VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA,BELAGAVI – 590018

KARNATAKA

Assignment Report
On
"2048 GAME"

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DATA STRUCTURES AND APPLICATIONS (BCS304) COURSE OF III SEMESTER

Submitted by

AMITH A N
1CG22CS008
JAYASURYA N
1CG22CS047

Guide:
Mr.Asif Ulla Khan .M. Tech.
Asst. Prof., Dept. of CSE
CIT, Gubbi.

HOD:
Dr. Shantala C P PhD.,
Head, Dept. of CSE
CIT, Gubbi.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Channabasaveshwara Institute of Technology

(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)

(**NAAC Accredited & ISO 9001:2015 Certified Institution)**

NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka

2023-24

Rubric – B.E. Mini-Project [BCS304]

Course outcome	Rubric/Level	Excellent (91-100%)	Good (81-90%)	Average (61-80%)	Moderate (40-60%)	Score
CO1	Identification of project proposal (05 Marks)					
CO2	Design and Implementation (10 Marks)					
CO3	Presentation skill (05 Marks)					
CO4	Report (05 Marks)					
Total						

Course outcome:

CO 1: Identification of project proposal which is relevant to subject of engineering.

CO 2: Design and implement proposed project methodology.

CO 3: Effective communication skill to assimilate their project work.

CO 4: Understanding overall project progress and performance.

Student Signature **Faculty signature**

ABSTRACT

"2048" is a popular single-player puzzle game played on a 4x4 grid. The game starts with two tiles, each displaying a value of 2 or 4. Players move the tiles in four directions: up, down, left, and right, with each move causing all tiles to slide as far as possible in the chosen direction. When two tiles with the same value collide, they merge into a single tile displaying their sum. The objective is to create a tile with the value of 2048 by strategically combining tiles while preventing the grid from filling up. The game presents a simple yet addictive gameplay mechanic, challenging players to plan their moves carefully to achieve the highest possible score.

CHAPTER 1:

INTRODUCTION

2048" is a captivating and addictive single-player puzzle game that has taken the gaming world by storm since its inception in 2014. Played on a 4x4 grid, the game involves combining numbered tiles to reach the elusive tile displaying the value of 2048. With simple rules and mechanics, players must slide tiles in four directions—up, down, left, and right—to merge matching numbers and create higher-valued tiles. As the grid fills up with each move, strategic planning becomes crucial to avoid getting stuck and ultimately achieve the coveted 2048 tile. Easy to learn yet challenging to master, 2048 offers hours of engaging gameplay and has become a favorite pastime for puzzle enthusiasts worldwide.

CHAPTER 2:

PROBLEM STATEMENT

The 2048 game, although popular and engaging, presents several challenges and opportunities for improvement. Key issues include:

- Lack of Advanced Strategy Guidance: While the 2048 game is easy to grasp, mastering advanced strategies and techniques remains elusive for many players. There is a need for comprehensive guidance or tutorials that delve deeper into strategic approaches, optimal tile placement, and effective decision-making to enhance player performance and satisfaction.
- Limited Cognitive Feedback: While playing 2048 can stimulate cognitive skills such as pattern recognition and decision-making, the game currently lacks explicit feedback mechanisms to inform players about their cognitive strengths and areas for improvement. Incorporating cognitive feedback tools or metrics could enhance the educational value of the game and promote cognitive skill development.
- Accessibility and Inclusivity: While 2048's gameplay is straightforward, it may not be fully accessible to players with diverse abilities or preferences. There is a need to explore options for customization, such as adjustable difficulty levels, alternative control schemes, or accessibility features, to ensure inclusivity and accommodate a broader range of players.
- Engagement and Longevity: Despite its initial popularity, player engagement with the 2048 game may decline over time due to repetitive gameplay and limited content variety. Introducing new game modes, challenges, or social features could reinvigorate player interest and prolong the game's longevity.
- Integration of Learning Elements: While the 2048 game inherently promotes cognitive skills development, there is an opportunity to integrate explicit learning elements or educational content seamlessly into the gameplay experience. This could include incorporating mathematical concepts, problem-solving challenges, or interactive tutorials to foster both entertainment and learning outcomes.

Addressing these challenges presents an opportunity to enhance the 2048 game's appeal, accessibility, and educational value. By leveraging advanced strategies, providing cognitive feedback, improving accessibility features, increasing engagement, and integrating learning elements, developers can create a more immersive and enriching gaming experience for players of all backgrounds and abilities.

