How to Compile and Run

To compile the program, use the provided Makefile. It compiles the source files game2048.c and utility.c into an executable. The Makefile uses the GCC compiler with warnings enabled (-Wall). Just open a terminal in the project folder and type make. This will create an executable called game2048. To run the game, type "./game2048" in the terminal. If you want to clean up the folder by removing compiled files, run "make clean".

How to Play

The goal is to combine tiles on a 4x4 grid to get a tile with the number 2048. The game starts with two tiles, either 2 or 4, placed randomly. Use the keys W, A, S, and D to move tiles up, left, down, and right, respectively. When two tiles with the same number collide during a move, they merge into one tile with double the value. After each move, a new tile (mostly 2, sometimes 4) appears randomly on the board. The game ends when you reach 2048 or when there are no more moves left. Press Q anytime to quit.

How The Program Works

The main logic is in game2048.c. It handles the board, user input, tile movements, merges, and spawning new tiles. The file utility.c has helper functions, including one to read keyboard input without needing Enter. The board is stored as a 4x4 grid of numbers. Moves slide and combine tiles according to the rules. After every valid move, new tiles appear in empty spots. The program also checks if moves are still possible to know when the game is over. The board is displayed in the terminal using simple text.

Appendix

game2048.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "utility.c"
#include "utility.h"
#include <stdbool.h>

void print_board(int board[4][4]);
void move_left(int board[4][4]);
void move_right(int board[4][4]);
```

```c
void move_up(int board[4][4]);
void move_down(int board[4][4]);
bool random_spawn(int board[4][4]);
bool moves_available(int board[4][4]);
void game(char input, int board[4][4]);

void print_board(int board[4][4])
{
    printf("\n-----------------------\n");
    for (int i = 0; i < 4; i++)
    {
        printf("|");
        for (int j = 0; j < 4; j++)
        {
            if (board[i][j] == 0)
                printf("    .");
            else
                printf("%5d", board[i][j]);
        }
        printf(" |\n");
    }
    printf("-----------------------\n");
}

void move_left(int board[4][4])
{
    for (int i = 0; i < 4; i++) // repeat for each row
    {
        int temp_row[4] = {0};
        int index = 0;

        // 1. slide non-zero tiles
        for (int j = 0; j < 4; j++)
        {
            if (board[i][j] != 0)
            {
                temp_row[index] = board[i][j];
                index++;
            }
        }

        // 2. combine same tiles
        for (int j = 0; j < 3; j++)
        {
            if (temp_row[j] != 0 && temp_row[j] == temp_row[j + 1])
            {
                temp_row[j] *= 2;
                temp_row[j + 1] = 0;
```

```
            }
        }

        // 3. slide non-zero tiles again and assign to actual board row
        index = 0;
        for (int j = 0; j < 4; j++)
        {
            if (temp_row[j] != 0)
            {
                board[i][index] = temp_row[j];
                index++;
            }
        }
        while (index < 4) // make sure the rest of the row is zero
        {
            board[i][index] = 0;
            index++;
        }
    }
}

void move_right(int board[4][4])
{
    for (int i = 0; i < 4; i++)
    {
        int temp_row[4] = {0};
        int index = 3;

        // slide right to collect non-zero values
        for (int j = 3; j >= 0; j--)
        {
            if (board[i][j] != 0)
            {
                temp_row[index] = board[i][j];
                index--;
            }
        }

        // combine equal tiles
        for (int j = 3; j > 0; j--)
        {
            if (temp_row[j] != 0 && temp_row[j] == temp_row[j - 1])
            {
                temp_row[j] *= 2;
                temp_row[j - 1] = 0;
            }
        }
```

```
        // slide again to fill gaps
        int final_row[4] = {0};
        index = 3;
        for (int j = 3; j >= 0; j--)
        {
            if (temp_row[j] != 0)
            {
                final_row[index] = temp_row[j];
                index--;
            }
        }

        // copy to board
        for (int j = 0; j < 4; j++)
            board[i][j] = final_row[j];
    }
}

void move_up(int board[4][4])
{
    for (int j = 0; j < 4; j++)
    {
        int temp_col[4] = {0};
        int index = 0;

        for (int i = 0; i < 4; i++)
        {
            if (board[i][j] != 0)
            {
                temp_col[index] = board[i][j];
                index++;
            }
        }

        for (int i = 0; i < 3; i++)
        {
            if (temp_col[i] != 0 && temp_col[i] == temp_col[i + 1])
            {
                temp_col[i] *= 2;
                temp_col[i + 1] = 0;
            }
        }

        int final_col[4] = {0};
        index = 0;
        for (int i = 0; i < 4; i++)
        {
            if (temp_col[i] != 0)
```

