TITLE: 2048 GUI GAME

Group members:

1. Abdul Salam Kalhoro (23K-0806) Group Leader.
2. Abdul Bari (23K-0659)
3. Syed Waleed Hussain (23K-0885)

# ABSTRACT:

2048 is a popular single player game played on 4x4 grid, where player’s task is to reach 2048 number by combining and rearranging numbers of same level for example: (2+2),(4+4),(8+8) up to **2048.**

Our project focuses on creating the popular 2048 puzzle game using the C programming language. We aim to build a well-structured and efficient solution by applying fundamental programming principles. The key aspects include representing the game board, effectively managing the game state, and implementing user interactions for moving tiles. Our main goal is to gain a deep understanding of algorithms, organize the code effectively, and create an enjoyable user interface for this classic game.

# Members Contribution:

1. Abdul Salam Kalhoro:

Contributed to basic structure and logic building.

Mainly worked on main function (), move () and can do move function () and also in system functions which includes clear screen and text color change to maximize the beauty of console-based program.

Main function includes function calling, conditional expressions and keymapping which is further enhanced by the member Syed Waleed Hussain.

2. Syed Waleed Hussain:

Contributed in a unique keymapping logic which helps to eliminate the use of characters in conditional expressions.

Mainly worked on new game () and add piece () function.

New game function starts with calling another function named player (), to ask for player information, and then proceed by initializing the whole 4x4 gameboard with zero and then calls another function add piece to randomly place digit “2” at free space.

3. Abdul Bari:

Contributed to score making logic, by going through the gameboard and making the highest number as the current score.

Mainly worked on player (), scores and lines () function,

The 2048 game implemented in C language embraces several key features aligning with the original project idea:

# Game Board Representation:

The project excels in accurately representing the 4x4 game board, essential for capturing the dynamic evolution of tiles during gameplay.

# Game State Management:

A robust mechanism is in place for managing the intricate game state, ensuring the seamless progression of the 2048 game and accurate tracking of the current state.

# Tile Movement and Merging Logic:

The heart of the implementation lies in the precise logic governing tile movements and mergers, adhering closely to the original rules of the 2048 game.

# User Input Handling:

The project incorporates a responsive system for handling user inputs, translating them into meaningful actions that drive the movement and merging of tiles on the game board.

# Code Organization and Modularity:

The codebase is organized with a keen eye on modularity, promoting clarity and maintainability. Each component contributes cohesively to the overall functionality of the game.

# Efficient Algorithm Implementation:

Algorithmic efficiency is a focal point, ensuring that the implementation is optimized for performance, enabling a smooth and responsive gaming experience.

# User Interface Design:

Through a text-based interface, the project pays attention to creating an engaging and user-friendly environment for players to interact with the game.

# Testing and Validation:

# We have used strict testing methods by individually executing functions one by one to make sure our implementation is correct. This includes doing small tests for different parts of the game and trying out different situations to see if everything works as it should.

# Challenges and Solutions:

The project report candidly discusses challenges encountered during development, accompanied by insightful solutions that showcase adaptability and problem-solving prowess.

Here is a summary of the code:

* 1. **Header Includes:** <stdio.h>, <stdlib.h>, and <conio.h>.



The code includes standard C libraries like <stdio.h>, <stdlib.h>, and <conio.h>. Note that the use

of might not be portable across all systems.

conio.h

**2. Function Declarations:**

Function prototypes for various functions used in the program are declared at the beginning of the code.

**3. Global Variables and Arrays:**

Global variables include the gameboard ( ), arrays for direction (dir\_Row

int gameboard [4] [4]

and dir\_Col), the player's name (char name [100]), and the score (int score).

**4. Functions:**

Player (): Takes the player's name as input.

Endgame (): Displays a game-over message with the player's name and score, then exits the program.

Scores (): Calculates and displays the current score based on the maximum value on the gameboard.

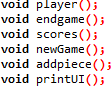
New Game (): Initializes the gameboard to zeros and adds the initial random piece.

Addpiece (): Adds a random piece (2) to the gameboard at an empty location.

PrintUI (): Prints the current state of the gameboard along with the player's score and available commands.

Move (): Moves the pieces on the gameboard based on the user's input direction.

Candomove (): Checks if a move is possible from the current position to the next position.



**5. Main Function:**

The main function initializes a mapping of commands to directions, starts a new game, and enters a loop where the user can input commands.

The user can input commands to start a new game (n), move the pieces (w, a, s, d), or quit the game (q).

The program continues until the user decides to quit.

**6. Suggestions for Improvement:**

**Include Headers:**

Included necessary C libraries<stdio.h>, <stdlib.h>, and <conio.h>.

**Function Declarations**

Declare functions for different aspects of the game such as player input, ending the game, scoring, starting a new game, adding a new piece, printing the game UI, and checking if a move is possible.

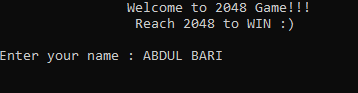
**Global Variables and Arrays:**

Declare global variables and arrays to store the game state, player information, and direction vectors.

**Player Input (player ()):**

Clear the console screen. Print a welcome message.

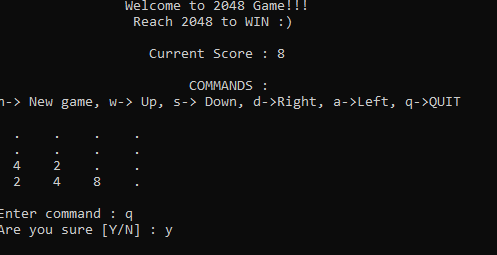
Prompt the player to enter their name using gets ().



**End Game (endgame ()):**

Clear the console screen.

Display a game-over message with the player's name and score. Exit the program using exit (q).



**Scoring (scores ()):**

Find the maximum value on the gameboard to determine the player's score. Display the current score.

**Initialize New Game (newGame ()):**

Call the function to get the player's name.

player ()

Initialize the gameboard to all zeros.

Add the initial random piece (2) using the function.

addpiece ()

**Add a Random Piece (addpiece ()):**

Generate random coordinates until an empty cell is found on the gameboard. Place a new piece (2) at the empty cell.

**Print Game UI (printUI ()):**

Clear the console screen.

Display the welcome message, instructions, and the current score. Print the gameboard along with dots for empty cells.

**Move Pieces (move ()):**

Determine the starting row and column based on the direction.

Use nested loops to iterate through the gameboard and move pieces in the specified direction.

Check if a move is possible using Update the gameboard.

candomove ()

Add a new random piece after the move.

and combine pieces if applicable.

**Check if Move is Possible (Candomove ()):**

Check if a move from the current position to the next position is valid. Ensure the next position is within the bounds of the gameboard.

Check if the pieces can be combined according to the game rules.

**Main Function (main ()):**

Initialize a mapping of commands to directions. Start a new game using new Game ().

Enter a loop to continuously prompt the user for commands.

Interpret user commands, allowing them to start a new game, move pieces, or quit the game.

These steps outline the flow of the program and the functionality at each stage. Keep in mind that the program uses the console for input and output and relies on a simple loop to handle user interactions.