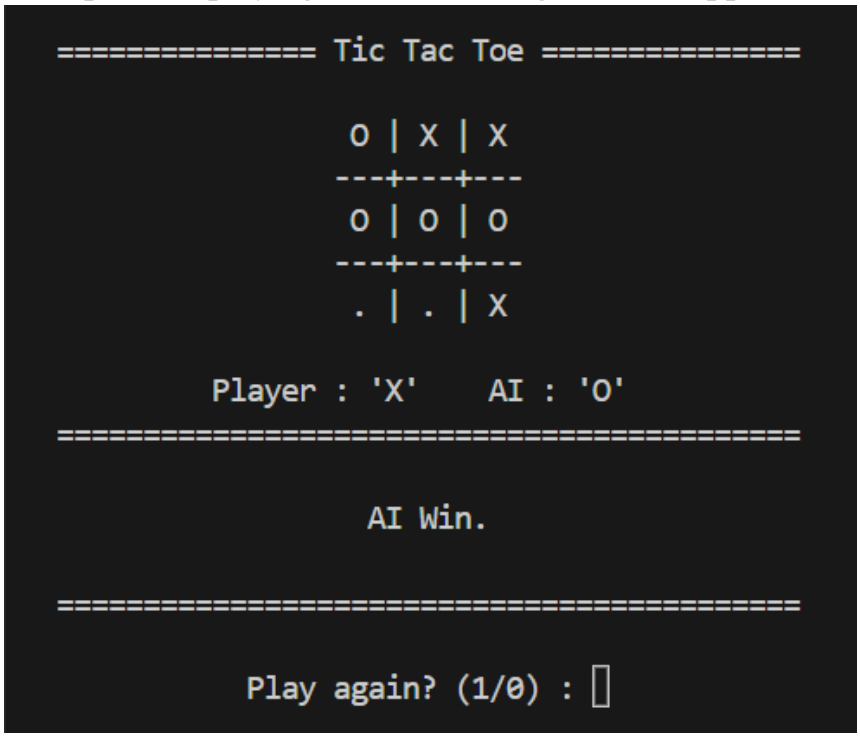
- Players can immediately start the game

- Then, the terminal will display player switching. Then, so on.

- The following is a display if the game is a draw. Then the option to play again or exit the game will appear.



- When the game has ended, the terminal will display the winner of the game. Then the option to play again or exit the game will appear.



Conclusion :

In the code above we have an implementation of the **Tic Tac Toe** game using the C++ programming language. The program displays a main menu with three difficulty levels: two-player game, player vs. bot, and player vs. artificial intelligence (AI).

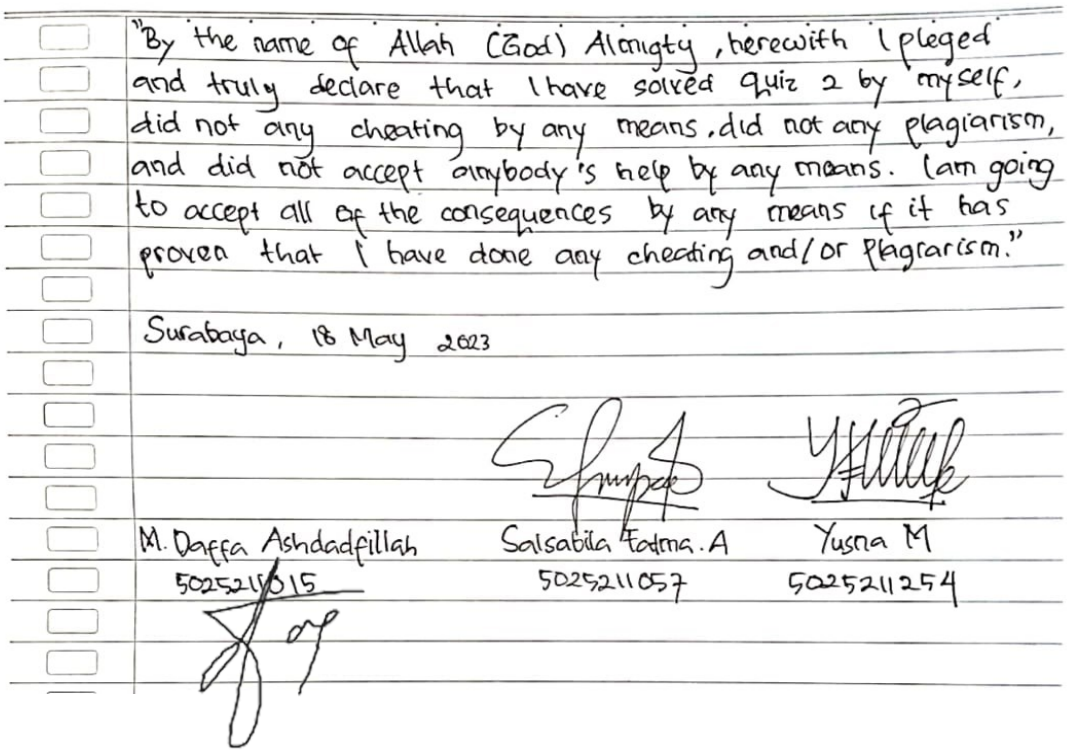
The code utilizes functions such as `**mainScreen()**` to display the menu, `**gameBoard()**` to show the game board, `**init()**` to initialize the game board, and `**result()**` to check the game result. Additionally, there are the `**minimax()**` function to implement the minimax algorithm in player vs. AI mode, and `**bestMove()**` to determine the best move for the AI.

Overall, this code implements the Tic Tac Toe game with various difficulty levels, incorporating programming concepts such as loops, functions, and decision-making.

Job Description and Distribution :

1. (5025211015) Muhammad Daffa Ashdaqfillah (33,33%)
    - Validation movement, resolving bot random move by time, debugging minimax algorithm.
2. (5025211057) Salsabila Fatma Aripa (33,33%)
    - Layouting terminal, making result checker, debugging minimax algorithm.
3. (5025211254) Yusna Millaturrosyidah (33,33%)
    - Creating game concept, adding turn bot and player movement, debugging minimax algorithm.

Github Link :
https://github.com/yusnaaaaa/IF184401_DAA_Q2_5025211015_5025211057_5025211254


"By the name of Allah (God) Almighty, herewith I pleged and truly declare that I have solved quiz 2 by myself, did not any cheating by any means, did not any plagiarism, and did not accept anybody's help by any means. I am going to accept all of the consequences by any means if it has proven that I have done any cheating and/or plagiarism."

Surabaya, 18 May 2023

M. Daffa Ashdadfillah 5025211015

Salsabila Fatma. A 5025211057

Yusna M 5025211254