CHAPTER 3:

IMPLEMENTATION

```
#define _XOPEN_SOURCE 500 // for: usleep
#include <stdio.h>      // defines: printf, puts, getchar
#include <stdlib.h>     // defines: EXIT_SUCCESS
#include <string.h>     // defines: strcmp
#include <unistd.h>     // defines: STDIN_FILENO, usleep
#include <termios.h>    // defines: termios, TCSANOW, ICANON, ECHO
#include <stdbool.h>    // defines: true, false
#include <stdint.h>     // defines: uint8_t, uint32_t
#include <time.h>       // defines: time
#include <signal.h>     // defines: signal, SIGINT

#define SIZE 4

// this function receives 2 pointers (indicated by *) so it can set their values
void getColors(uint8_t value, uint8_t scheme, uint8_t *foreground, uint8_t *background)
{
        uint8_t original[] = {8, 255, 1, 255, 2, 255, 3, 255, 4, 255, 5, 255, 6, 255, 7, 255, 9, 0,
10, 0, 11, 0, 12, 0, 13, 0, 14, 0, 255, 0, 255, 0};
        uint8_t blackwhite[] = {232, 255, 234, 255, 236, 255, 238, 255, 240, 255, 242, 255,
244, 255, 246, 0, 248, 0, 249, 0, 250, 0, 251, 0, 252, 0, 253, 0, 254, 0, 255, 0};
        uint8_t bluered[] = {235, 255, 63, 255, 57, 255, 93, 255, 129, 255, 165, 255, 201, 255,
200, 255, 199, 255, 198, 255, 197, 255, 196, 255, 196, 255, 196, 255, 196, 255, 196, 255};
        uint8_t *schemes[] = {original, blackwhite, bluered};
        // modify the 'pointed to' variables (using a * on the left hand of the assignment)
        *foreground = *(schemes[scheme] + (1 + value * 2) % sizeof(original));
        *background = *(schemes[scheme] + (0 + value * 2) % sizeof(original));
        // alternatively we could have returned a struct with two variables
}

uint8_t getDigitCount(uint32_t number)
{
        uint8_t count = 0;
```

```
	do
	{
		number /= 10;
		count += 1;
	} while (number);
	return count;
}

void drawBoard(uint8_t board[SIZE][SIZE], uint8_t scheme, uint32_t score)
{
	uint8_t x, y, fg, bg;
	printf("\033[H"); // move cursor to 0,0
	printf("2048.c %17d pts\n\n", score);
	for (y = 0; y < SIZE; y++)
	{
		for (x = 0; x < SIZE; x++)
		{
			// send the addresses of the foreground and background variables,
			// so that they can be modified by the getColors function
			getColors(board[x][y], scheme, &fg, &bg);
			printf("\033[38;5;%d;48;5;%dm", fg, bg); // set color
			printf("       ");
			printf("\033[m"); // reset all modes
		}
		printf("\n");
		for (x = 0; x < SIZE; x++)
		{
			getColors(board[x][y], scheme, &fg, &bg);
			printf("\033[38;5;%d;48;5;%dm", fg, bg); // set color
			if (board[x][y] != 0)
			{
				uint32_t number = 1 << board[x][y];
				uint8_t t = 7 - getDigitCount(number);
				printf("%*s%u%*s", t - t / 2, "", number, t / 2, "");
			}
```

```
                else
                {
                        printf("  ·  ");
                }
                printf("\033[m"); // reset all modes
        }
        printf("\n");
        for (x = 0; x < SIZE; x++)
        {
                getColors(board[x][y], scheme, &fg, &bg);
                printf("\033[38;5;%d;48;5;%dm", fg, bg); // set color
                printf("      ");
                printf("\033[m"); // reset all modes
        }
        printf("\n");
    }
    printf("\n");
    printf("       ←,↑,→,↓ or q       \n");
    printf("\033[A"); // one line up
}

uint8_t findTarget(uint8_t array[SIZE], uint8_t x, uint8_t stop)
{
    uint8_t t;
    // if the position is already on the first, don't evaluate
    if (x == 0)
    {
        return x;
    }
    for (t = x - 1;; t--)
    {
        if (array[t] != 0)
        {
            if (array[t] != array[x])
            {
```

```
                        // merge is not possible, take next position
                        return t + 1;
                }
                return t;
        }
        else
        {
                // we should not slide further, return this one
                if (t == stop)
                {
                        return t;
                }
        }
    }
    // we did not find a target
    return x;
}

bool slideArray(uint8_t array[SIZE], uint32_t *score)
{
    bool success = false;
    uint8_t x, t, stop = 0;

    for (x = 0; x < SIZE; x++)
    {
        if (array[x] != 0)
        {
            t = findTarget(array, x, stop);
            // if target is not original position, then move or merge
            if (t != x)
            {
                // if target is zero, this is a move
                if (array[t] == 0)
                {
                    array[t] = array[x];
```

```
                        }
                        else if (array[t] == array[x])
                        {
                                // merge (increase power of two)
                                array[t]++;
                                // increase score
                                *score += (uint32_t)1 << array[t];
                                // set stop to avoid double merge
                                stop = t + 1;
                        }
                        array[x] = 0;
                        success = true;
                }
        }
    }
    return success;
}

void rotateBoard(uint8_t board[SIZE][SIZE])
{
    uint8_t i, j, n = SIZE;
    uint8_t tmp;
    for (i = 0; i < n / 2; i++)
    {
        for (j = i; j < n - i - 1; j++)
        {
            tmp = board[i][j];
            board[i][j] = board[j][n - i - 1];
            board[j][n - i - 1] = board[n - i - 1][n - j - 1];
            board[n - i - 1][n - j - 1] = board[n - j - 1][i];
            board[n - j - 1][i] = tmp;
        }
    }
}
```

```
bool moveUp(uint8_t board[SIZE][SIZE], uint32_t *score)
{
	bool success = false;
	uint8_t x;
	for (x = 0; x < SIZE; x++)
	{
		success |= slideArray(board[x], score);
	}
	return success;
}

bool moveLeft(uint8_t board[SIZE][SIZE], uint32_t *score)
{
	bool success;
	rotateBoard(board);
	success = moveUp(board, score);
	rotateBoard(board);
	rotateBoard(board);
	rotateBoard(board);
	return success;
}

bool moveDown(uint8_t board[SIZE][SIZE], uint32_t *score)
{
	bool success;
	rotateBoard(board);
	rotateBoard(board);
	success = moveUp(board, score);
	rotateBoard(board);
	rotateBoard(board);
	return success;
}

