### VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**Jnana Sangama, Belagavi - 590018**



## TECHNICAL TRAINING

**Mini project On**

## “ 2048” GAME

**By**

**ROHAN AMBIG 4MT21CS121**

**ROYAL GREESHAN DSOUZA 4MT21CS122**

**SACHIDANANDA NAYAK 4MT21CS123**

**SAHITH RAJ 4MT21CS125**

**DEPARTMENT OF COMPUTER SCIENCE& ENGINEERING**

**(*Accredited by NBA*)**

## MANGALORE INSTITUTE OF TECHNOLOGY & ENGINEERING

*Accredited by NAAC with A+ Grade, An ISO 9001: 2015 Certified Institution* (*A Unit of Rajalaxmi Education Trust®, Mangalore - 575001*) Affiliated to VTU, Belagavi, Approved by AICTE, New Delhi,**Badaga Mijar, Moodabidri-574225, Karnataka**

**2022-23**

**1. Abstract:**

This C program is an abstract representation of a simple command-line version of the popular game "2048." The program provides a text-based interface for playing the game on a 4x4 grid. Initialization , Printing the Board , Game Logic , User Input , Random Tile Generation , Game Loop, Randomization. This report provides an in-depth overview of the project's development, design, features, and functionality.

**2. Introduction:**

**2.1 Background:**

The implementation of the "2048" game in C provides a text-based representation of the popular puzzle game, "2048."

The program draws inspiration from the original "2048" game created by Gabriele Cirulli in 2014. The game involves sliding numbered tiles on a grid to combine them and achieve the tile with the value of 2048.

**2.2 Objectives:**

The primary objectives of this project are as follows:

* Educational Purpose: To serve as an educational project for programmers.
* Replicate Gameplay: To faithfully replicate the gameplay of the original "2048" game, including grid manipulation and tile combination mechanics.
* Text-Based Interface: To provide a minimalistic, text-based interface for accessibility, allowing users to play the game in a command-line environment.
* Randomization: To implement randomization of tile values and positions to create a dynamic and unpredictable gaming experience.
* Game Loop: To create a functional game loop that allows players to make moves, receive feedback, and continue playing until they decide to quit or the game concludes.

**3. Technologies Used:**

The technologies and tools used in the development of the Real Estate Advisor project include:

* C programming language
* Standard Input/Output (stdio.h)
* Standard Library (stdlib.h)
* stdbool.h
* Basic Control Flow
* User Input Handling

**4. System Architecture:**

The front-end displays the game board and relevant information using text characters, such as numbers to represent tile values and ASCII art to create a visual grid.

**4.1 Front-End:**

The front-end of the application is a command-line interface (CLI) that interacts with users.

Players interact with the game by providing input through the keyboard. They can use keys like 'W' (up), 'A' (left), 'S' (down), and 'D' (right) to make moves, and 'q' to quit the game.

**4.2 Back-End:**

Game Logic: This component includes functions and data structures responsible for managing the core game logic. It handles tasks such as tile movement, tile merging, score tracking, and detecting game over conditions, Game State, Randomization, Move Handling, Winning Condition, Data Structures.

**4.3 Database:**

The "2048" game implementation in C, as described in the previous responses, does not include a traditional database because it's a simple command-line-based game without persistent data storage needs.

**5. Project Modules:**

**5.1 Module 1: User Interface Module:**

Game Board Display, User Input Handling, Score Display.

**5.2 Module 2:Game Logic Module:**

Tile Movement Logic, Game State Management, Random Tile Generation.

**5.3 Module 2: Game Logic:**

Users can search for properties based on their location using this module.

**5.4 Module 3: Game Control Module:**

Game Initialization, Game Loop, Game Over Handling.

**5.5 Module 4: Graphics and Animation Module:**

Tile Animation.

**6. Design and Implementation:**

**6.1 Front-End Design:**

Designing the front-end of a graphical "2048" game involves creating an appealing and user-friendly interface for players.

**6.2 Back-End Design:**

The back-end design for a graphical "2048" game involves managing the game's logic, data, and functionality behind the scenes.

**6.3 Database Design:**

Designing a database for a "2048" game involves structuring the database schema to store essential game data, such as high scores, user profiles, and saved game states.

**7. Features and Functionality:**

**7.1 Gameplay Mechanics:**

Tile Movement, Tile Merging, Win Condition, Game Over.

**7.2 Game Board:**

Grid Layout, Tile Appearance, Tile Animation

**7.3 User Interaction:**

Keyboard Controls.

**8. Testing:**

Testing is a crucial phase in the development of a "2048" game to ensure that it works as intended, is free from bugs, and provides a positive user experience.

