T4 - 2048

Nome:
Leonardo Guarnieri de Bastiani
Engenharia de Computação

N° USP:
8910434

Arquivos:

Os arquivos que são encontrados nesse trabalho são:

- **2048.h** - Contém todos os includes necessários no programa e as declarações de variáveis globais e funções que serão utilizadas ao longo do código.
- **main.c** - Contém a função **main** do programa.
- **variables.c** - Contém todas as variáveis globais utilizadas no programa.
- **screen.c** - Contém todas as funções responsáveis pelo gráfico do programa.
- **animation.c** - Contém as funções responsáveis pelas animações visuais do programa.
- **logic.c** - Contém a parte lógica do programa, trata sobre os cálculos realizados pelo programa para que o jogo funcione.

O programa basicamente utiliza funções que fazem todas as operações necessárias pelo jogo, analisaremos cada função de maneira que ficará fácil entender o jogo.

variables.c e 2048.h

```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <unistd.h>

//---------------------------------------
// VARIABLES
//---------------------------------------
#define KEY_RIGHT   1
#define KEY_UP      2
#define KEY_LEFT    3
#define KEY_DOWN    4
#define KEY_ESC     27

extern char screen[20][71];

struct Game {
    int tabuleiro[4][4];
    int score;
    char win;
    int turn;
};

extern struct Game game;
extern struct Game undo[20];

extern char combinated[4][4];

extern char borders[11];
extern int numborders;

//---------------------------------------
// SCREEN
//---------------------------------------
int log_base2(int num);
void write_str(int x, int y, char *str);
void render_border(int x, int y, char border);
void render_number(int x, int y, int number);
void render_line(int x);
void clear();
void render_game();
void render_score();
void draw();
void render();
//---------------------------------------
// ANIMATION
//---------------------------------------
void clear_block(int x, int y);
void animation_newblock(int x, int y);
//---------------------------------------
// INPUT
//---------------------------------------
int button();
//---------------------------------------
// LOGIC
//---------------------------------------
int new_block(int create);
int move_block(int i, int j, int x, int y, int execute);
void clear_combinated();
void set_line_col(int line[], int col[], int x, int y);
int move(int dir, int execute);
void undo_move();
int endgame();
```

Nas linhas de 1 a 4 são feitos os includes para as bibliotecas padrões.

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <time.h>
4 #include <unistd.h>
```

<time.h> foi necessária para gerar uma seed para números randômicos.

```
1 #include <time.h>
```

<unistd.h> foi necessária para as animações, é nela que está definida a função de sleep.

```
1 #include <unistd.h>
```

Os defines estão presentes apenas para seguir um padrão. Como as teclas utilizadas foram WASD, não foi necessário inserir os valores reais das setas, porém, para implementar as setas no programa ficaria muito fácil se os valores estivessem inseridos corretamente. Como o foco é WASD, assim está ótimo.

```
1 #define KEY_RIGHT   1
2 #define KEY_UP      2
3 #define KEY_LEFT    3
4 #define KEY_DOWN    4
5 #define KEY_ESC     27
```

A variável **screen** contém toda a tela que é apagada e reescrita pra cada atualização feita no jogo.

```
1 extern char screen[20][71];
```

A **struct Game** foi feita para armazenar as informações da partida, com ela foi feita a variável **game** e **undo**. A variável **game** possui as informações atuais do jogo, a variável **undo** é um backup de **game** para cada movimento, assim, é só resgatar as informações de **undo** para que **game** volte algumas jogadas.

```
1  struct Game {
2      int tabuleiro[4][4];
3      int score;
4      char win;
5      int turn;
6  };
7
8  extern struct Game game;
9  extern struct Game undo[20];
10
11 extern char combinated[4][4];
```

A variável **combinated** informa ao jogo quais peças já combinaram.

```
1 extern char combinated[4][4];
```

A variável **borders** contém todas as bordas das peças do jogo e **numborders**, o número de bordas que **borders** possui.