bool moveRight(uint8_t board[SIZE][SIZE], uint32_t *score)
{
```

```
	bool success;
	rotateBoard(board);
	rotateBoard(board);
	rotateBoard(board);
	success = moveUp(board, score);
	rotateBoard(board);
	return success;
}

bool findPairDown(uint8_t board[SIZE][SIZE])
{
	bool success = false;
	uint8_t x, y;
	for (x = 0; x < SIZE; x++)
	{
		for (y = 0; y < SIZE - 1; y++)
		{
			if (board[x][y] == board[x][y + 1])
				return true;
		}
	}
	return success;
}

uint8_t countEmpty(uint8_t board[SIZE][SIZE])
{
	uint8_t x, y;
	uint8_t count = 0;
	for (x = 0; x < SIZE; x++)
	{
		for (y = 0; y < SIZE; y++)
		{
			if (board[x][y] == 0)
			{
				count++;
```

```
                        }
                }
        }
        return count;
}

bool gameEnded(uint8_t board[SIZE][SIZE])
{
        bool ended = true;
        if (countEmpty(board) > 0)
                return false;
        if (findPairDown(board))
                return false;
        rotateBoard(board);
        if (findPairDown(board))
                ended = false;
        rotateBoard(board);
        rotateBoard(board);
        rotateBoard(board);
        return ended;
}

void addRandom(uint8_t board[SIZE][SIZE])
{
        static bool initialized = false;
        uint8_t x, y;
        uint8_t r, len = 0;
        uint8_t n, list[SIZE * SIZE][2];

        if (!initialized)
        {
                srand(time(NULL));
                initialized = true;
        }
```

```
	for (x = 0; x < SIZE; x++)
	{
		for (y = 0; y < SIZE; y++)
		{
			if (board[x][y] == 0)
			{
				list[len][0] = x;
				list[len][1] = y;
				len++;
			}
		}
	}

	if (len > 0)
	{
		r = rand() % len;
		x = list[r][0];
		y = list[r][1];
		n = (rand() % 10) / 9 + 1;
		board[x][y] = n;
	}
}

void initBoard(uint8_t board[SIZE][SIZE])
{
	uint8_t x, y;
	for (x = 0; x < SIZE; x++)
	{
		for (y = 0; y < SIZE; y++)
		{
			board[x][y] = 0;
		}
	}
	addRandom(board);
	addRandom(board);
```

```
}

void setBufferedInput(bool enable)
{
	static bool enabled = true;
	static struct termios old;
	struct termios new;

	if (enable && !enabled)
	{
		// restore the former settings
		tcsetattr(STDIN_FILENO, TCSANOW, &old);
		// set the new state
		enabled = true;
	}
	else if (!enable && enabled)
	{
		// get the terminal settings for standard input
		tcgetattr(STDIN_FILENO, &new);
		// we want to keep the old setting to restore them at the end
		old = new;
		// disable canonical mode (buffered i/o) and local echo
		new.c_lflag &= (~ICANON & ~ECHO);
		// set the new settings immediately
		tcsetattr(STDIN_FILENO, TCSANOW, &new);
		// set the new state
		enabled = false;
	}
}

int test()
{
	uint8_t array[SIZE];
	// these are exponents with base 2 (1=2 2=4 3=8)
	// data holds per line: 4x IN, 4x OUT, 1x POINTS
```

```
uint8_t data[] = {
    0, 0, 0, 1, 1, 0, 0, 0, 0,
    0, 0, 1, 1, 2, 0, 0, 0, 4,
    0, 1, 0, 1, 2, 0, 0, 0, 4,
    1, 0, 0, 1, 2, 0, 0, 0, 4,
    1, 0, 1, 0, 2, 0, 0, 0, 4,
    1, 1, 1, 0, 2, 1, 0, 0, 4,
    1, 0, 1, 1, 2, 1, 0, 0, 4,
    1, 1, 0, 1, 2, 1, 0, 0, 4,
    1, 1, 1, 1, 2, 2, 0, 0, 8,
    2, 2, 1, 1, 3, 2, 0, 0, 12,
    1, 1, 2, 2, 2, 3, 0, 0, 12,
    3, 0, 1, 1, 3, 2, 0, 0, 4,
    2, 0, 1, 1, 2, 2, 0, 0, 4};
uint8_t *in, *out, *points;
uint8_t t, tests;
uint8_t i;
bool success = true;
uint32_t score;

tests = (sizeof(data) / sizeof(data[0])) / (2 * SIZE + 1);
for (t = 0; t < tests; t++)
{
        in = data + t * (2 * SIZE + 1);
        out = in + SIZE;
        points = in + 2 * SIZE;
        for (i = 0; i < SIZE; i++)
        {
                array[i] = in[i];
        }
        score = 0;
        slideArray(array, &score);
        for (i = 0; i < SIZE; i++)
        {
                if (array[i] != out[i])
```

```
                {
                        success = false;
                }
        }
        if (score != *points)
        {
                success = false;
        }
        if (success == false)
        {
                for (i = 0; i < SIZE; i++)
                {
                        printf("%d ", in[i]);
                }
                printf("=> ");
                for (i = 0; i < SIZE; i++)
                {
                        printf("%d ", array[i]);
                }
                printf("(%d points) expected ", score);
                for (i = 0; i < SIZE; i++)
                {
                        printf("%d ", in[i]);
                }
                printf("=> ");
                for (i = 0; i < SIZE; i++)
                {
                        printf("%d ", out[i]);
                }
                printf("(%d points)\n", *points);
                break;
        }
}
if (success)
{
```

```
            printf("All %u tests executed successfully\n", tests);
        }
        return !success;
}

void signal_callback_handler(int signum)
{
        printf("       TERMINATED       \n");
        setBufferedInput(true);
        // make cursor visible, reset all modes
        printf("\033[?25h\033[m");
        exit(signum);
}

int main(int argc, char *argv[])
{
        uint8_t board[SIZE][SIZE];
        uint8_t scheme = 0;
        uint32_t score = 0;
        char c;
        bool success;

        if (argc == 2 && strcmp(argv[1], "test") == 0)
        {
            return test();
        }
        if (argc == 2 && strcmp(argv[1], "blackwhite") == 0)
        {
            scheme = 1;
        }
        if (argc == 2 && strcmp(argv[1], "bluered") == 0)
        {
            scheme = 2;
        }
```

```
// make cursor invisible, erase entire screen
printf("\033[?25l\033[2J");

// register signal handler for when ctrl-c is pressed
signal(SIGINT, signal_callback_handler);

initBoard(board);
setBufferedInput(false);
drawBoard(board, scheme, score);
while (true)
{
	c = getchar();
	if (c == -1)
	{
		puts("\nError! Cannot read keyboard input!");
		break;
	}
	switch (c)
	{
	case 97:  // 'a' key
	case 104: // 'h' key
	case 68:  // left arrow
		success = moveLeft(board, &score);
		break;
	case 100: // 'd' key
	case 108: // 'l' key
	case 67:  // right arrow
		success = moveRight(board, &score);
		break;
	case 119: // 'w' key
	case 107: // 'k' key
	case 65:  // up arrow
		success = moveUp(board, &score);
		break;
	case 115: // 's' key
```