```
            {
                final_col[index] = temp_col[i];
                index++;
            }
        }

        for (int i = 0; i < 4; i++)
            board[i][j] = final_col[i];
    }
}

void move_down(int board[4][4])
{
    for (int j = 0; j < 4; j++)
    {
        int temp_col[4] = {0};
        int index = 3;

        for (int i = 3; i >= 0; i--)
        {
            if (board[i][j] != 0)
            {
                temp_col[index] = board[i][j];
                index--;
            }
        }

        for (int i = 3; i > 0; i--)
        {
            if (temp_col[i] != 0 && temp_col[i] == temp_col[i - 1])
            {
                temp_col[i] *= 2;
                temp_col[i - 1] = 0;
            }
        }

        int final_col[4] = {0};
        index = 3;
        for (int i = 3; i >= 0; i--)
        {
            if (temp_col[i] != 0)
            {
                final_col[index] = temp_col[i];
                index--;
            }
        }

        for (int i = 0; i < 4; i++)
```

```
            board[i][j] = final_col[i];
    }
}

bool random_spawn(int board[4][4])
{
    int empty_cells[16][2] = {0};
    int empty_cells_count = 0;

    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (board[i][j] == 0)
            {
                empty_cells[empty_cells_count][0] = i;
                empty_cells[empty_cells_count][1] = j;
                empty_cells_count++;
            }
        }
    }

    int spawn_count = empty_cells_count;
    if (empty_cells_count > 2)
        spawn_count = 2;

    for (int s = 0; s < spawn_count; s++) // made it a loop so i don't have to rewrite
it for 1 vs 2 empty spots
    {
        int rand_pos = rand() % empty_cells_count;
        int i = empty_cells[rand_pos][0], j = empty_cells[rand_pos][1];
        int two_or_four = (rand() % 10 == 0) ? 4 : 2; // 10% chance of 4
        board[i][j] = two_or_four;
        empty_cells[rand_pos][0] = empty_cells[empty_cells_count - 1][0];
        empty_cells[rand_pos][1] = empty_cells[empty_cells_count - 1][1];
        empty_cells_count--;
    }

    return empty_cells_count == 0;
}

bool moves_available(int board[4][4])
{
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (board[i][j] == 0)
```

```
                return true;
            if (j < 3 && board[i][j] == board[i][j + 1])
                return true;
            if (i < 3 && board[i][j] == board[i + 1][j])
                return true;
        }
    }
    return false;
}

void game(char input, int board[4][4])
{
    switch (input)
    {
    case 'a': // left
        move_left(board);
        break;
    case 'd': // right
        move_right(board);
        break;
    case 'w': // up
        move_up(board);
        break;
    case 's': // down
        move_down(board);
        break;
    default:
        break;
    }
}

int main()
{
    srand(time(NULL));
    int board[4][4] = {0};
    bool board_full = false;

    // spawn two initial tiles at game start
    random_spawn(board);

    while (1)
    {
        print_board(board);
        printf("Use WASD to move, Q to quit: ");

        char input = getch_unix();
        printf("%c\n", input);
```

```
        // convert uppercase input to lowercase
        if (input >= 'A' && input <= 'Z')
            input = input + ('a' - 'A');

        // check if input valid
        if (input != 'w' && input != 'a' && input != 's' && input != 'd' && input !=
'q')
            continue;

        if (input == 'q')
            break;

        // copy board to check if move changes anything
        int board_copy[4][4];
        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 4; j++)
                board_copy[i][j] = board[i][j];

        // do move
        game(input, board);

        // check if board changed (don't spawn more if no move)
        bool moved = false;
        for (int i = 0; i < 4 && !moved; i++)
            for (int j = 0; j < 4; j++)
                if (board[i][j] != board_copy[i][j])
                {
                    moved = true;
                    break;
                }

        if (moved)
        {
            board_full = random_spawn(board);
        }

        // check if game over
        if (board_full && !moves_available(board))
        {
            print_board(board);
            printf("Game Over! No moves left.\n");
            break;
        }

        bool won = false;
        for (int i = 0; i < 4 && !won; i++)
            for (int j = 0; j < 4; j++)
                if (board[i][j] == 2048)
```

```
                    won = true;

        if (won)
        {
            print_board(board);
            printf("You reached 2048! You win!\n");
            break;
        }
    }

    return 0;
}
```

utility.c

```
// CODE: Include necessary librar(y/ies)
#include <termios.h> // used in getch_unix()
#include <unistd.h>  // used in getch_unix()


// CODE: the implementation of functions and related variables


/*Receiving an input from keyboard
IMPORTANT: DO NOT CHANGE ANYTHING IN getch_unix() FUNCTION.
To test this function and see if it works on your OS you can use testMyInput.c.
It should work on any UNIX based OS.
This function receives all possible charaters from a keyboard.
The returned character `ch` will be used in "your game logic" as:
if ch='w' // Move Up
if ch='s' // Move Down
if ch='a' // Move Left
if ch='d' // Move Right
if ch='q' // break the game loop and quit
otherwise  do nothing
*/
char getch_unix() {
    struct termios oldt, newt;
    char ch;
    tcgetattr(STDIN_FILENO, &oldt); // Gets the current terminal settings
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    ch = getchar();
    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
    return ch;
```

```
}
```