**Project Title: *2048 Game in C***

**Introduction:**

The 2048 game has gained immense popularity since its release in 2014, captivating puzzle enthusiasts with its addictive gameplay and strategic challenges.

As part of our undergraduate assignment, this project report aims to present the implementation and analysis of the renowned 2048 game. Our objective is to recreate the game and delve into its mechanics, gameplay strategies, and performance aspects. By undertaking this project, we seek to understand the underlying algorithms, evaluate user experience through the graphical user interface (GUI), and analyze the game's performance in terms of execution time and memory utilization.

The insights gained from this project will provide valuable insights into the intricacies of game development and offer an opportunity to explore optimization techniques for enhancing gameplay and performance.

**Functions Used in making the 2048 Game:**

The source code for the 2048 game may appear lengthy, but it is designed to be easily comprehensible. The C program is organized into several functions, each serving a specific purpose. These functions include initializing the game board, printing the board, adding tiles, checking the game over condition, and implementing movements in different directions such as up, down, right, and left. The following are the key functions used in the program, which will aid in better understanding of the project:

1. **`initializeBoard()`**: This function initializes the game board by setting all elements of the **`board`** array to 0. It is called at the beginning of the program to prepare the game board for gameplay.
2. **`printBoard()`**: This function displays the current state of the game board. It prints the elements of the **`board`** array in a tabular format and prompts the user for their next move.
3. **`addRandomTile()`**: This function adds a random tile (either 2 or 4) to an empty cell on the game board. It first identifies all the empty cells by iterating over the **`board`** array and stores their coordinates in the **`emptyCells`** array. It then randomly selects an empty cell and assigns a random value (2 or 4) to it.
4. **`isGameOver()`**: This function checks if the game is over by examining the game board. It returns 1 if the player has achieved the 2048 tile, 0 if there are empty cells on the board, and no neighboring cells with the same value, indicating that the game can still continue. It is used in the main loop to determine if the game should end.
5. **`mergeTiles(int row[], int size)`**: This function is responsible for merging adjacent tiles with the same value in a given row. It iterates over the row and if it finds two adjacent tiles with the same value, it doubles the value of the first tile and sets the second tile to 0.
6. **`shiftTilesLeft(int row[], int size)`**: This function shifts all the tiles in a given row to the left, eliminating any gaps caused by merging tiles. It moves each non-zero tile to the leftmost available position and fills the remaining positions with zeros.
7. **`moveLeft()`**: This function instructs the game to move all the tiles on the board to the left. It achieves this by calling **`mergeTiles()`** and **`shiftTilesLeft()`** for each row in the board.
8. **`reverse(int row[], int size)`**: This function reverses the order of elements in a given row. It is used to enable moving tiles to the right by reversing a row, performing the left movement, and reversing it back.
9. **`transposeBoard()`**: This function transposes the game board by swapping the elements across the main diagonal. It is used to enable moving tiles up and down by treating the columns as rows.
10. **`moveRight()`**: This function instructs the game to move all the tiles on the board to the right. It achieves this by reversing each row, calling **`mergeTiles()`** and **`shiftTilesLeft()`**, and reversing the row back.
11. **`moveUp()`**: This function instructs the game to move all the tiles on the board upwards. It achieves this by transposing the board, calling **`moveLeft()`**, and transposing the board back.
12. **`moveDown()`**: This function instructs the game to move all the tiles on the board downwards. It achieves this by transposing the board, calling **`moveRight()`**, and transposing the board back.

These functions collectively implement the core functionalities of the 2048 game, including initializing the board, printing the board, adding random tiles, checking the game over condition, merging and shifting tiles, and handling player moves in different directions.

