Project Report

By:

Shahzaib Faisal\_2212349.

**ACKNOWLEDGMENT**

We are really grateful because we managed to complete our project within the time given by our teacher **Abid Ali & Miss Bakhtawar**. This assignment cannot be completed without the effort and co-operation from our group members, Group member **Aatir Raza\_221277.**

We also sincerely thank our teacher **Abid Ali & Miss Bakhtawar** for the guidance and encouragement in finishing this project and also for teaching us in this course.

Last but not the least, we would like to express our gratitude to our friends and respondents for the support and willingness directly or indirectly to spend some times with us to fill in the questionnaires.

**CONTENT**

[Introduction 4](#_Toc87292973)

[Objectives of the project 7](#_Toc87292974)

[Topic of Our Project 8](#_Toc87292975)

[Hardware/ Software Requirements 9](#_Toc87292976)

[WORK ANALYSIS 10](#_Toc87292977)

# Introduction

This code is a C program language code for a game where players can play against each other or against a computer. The game is played on a board of user-specified size, Row \*Column min(3\*3),max(9\*9) and the code includes functions for displaying the board, handling the logic of the game, and determining the end of the game or a win condition. It also includes a function for safe placement on the board and a function for the computer to play as either an eater or a passer. The code uses various libraries such as stdio, stdlib, unistd, string**,** and string to handle inputs, clear the screen, and use sleep mode.

* Two players take turns placing marking P or E into a grid
* Goal is to create a connected path of own markers
* If both players choose the same cell, "Eater" marker remains and "Passer" marker is eliminated
* Passer wins if they create a connected path from top to bottom
* Eater wins if Passer is unable to do so and the board is full
* Can be implemented as a computer program using a grid data structure and algorithms to check for connected path and determine winner.
* Computer can play as Eater and Passer if the player choose the second option.

# Objectives of the project

* Learning game development concepts
* Improving problem-solving skills
* Understanding memory management
* Collaborative learning
* Low-level optimization
* Building industry-level skills
* Creating a polished final product
* Understanding project management
* Improving technical skills
* Understanding software development process
* Creating a game that is relevant to their field of study
* Learning how to work with different tools and technologies

# Topic of Our Project

This is a C program for a game called Eater and Passer game that can be played in multiplayer mode or against a computer. The game's logic is implemented in the function "logic\_of\_game" which takes three parameters: size, board and check. The function "Gamestart" is called at the end of the "displayScreen" function, which allows the user to select the mode of the game. The function "Showboard" is used to display the game board to the user. The program uses some features of the C programming language, such as dynamic memory allocation, pointers, and data types such as int and bool. It also uses the sleep function from the unistd.h library to pause the program for a specified amount of time.

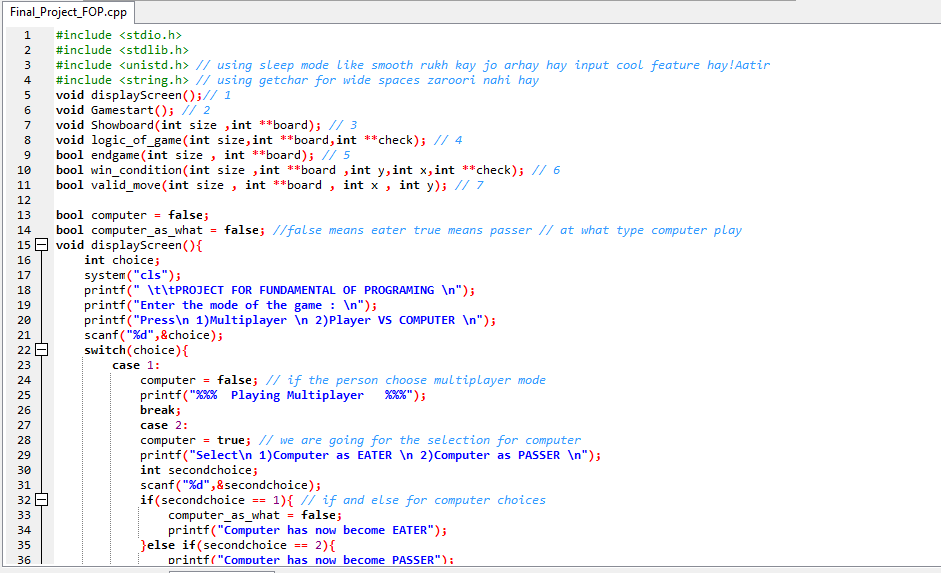
# Hardware/ Software Requirements

1. Computer or laptop
2. Mouse
3. Keyboard
4. IDE: DEV C/C++ or Code block

# WORK ANALYSIS

|  |  |  |
| --- | --- | --- |
| Task | Shahzaib Faisal | Aatir Raza |
| Analysis |  |  |
| Design |  |  |
| Coding |  |  |
| Testing |  |  |
| Documentation |  |  |