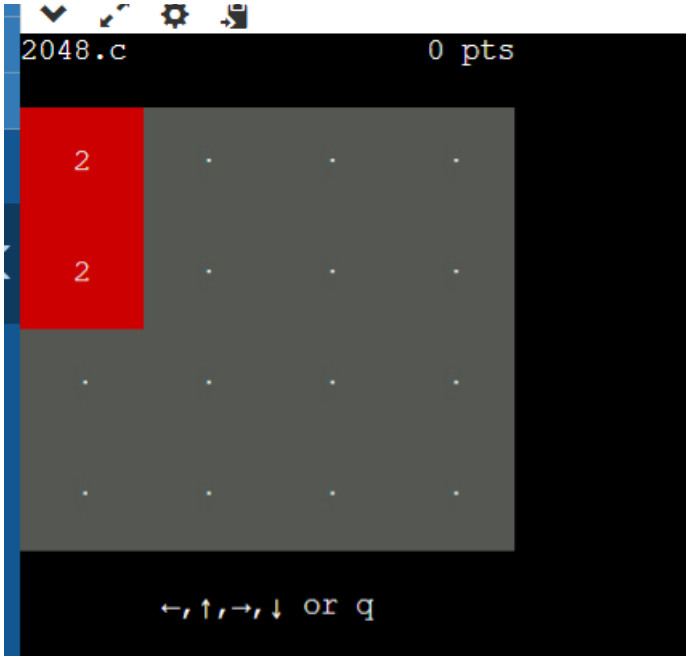
```
case 106: // 'j' key
case 66:  // down arrow
        success = moveDown(board, &score);
        break;
default:
        success = false;
}
if (success)
{
        drawBoard(board, scheme, score);
        usleep(150 * 1000); // 150 ms
        addRandom(board);
        drawBoard(board, scheme, score);
        if (gameEnded(board))
        {
                printf("        GAME OVER         \n");
                break;
        }
}
if (c == 'q')
{
        printf("        QUIT? (y/n)        \n");
        c = getchar();
        if (c == 'y')
        {
                break;
        }
        drawBoard(board, scheme, score);
}
if (c == 'r')
{
        printf("       RESTART? (y/n)       \n");
        c = getchar();
        if (c == 'y')
        {
```

```
                        initBoard(board);
                        score = 0;
                    }
                    drawBoard(board, scheme, score);
                }
            }
            setBufferedInput(true);

            // make cursor visible, reset all modes
            printf("\033[?25h\033[m");

            return EXIT_SUCCESS;
}
```