**8.1 Unit Testing:**

Develop unit tests for individual components, functions, or classes within the game code.

**8.2 Integration Testing:**

Conduct integration tests to verify that different game components work together seamlessly. Test interactions between the game logic, user interface, and input handling.

**8.3 User Acceptance Testing:**

Before releasing the game to the public, involve stakeholders or beta testers to perform UAT. Gather feedback and make final adjustments based on user preferences.

**9. Challenges Faced:**

During the development of the “2048” game project, several challenges were encountered, including:

* Game Logic Complexity
* User Interface Design
* Performance Optimization

These challenges were addressed through careful coding and testing.

**10. Future Enhancements:**

To further improve the “2048” game project, future enhancements can be considered, such as:

* Implementing a graphical user interface (GUI) for a more user-friendly experience.
* Multiplayer Mode.
* Tournaments.
* Leaderboards.

**11. Conclusion:**

In conclusion, developing a "2048" game offers valuable insights into programming and game design. Challenges are inevitable, but they provide opportunities for growth. Future enhancements can make the game more engaging. Game development is a journey of creativity, learning, and player enjoyment, promising both challenges and rewards.

**12)References:**

|  |  |
| --- | --- |
| |  | | --- | |  | |

<https://2048game.com/?ref=google-search-classic&gclid=Cj0KCQjwgNanBhDUARIsAAeIcAsa5Vs4iAqQZ4eifU0eHbFHYYQ05a2UFh5D7N-JxKYaJaD_Sq5NUfgaAqL5EALw_wcB>

<https://www.gameskinny.com/tips/2048-game-strategy-how-to-always-win-at-2048/>

**13)Appendices:**

**13.2)Code Snippets:**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <termios.h>

#include <unistd.h>

#define SIZE 4

struct game {

int tiles[SIZE][SIZE];

};

enum {

DOWN,

UP,

LEFT,

RIGHT,

MOVES

};

int random\_spot(struct game\* g)

{

// Generate a random number between 0 and SIZE - 1....

return rand() % SIZE;

}

int random\_value(struct game\* g)

{

// Generate a random number between 0 and 9 (inclusive)....

// If the result is not equal to 0, return 2. Otherwise, return 4....

return (rand() % 10) ? 2 : 4;

}

void print(struct game\* g)

{

int i,j;

printf("--------------------------\n");

for (j = SIZE - 1; j >= 0; --j) {

printf("|");

for (i = 0; i < SIZE; ++i) {

if (g->tiles[i][j])

printf("%4d ", g->tiles[i][j]);// used to access a specific tile value within a 2D array....

else

printf(" ");

}

printf("|\n");

}

printf("--------------------------\n");

}

void twist(struct game\* g)

{

int i,j;

struct game g2;// Create a temporary game structure....

for (i = 0; i < SIZE; ++i)

for (j = 0; j < SIZE; ++j)

g2.tiles[i][j] = g->tiles[j][i];// Swap rows and columns....

\*g = g2;

}

void flip(struct game\* g)

{

int i,j;

struct game g2;

for (i = 0; i < SIZE; ++i)

for (j = 0; j < SIZE; ++j)

g2.tiles[i][j] = g->tiles[i][SIZE - j - 1]; // Reverse order of tiles....

\*g = g2;

}

void begin(struct game\* g)

{

struct game g2 = {};// Create a new game structure g2 and initialize it with zeros....

\*g = g2;// Assign the values of g2 to the game structure pointed to by g....

g->tiles[random\_spot(g)][random\_spot(g)] = random\_value(g);// Place two random tiles on the board....

g->tiles[random\_spot(g)][random\_spot(g)] = random\_value(g);

}

void fall\_column(int\* a, int\* b)

{

int i,j;

int prev = 0;

j = 0;

for (i = 0; i < SIZE; ++i)

if (a[i]) {

if (a[i] == prev) {

b[j - 1] \*= 2;

prev = 0;

} else {

b[j++] = a[i];

prev = a[i];

}

}

}

void fall(struct game\* g)

{

struct game g2 = {};

int i;

for (i = 0; i < SIZE; ++i)

fall\_column(g->tiles[i], g2.tiles[i]);

\*g = g2;

}

int same(struct game\* a, struct game\* b)

{

int i,j;

for (i = 0; i < SIZE; ++i)

for (j = 0; j < SIZE; ++j)

if (a->tiles[i][j] != b->tiles[i][j])

return 0;

return 1;

}

int tryfalling(struct game\* g)

{

struct game g2 = \*g;

fall(g);

if (same(g, &g2))

return 0;

return 1;

}

void popup(struct game\* g)

{

int i,j;

while (1) {

i = random\_spot(g);

j = random\_spot(g);

if (!g->tiles[i][j]) {

g->tiles[i][j] = random\_value(g);

return;

}

}

}

void move(struct game\* g, int way)

{

if (way / 2)// If way is even, twist the game board....

twist(g);

// If way is odd....

if (way % 2)// The function calls the flip function to flip the game board horizontally....

flip(g);

if (tryfalling(g))

popup(g);

if (way % 2)

flip(g);

if (way / 2)

twist(g);

}

int read\_move(void)

{

char keys[MOVES] = {'k','i','j','l'};

int c;

int i;

//isspace is a standard library function....