Code Snippets:



Explain the code:

This code is a program that allows a user to play a game called "Eater and Passer" on a square board of user-specified size. The user can choose to play in multiplayer mode or against a computer. If playing against a computer, the user can choose for the computer to play as an "Eater" or a "Passer." The game logic, board display, and various functions for validating moves and determining the end of the game are all implemented in this code.

FUNCTIONS:

1. void displayScreen();// 1
2. void Gamestart(); // 2
3. void Showboard(int size ,int \*\*board); // 3
4. void logic\_of\_game(int size, int \*\*board,int \*\*check); // 4
5. bool endgame(int size , int \*\*board); // 5
6. bool win\_condition(int size ,int \*\*board ,int y,int x,int \*\*check); // 6
7. bool valid\_move(int size , int \*\*board , int x , int y); // 7

1) void displayScreen();// 1

This code defines a function called "displayScreen()" that is used to display options for the user to select the mode of the game (multiplayer or player vs computer).

When the function is called, it first clears the screen using the system("cls") command and then displays a message asking the user to enter the mode of the game by pressing 1 for multiplayer or 2 for player vs computer. The user's choice is stored in the variable "choice" using the scanf() function.

Next, the code uses a switch statement to determine the action to take based on the value of "choice". If the user enters 1, the variable "computer" is set to false, indicating that the game is in multiplayer mode. If the user enters 2, the variable "computer" is set to true, indicating that the game is in player vs computer mode. In this case, the user is prompted to select whether the computer will be an "eater" or a "passer". This choice is stored in the variable "secondchoice".

If the user enters an invalid option, the default case is executed, which displays a message asking the user to contact the developer and then calls the displayScreen() function again to restart the game.

After the user has made a valid choice, the code displays a message saying that the game will start in 5 seconds and then calls the function "Gamestart()" to begin the game.

2) void Gamestart(); // 2

This code defines a function called "Gamestart()" which is responsible for setting up the game board for the player. It starts by prompting the user to enter the size of the board, which is stored in the variable "size". Next, the function uses malloc to dynamically allocate memory for two 2D arrays, "board" and "check". The board array is used to store the current state of the game, while the check array is used to keep track of which cells have been occupied or marked.

After allocating memory for the two arrays, the function uses nested for loops to initialize all elements of the board and check arrays to 0. This means that all cells on the game board are empty and unoccupied at the start of the game.

Finally, the function calls the "logic\_of\_game" and "Showboard" functions, passing in the size of the board, board array, and check array as arguments. The "logic\_of\_game" function is responsible for implementing the game's mechanics and rules, while the "Showboard" function is used to display the current state of the game board to the player.

3) void Showboard(int size ,int \*\*board); // 3

This code defines a function called "Showboard" which takes in two parameters: an integer "size" and a 2D integer array "board". The function is used to display the game board to the user.

The first line of the function prints the title "ROW:COLUMN:" followed by the column numbers. Then, the function uses a nested for loop to iterate through each element of the board array.

The first for loop iterates through the rows of the board, and the second for loop iterates through the columns of each row. Within the nested for loop, the code uses a switch statement to determine what to display for each element of the board array.

If the element is equal to 0, the code prints "|\*|" which represents an empty space on the board. If the element is equal to 1, the code prints "|P|" which represents a "Passer" move on the board. If the element is equal to 2, the code prints "|E|" which represents an "Eater" move on the board.

The function also prints new line after each row to show it like a matrix.

4) void logic\_of\_game(int size,int \*\*board,int \*\*check); // 4

This code is a function called "logic\_of\_game" that takes in three parameters: an integer called "size", a 2D integer array called "board", and a 2D integer array called "check".

The function starts by declaring some variables, including a boolean called "end" that is initially set to false. It also declares variables "p\_column", "p\_row", "e\_column", "e\_row" and "i".