CHAPTER 4:

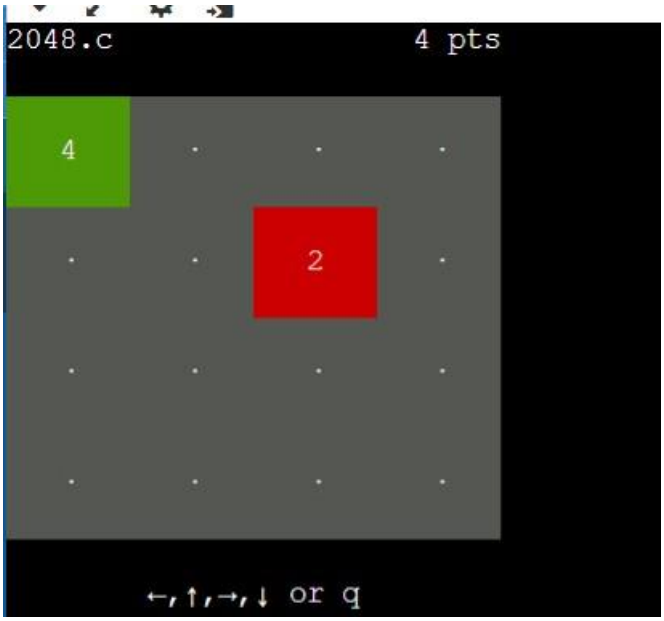
RESULT / SCREENSHOT

STEP-1:



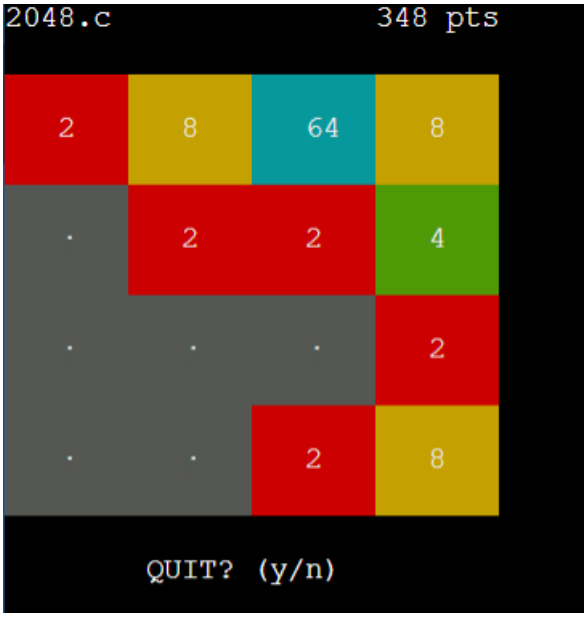
The game begins with an empty 4x4 grid. Two randomly placed tiles with a value of either 2 or 4 appear on the grid.

STEP-2:



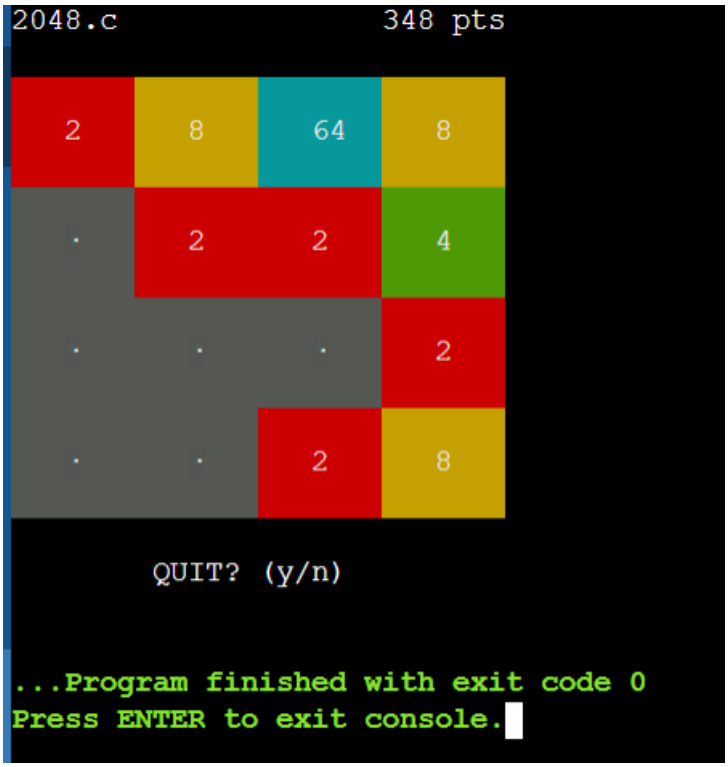
Players can swipe in four directions: up, down, left, or right. Each swipe moves all tiles in the chosen direction as far as possible. Tiles with the same value merge into a single tile of twice the original value when they collide during a move.

STEP-3:



Players earn points whenever tiles merge.The score increases by the value of the merged tile (e.g., merging two tiles of 32 each earns 64 points).

STEP-4:



The game ends when the grid is full, and no more moves can be made.Players lose if there are no available moves to merge tiles.

CHAPTER 5:

CONCLUSION

In conclusion, 2048 is a simple yet engaging puzzle game that has captivated players worldwide with its addictive gameplay and strategic depth. With its minimalist design and easy-to-understand mechanics, 2048 offers a challenging experience that keeps players coming back for more. Whether you're a casual gamer looking for a quick mental challenge or a puzzle aficionado seeking to master the intricacies of tile merging, 2048 provides endless entertainment and satisfaction. So, whether you've reached the elusive 2048 tile or are still striving to achieve it, 2048 remains a timeless classic in the realm of puzzle gaming.

Reference

- https://github.com/mevdschee/2048.c/blob/main/2048.c
- https://chat.openai.com/
- https://www.geeksforgeeks.org/2048-game-in-c/