```
1 extern char borders[11];
2 extern int numborders;
```

No arquivo **variables.c**, todas essa variáveis estão definidas. Já no **2048.h**, elas estão declaradas.

screen.c

```
1  #include "2048.h"
2
3  int log_base2(int num) {
4      int i = 0;
5      while(num != 0) {
6          num /= 2;
7          i++;
8      }
9      return i-1;
10 }
11
12 void write_str(int x, int y, char *str) {
13     while(*str) {
14         screen[x][y++] = *str++;
15     }
16 }
17
18 void render_border(int x, int y, char border) {
19     int i, j;
20     i = x;
21     j = y;
22
23     for(j=y; j<y+7; j++) {
24         screen[i][j] = border;
25     }
26     i++;
27
28     j = y;
29     while(i<=x+3) {
30         screen[i][j] = border;
```

```
        screen[i][j+6] = border;
        i++;
    }

    for(j=y; j<y+7; j++) {
        screen[i][j] = border;
    }
}

void render_block(int x, int y, int number) {
    char str[5];
    char border;
    int i, j;


    if(number) {
        border = borders[(log_base2(number)-1) % numborders];
        sprintf(str, "%d", number);

        i = x+2;
        j = y+2;
        write_str(i, j, str);
    } else {
        border = '.';
    }

    render_border(x, y, border);
}

void render_line(int i) {
    int j;
    for(j=0; j<4; j++) {
        render_block(i*5, j*7, game.tabuleiro[i][j]);
    }
}


void clear() {
    int i, j;
    for(i = 0; i < 20; i++) {
        for(j = 0; j < 70; j++) {
            screen[i][j] = ' ';
        }
        screen[i][j] = '\n';
    }
    screen[i][j] = '\0';
}

void render_game() {
    int i;
    for(i=0;i<4;i++) {
        render_line(i);
    }
}

void render_score() {
    char str[9];
    sprintf(str, "%d", game.score);
    write_str(5, 40, "SCORE");
    write_str(6, 40, str);
}

void draw() {
    system("clear");
    printf("%s", (char *) screen);
}

void render() {
    clear();
    render_game();
    render_score();
}
```

A função **log_base2** recebe um número e retorna o inteiro do logarítmo dele na base 2.

```
int log_base2(int num) {
    int i = 0;
    while(num != 0) {
        num /= 2;
        i++;
    }
    return i-1;
}
```

A função **write_str** recebe os valores **x** e **y**, que são posições na tela, e altera a variável screen, de caractere em caractere, até que encontre o \0.

```
void write_str(int x, int y, char *str) {
    while(*str) {
        screen[x][y++] = *str++;
    }
```

```
}
```

A função **render_border** é responsável por desenhar a borda de cada peça na variável **screen**.

```
void render_border(int x, int y, char border) {
    int i, j;
    i = x;
    j = y;

    for(j=y; j<y+7; j++) {
        screen[i][j] = border;
    }
    i++;

    j = y;
    while(i<=x+3) {
        screen[i][j] = border;
        screen[i][j+6] = border;
        i++;
    }

    for(j=y; j<y+7; j++) {
        screen[i][j] = border;
    }
}
```

A função **render_block** é responsável por escrever o número e sua borda na variável **screen**. Nela foi utilizada a função **sprintf** que altera a variável **str** para receber algo como se fosse **printf**("%d", number);.

```
void render_block(int x, int y, int number) {
    char str[5];
    char border;
    int i, j;


    if(number) {
        border = borders[(log_base2(number)-1) % numborders];
        sprintf(str, "%d", number);

        i = x+2;
        j = y+2;
        write_str(i, j, str);
    } else {
        border = '.';
    }

    render_border(x, y, border);
}
```

A função **render_line** chama **render_block** para cada coluna de **i**, assim, ao chamar essa função, ele altera a variável **screen** com a linha do tabuleiro.

```
void render_line(int i) {
    int j;
    for(j=0; j<4; j++) {
        render_block(i*5, j*7, game.tabuleiro[i][j]);
    }
}
```

A função **clear** altera a variável **screen** com espaços em brancos, como se fosse apagá-la.

```
void clear() {
    int i, j;
    for(i = 0; i < 20; i++) {
        for(j = 0; j < 70; j++) {
            screen[i][j] = ' ';
        }
        screen[i][j] = '\n';
    }
    screen[i][j] = '\0';
}
```

A função **render_game** chama 4 vezes a função **render_line**, renderizando todas as linhas de **game.tabuleiro.**

```
void render_game() {
    int i;
    for(i=0;i<4;i++) {
        render_line(i);
    }
}
```

A função **render_score** altera a variável **screen** escrevendo nela "SCORE" e a pontuação do jogador logo embaixo, foi necessária a utilização de **sprintf** para converter a variável **game.score** em uma **string**.

```
void render_score() {
    char str[9];
    sprintf(str, "%d", game.score);
    write_str(5, 40, "SCORE");
    write_str(6, 40, str);
}
```

A função **draw** apaga toda a tela com **system**("clear") e imprime todo o conteúdo de **screen**. Foi necessário um **cast** para converter **screen** de um ponteiro para ponteiros de char, ou seja, uma matriz, para um ponteiro para char, ou seja, um vetor.