**Source Code:**

//C Program to create a 2048 Game using the Data Structure - Arrays

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define SIZE 4

int board[SIZE][SIZE];

//Initialises the Game Board in the form of memory

void initializeBoard() {

    int i, j;

    for (i = 0; i < SIZE; i++) {

        for (j = 0; j < SIZE; j++) {

            board[i][j] = 0;

        }

    }

}

//Displays the Game Board

void printBoard() {

    int i, j;

    system("cls");

    for (i = 0; i < SIZE; i++) {

        for (j = 0; j < SIZE; j++) {

            printf("%d\t", board[i][j]);

        }

        printf("\n");

    }

    printf("Enter move (w: up, s: down, a: left, d: right): ");

}

//Add 2 or 4 in any of the empty tiles

void addRandomTile() {

    int i, j;

    int emptyCells[SIZE \* SIZE][2];

    int numEmptyCells = 0;

    for (i = 0; i < SIZE; i++) {

        for (j = 0; j < SIZE; j++) {

            if (board[i][j] == 0) {

                emptyCells[numEmptyCells][0] = i;

                emptyCells[numEmptyCells][1] = j;

                numEmptyCells++;

            }

        }

    }

    if (numEmptyCells > 0) {

        int randomIndex = rand() % numEmptyCells;

        int randomValue = (rand() % 2 + 1) \* 2;

        int x = emptyCells[randomIndex][0];

        int y = emptyCells[randomIndex][1];

        board[x][y] = randomValue;

    }

}

//Checks if the Game is Over or not

int isGameOver() {

    int i, j;

    // Check for 2048 tile

    for (i = 0; i < SIZE; i++) {

        for (j = 0; j < SIZE; j++) {

            if (board[i][j] == 2048) {

                return 1;

            }

        }

    }

    // Check for empty cell

    for (i = 0; i < SIZE; i++) {

        for (j = 0; j < SIZE; j++) {

            if (board[i][j] == 0) {

                return 0;

            }

        }

    }

    // Check for neighboring cells with the same value

    for (i = 0; i < SIZE - 1; i++) {

        for (j = 0; j < SIZE - 1; j++) {

            if (board[i][j] == board[i][j + 1] || board[i][j] == board[i + 1][j]) {

                return 0;

            }

        }

    }

    return 1;

}

//Function to merge tiles into one

void mergeTiles(int row[], int size) {

    int i, j;

    for (i = 0; i < size - 1; i++) {

        //Adjacent tiles are same

        if (row[i] == row[i + 1]) {

            row[i] \*= 2;

            row[i + 1] = 0;

        }

    }

}

//Function to shift the tiles left

void shiftTilesLeft(int row[], int size) {

    int i, j;

    for (i = 0; i < size; i++) {

        if (row[i] == 0) {

            for (j = i; j < size - 1; j++) {

                row[j] = row[j + 1];

            }

            row[size - 1] = 0;

        }

    }

}

//Function to instruct to move left

void moveLeft() {

    int i;

    for (i = 0; i < SIZE; i++) {

        mergeTiles(board[i], SIZE);

        shiftTilesLeft(board[i], SIZE);

    }

}

//Function to reverse

void reverse(int row[], int size) {

    int i, j, temp;

    for (i = 0, j = size - 1; i < j; i++, j--) {

        temp = row[i];

        row[i] = row[j];

        row[j] = temp;

    }

}

//Function to transpose the board

void transposeBoard() {

    int i, j, temp;

    for (i = 0; i < SIZE; i++) {

        for (j = i + 1; j < SIZE; j++) {

            temp = board[i][j];

            board[i][j] = board[j][i];

            board[j][i] = temp;

        }

    }

}

//Function to instruct to move right

void moveRight() {

    int i;

    for (i = 0; i < SIZE; i++) {

        reverse(board[i], SIZE);

        mergeTiles(board[i], SIZE);

        shiftTilesLeft(board[i], SIZE);

        reverse(board[i], SIZE);

    }

}

//Function to instruct to move up

void moveUp() {

    transposeBoard();

    moveLeft();

    transposeBoard();

}

//Function to instruct to move down

void moveDown() {

    transposeBoard();

    moveRight();

    transposeBoard();

}

//Main function

int main() {

    srand(time(NULL));

    initializeBoard();

    addRandomTile();

    addRandomTile();

    printBoard();

    char move;

    while (1) {

        scanf(" %c", &move);

        if (move == 'a') {

            moveLeft();

        } else if (move == 'd') {

            moveRight();

        } else if (move == 'w') {

            moveUp();

        } else if (move == 's') {

            moveDown();

        } else {

            printf("Invalid move!\n");

            continue;