The function then enters a while loop that continues until the "endgame" function (which is not defined in this code) returns true, or the "end" variable is set to true. Within the while loop, the function calls a "Showboard" function (also not defined in this code) to display the current state of the board.

Next, the function checks if the "computer" variable is true and the "computer\_as\_what" variable is also true. If so, it enters a loop that uses the "rand()" function to generate random coordinates for the "Passer" player's move. It continues looping until it finds a valid move (a spot on the board that is empty) and then prints the move to the console.

If the "computer" variable is not true or the "computer\_as\_what" variable is false, the function prompts the user for the Passer's move and reads the input from the console. It continues looping until it finds a valid move.

The function then does the same process for the "Eater" player, with the only difference being that it checks if the "computer" variable is true and the "computer\_as\_what" variable is false. If so, it generates random coordinates for the Eater's move using the "rand()" function, otherwise it prompts the user for the Eater's move.

After both players have made their moves, the function updates the board by setting the appropriate spots to 1 (for the Passer) and 2 (for the Eater). The function then enters a for loop that iterates "size" number of times. Within the for loop, the function calls a "win\_condition" function (not defined in this code) and passes in the size of the board, the board itself, the starting row and column, and the "check" array. If the "win\_condition" function returns true, the function then clears the console and displays the final state of the board and the "check" array. It also prints out "passer win" and prompts the user to press any key to continue. The function then sets the "end" variable to true and calls a "displayScreen" function (not defined in this code) before exiting the while loop.

It's not entirely clear what the purpose of the win\_condition function is and what the check array is used for. Also, the code doesn't seem to be complete and is missing some functions like endgame, Showboard and win\_condition.

5) bool endgame(int size , int \*\*board); // 5

This code defines a function called "endgame" which takes in two parameters, the size of the board represented by the variable "size" and a 2D array called "board" which represents the game board. The function uses nested for loops to iterate through each element of the board and checks if any of the elements are equal to 0. If an element is found to be equal to 0, the function returns false, indicating that the board is not full and the game is still ongoing. If all elements have been checked and none are equal to 0, the function prints "Eater won", calls the Showboard function to display the current state of the board, and waits for the user to press a key before displaying the main menu screen again. The function then returns true, indicating that the board is full and the game has ended.

6)Bool win\_condition(int size ,int \*\*board ,int y,int x,int \*\*check);// 6

This is a recursive function called "win\_condition" that checks if the player has won the game on the given board. The function takes in four parameters:

* "size" is the size of the board,
* "board" is the current state of the board,
* "y" and "x" are the current coordinates of the player on the board, and
* "check" is a 2D array used to keep track of the positions that have already been visited in the recursive call.

The function starts by checking if the player has reached the last row of the board and the cell at that position is a 1, which indicates that the player has won. If the condition is true, it sets the corresponding cell in the "check" array to 1 and returns true.

The function then checks if the current position is a valid move on the board by calling another function called "valid\_move". If the move is valid, the function then checks if the current position has been visited before by checking the corresponding cell in the "check" array. If it has been visited, the function returns false.

If the current position is valid and has not been visited before, the function sets the corresponding cell in the "check" array to 1 and then calls the function recursively for the three possible next moves: down, down-right, and down-left. If any of these recursive calls return true, the function also returns true, indicating that the player has won.

If none of the recursive calls return true, the function sets the corresponding cell in the "check" array back to 0 and returns false, indicating that the player has not won.

7) bool valid\_move(int size , int \*\*board , int x , int y); // 7

This function, named "valid\_move", is a boolean function that takes in four arguments: an integer "size" that represents the size of the board, a 2D integer array "board" that represents the current state of the board, an integer "y" that represents the current y-coordinate of a position being checked, and an integer "x" that represents the current x-coordinate of a position being checked.

The function checks whether the given x and y coordinates are within the boundaries of the board, and whether the board position at those coordinates is equal to 1. If both of these conditions are true, the function returns true, indicating that the given position is a valid move. If either of these conditions is false, the function returns false, indicating that the given position is not a valid move.

Main Function:-

This code defines the main function of the program. The main function is the starting point of a C or C++ program. The first line of the main function, "displayScreen();", calls the function "displayScreen()" which is defined earlier in the program. The last line, "return 0;", indicates that the program is ending and returning a value of 0 to the operating system. A return value of 0 typically indicates that the program has run successfully.

THANK YOU