```
void draw() {
    system("clear");
    printf("%s", (char *) screen);
}
```

A função **render** limpa a variável **screen**, desenhando o jogo nela e a pontuação.

```
void render() {
    clear();
    render_game();
    render_score();
}
```

input.c

```
#include "2048.h"

int button() {
    char letter;
    char result = 0;
    while(!result) {
        letter = toupper(getchar()); // toupper pra deixar em maíscula
        switch(letter) {
            case 'D':
                result = KEY_RIGHT;
                break;
            case 'W':
                result = KEY_UP;
                break;
            case 'A':
                result = KEY_LEFT;
                break;
            case 'S':
                result = KEY_DOWN;
                break;
            case 'U':
                undo_move();
                break;
            case KEY_ESC:
                exit(0);
        }
    }
    return result;
}
```

Neste arquivo temos apenas uma função, ela é responsável pela entrada do teclado.

A variável **result** recebe um valor aceitável, sendo este 'A', 'S', 'D', 'W', 'U' ou ESC que dentro do **switch** alterando o valor dela e assim deixando de ser 0 e escapando do laço.

animation.c

```
#include "2048.h"

void clear_block(int x, int y) {
    int i, j;
    for(i=x;i<x+5;i++) {
        for(j=y;j<y+7;j++) {
            screen[i][j] = ' ';
        }
    }
}

void animation_newblock(int i, int j) {
    int x, y;
    int frame;

    x = i*5;
    y = j*7;

    render();
    draw();

    for(frame=0;frame<4;frame++) {
        int incx, incy;
        clear_block(x, y);
        switch(frame) {
            case 0:
                incx = 2;
                incy = 3;
                break;
            case 1:
                incx = 1;
                incy = 2;
```

```
                break;
            case 2:
                incx = 1;
                incy = 1;
                break;
            case 3:
                incx = 0;
                incy = 0;
                break;
        }

        screen[x+incx][y+incy] = '#';
        screen[x+incx][y+(6-incy)] = '#';
        screen[x+(4-incx)][y+(6-incy)] = '#';
        screen[x+(4-incx)][y+incy] = '#';
        draw();
        usleep(40000);
    }
}
```

A função **clear_block** altera a variável **screen** com espaços em brancos de modo a formar um quadrado.

```
void clear_block(int x, int y) {
    int i, j;
    for(i=x;i<x+5;i++) {
        for(j=y;j<y+7;j++) {
            screen[i][j] = ' ';
        }
    }
}
```

A função **animation_newblock** faz uma animação na tela com um novo bloco, ela usa a função **clear_block**, coloca caracteres na variável **screen**, só que dessa vez a função **draw** é chamada num intervalo de 40 microssegundos através da função **usleep** que deixa o processador sem executar comandos pelo tempo no parâmetro.

```
void animation_newblock(int i, int j) {
    int x, y;
    int frame;

    x = i*5;
    y = j*7;

    render();
    draw();

    for(frame=0;frame<4;frame++) {
        int incx, incy;
        clear_block(x, y);
        switch(frame) {
            case 0:
                incx = 2;
                incy = 3;
                break;
            case 1:
                incx = 1;
                incy = 2;
                break;
            case 2:
                incx = 1;
                incy = 1;
                break;
            case 3:
                incx = 0;
                incy = 0;
                break;
        }

        screen[x+incx][y+incy] = '#';
        screen[x+incx][y+(6-incy)] = '#';
        screen[x+(4-incx)][y+(6-incy)] = '#';
        screen[x+(4-incx)][y+incy] = '#';
        draw();
        usleep(40000);
    }
}
```

logic.c

```
#include "2048.h"

int new_block(int create) {
    int i, j;
    int numzeros = 0;
    int pos;
    int value; // valor do bloco

    for(i=0; i < 4; i++) {
        for(j=0; j < 4; j++) {
```

```
            if(game.tabuleiro[i][j] == 0) numzeros++;
        }
    }

    if(!numzeros || !create) return numzeros;

    pos = rand() % numzeros;
    value = (rand() % 10) ? 2 : 4; // 10% de chance de cair um 4

    int count_zeros = 0;
    i = 0;
    while(count_zeros != pos || game.tabuleiro[i/4][i%4] != 0) {
        if(game.tabuleiro[i/4][i%4] == 0) count_zeros++;
        i++;
    }

    game.tabuleiro[i/4][i%4] = value;

    animation_newblock(i/4, i%4);

    return numzeros;
}

void win() {
    game.win = 1;
    system("clear");
    puts("Parabens!!\n");
    getchar();
}

int move_block(int i, int j, int x, int y, int execute) {
    int newi, newj;