        }

        printf("\n");

        addRandomTile();

        printBoard();

        if (isGameOver()) {

            printf("Game Over!\n");

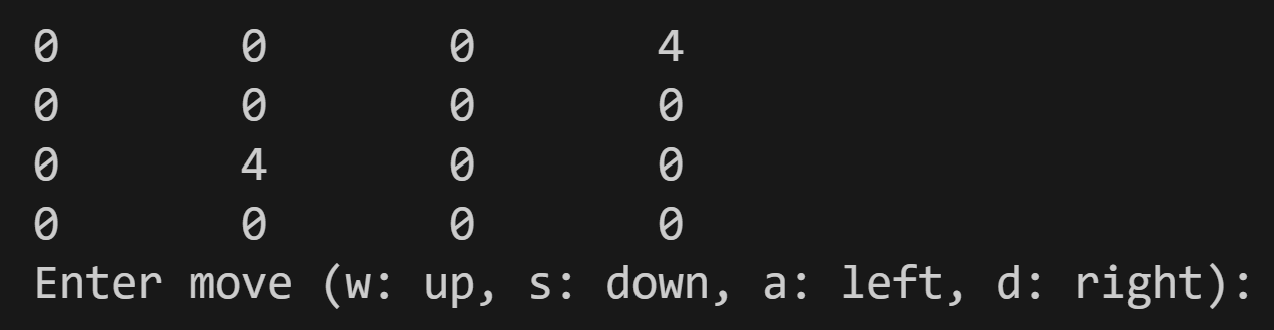
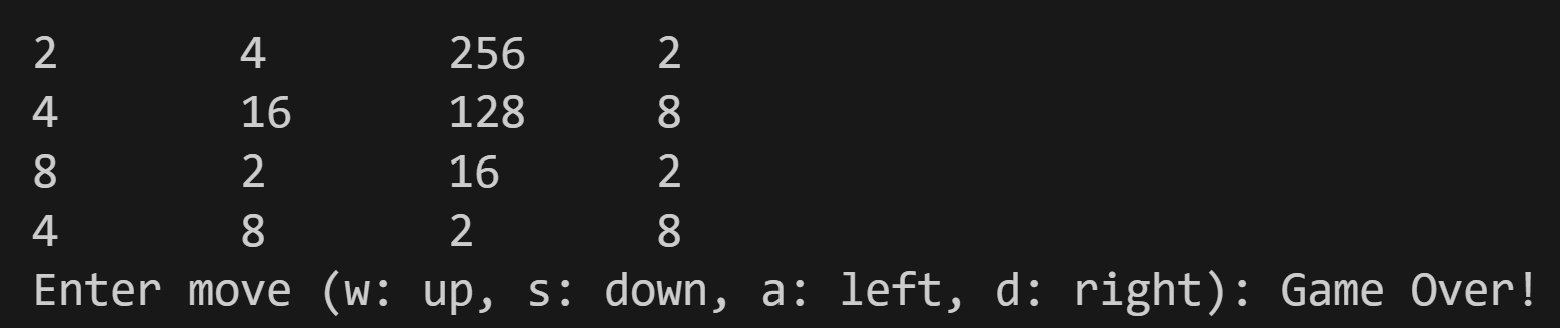
            break;

        }

    }

    return 0;

}

**Output Screenshots:**

**Data Structure Used:**

In the provided code, the data structure used is a two-dimensional array, specifically the array `board` of size **`SIZE x SIZE`**.

Arrays are an essential data structure in programming as they allow for efficient storage and access of multiple elements of the same type. In the context of the 2048 game, the array **`board`** represents the game board itself, where each element of the array represents a tile on the board.

The use of an array is particularly suitable for representing the game board because it provides a structured and organized way to store and manipulate the tiles. Each element of the array corresponds to a specific position on the board, and the values stored in the array represent the values of the tiles at those positions.

By using an array, various operations on the game board become more manageable. For example, initializing the board, printing the board, checking for game over conditions, merging and shifting tiles, and applying movements in different directions are made easier through array manipulations. The array allows for easy indexing and updating of tile values based on their positions.

Overall, the array data structure facilitates the representation, manipulation, and traversal of the game board, providing a convenient way to manage the state and progress of the 2048 game.