//that checks whether a character is a whitespace character (space, tab, newline, etc.)....

//checks if the read character is a whitespace character. If it is, the loop continues....

while (isspace(c = getchar()));

if (c == EOF)

return c;

// Compare the character with predefined keys....

for (i = 0; i < MOVES; ++i)

if (c == keys[i])

return i; // Return the index corresponding to the mov....

return EOF; // If the input character doesn't match any predefined key, return EOF....

}

static struct termios backup;

static struct termios current;

> Rohan Ambig:

void take\_stdin(void)

{

tcgetattr(STDIN\_FILENO, &backup); // Backup the current terminal attributes and it's helpful for storing the terminal attributes in backup structure

// STDIN\_FILENO : allows read the input from the user....

current = backup;

// This line copies the attributes from the backup structure to the current structure....

current.c\_lflag &= (~ICANON & ~ECHO);// this will helpful for avoiding the character display on the terminal after pressing the keys....

tcsetattr(STDIN\_FILENO, TCSANOW, &current);//is used to apply modified terminal attributes after press the key....

//TCSANOW indicates that the changes should take effect immediately in terminal....

// &current : which holds the modified terminal attributes that you want to apply....

}

void give\_stdin(void)

{

tcsetattr(STDIN\_FILENO, TCSANOW, &backup);

}

int main()

{

int c;

struct game g;// Declare an instance of the struct game called g....

begin(&g);// Call the begin function to initialize the game g....

print(&g);

take\_stdin();

// Read characters from standard input until EOF is encountered.....

while ((c = read\_move()) != EOF) {

move(&g, c);

print(&g);

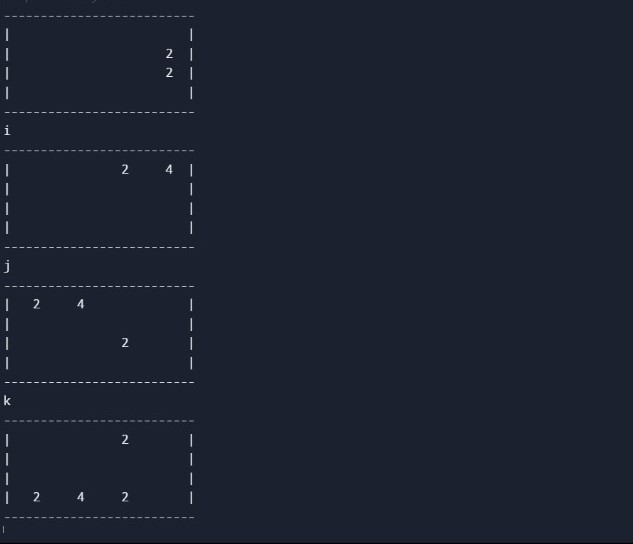
}

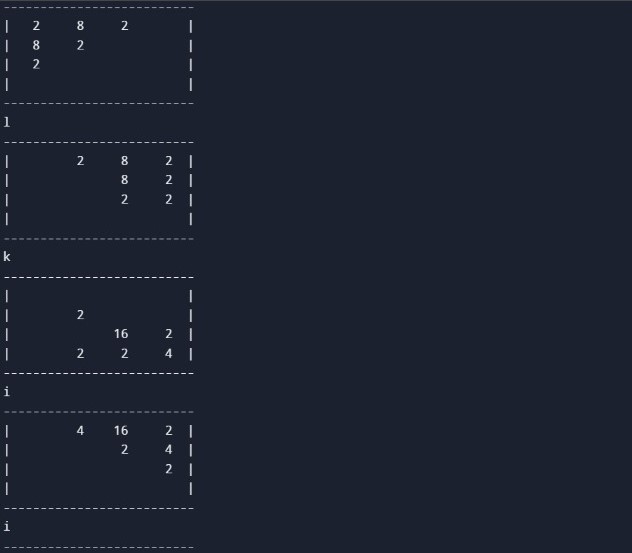
give\_stdin();

return 0;

}}

(Screenshots):

****

****