    newi = i+y;
    newj = j+x;

    if(newi < 0 || newi >= 4) return 0; // nao pode escapar do tabuleiro
    if(newj < 0 || newj >= 4) return 0;

    if(game.tabuleiro[newi][newj] == 0) {
        if(!execute) return 1;
        game.tabuleiro[newi][newj] = game.tabuleiro[i][j];
        game.tabuleiro[i][j] = 0;
        move_block(newi, newj, x, y, execute);
        return 1;
    } else if(game.tabuleiro[newi][newj] == game.tabuleiro[i][j]) {
        if(combinated[newi][newj]) { // combina duas vezes
            return 0;
        } else if(!execute) {
            return 1;
        }
        combinated[newi][newj] = 1;
        game.tabuleiro[newi][newj] = 2*game.tabuleiro[i][j];
        game.score += game.tabuleiro[newi][newj];
        game.tabuleiro[i][j] = 0;
        if(game.tabuleiro[newi][newj] == 2048 && game.win == 0) {
            win();
        }
        return 1;
    }

    return 0;
}

void clear_combinated() {
    int i, j;
    for(i=0;i<4;i++) {
        for(j=0;j<4;j++) {
            combinated[i][j] = 0;
        }
    }
}

void set_line_col(int line[], int col[], int x, int y) {
    int i;
    if(y == 1) { // de baixo para cima
        for(i=4; i>0; i--) line[4-i] = i-1;
    } else { // de cima para baixo
        for(i=0; i<4; i++) line[i] = i;
    }

    if(x == 1) { // da direita para a esquerda
        for(i=4; i>0; i--) col[4-i] = i-1;
    } else { // da esquerda para a direita
        for(i=0; i<4; i++) col[i] = i;
    }
}

int move(int dir, int execute) {
    int x = 0, y = 0, i, j; // coordenadas para andar
    int line[4], col[4];
```

```
    int nummoves = 0;
    switch(dir) {
        case KEY_RIGHT:
            x = 1;
            break;
        case KEY_LEFT:
            x = -1;
            break;
        case KEY_UP:
            y = -1;
            break;
        case KEY_DOWN:
            y = 1;
            break;
    }

    clear_combinated();
    set_line_col(line, col, x, y);

    for(i=0; i<4; i++) {
        int li;
        int co;
        for(j=0; j<4; j++) {
            li = line[i];
            co = col[j];
            if(game.tabuleiro[li][co]) {
                nummoves += move_block(li, co, x, y, execute); // retorna se mveu
            }
        }
    }

    return nummoves;
}

void undo_move() {
    if(game.turn > 0) {
        game = undo[(game.turn-1) % 20];
    }
    render();
    draw();
}

int endgame() {
    return  !( move(KEY_LEFT    , 0)
        || move(KEY_UP      , 0)
        || move(KEY_RIGHT   , 0)
        || move(KEY_DOWN    , 0));
}
```

A função **new_block** cria um novo bloco em **game.tabuleiro**, o parâmetro **create** diz se é para se realmente criar um novo bloco ou apenas simular a criação de um. Ela retorna a quantidade de zeros no tabuleiro.

A variável **numzeros** conta a quantidade de zeros no tabuleiro, **pos** me diz em qual zero será posto o novo bloco, **value** é o valor do bloco (2 ou 4). O **while** da função foi verbalizado pela seguinte frase: "Eu devo contar um zero e avançar a casa enquanto eu não cheguei no n-ésimo zero e a casa em que estou não é zero". Por fim, é posto **value** na posição do tabuleiro encontrada e é chamada a animação para novo bloco.

```
int new_block(int create) {
    int i, j;
    int numzeros = 0;
    int pos;
    int value; // valor do bloco

    for(i=0; i < 4; i++) {
        for(j=0; j < 4; j++) {
            if(game.tabuleiro[i][j] == 0) numzeros++;
        }
    }

    if(!numzeros || !create) return numzeros;

    pos = rand() % numzeros;
    value = (rand() % 10) ? 2 : 4; // 10% de chance de cair um 4

    int count_zeros = 0;
    i = 0;
    while(count_zeros != pos || game.tabuleiro[i/4][i%4] != 0) {
        if(game.tabuleiro[i/4][i%4] == 0) count_zeros++;
        i++;
    }

    game.tabuleiro[i/4][i%4] = value;

    animation_newblock(i/4, i%4);

    return numzeros;
}
```

A função **win** é chamada quando for formada uma peça de valor 2048.

```
void win() {
```

```
    game.win = 1;
    system("clear");
    puts("Parabens!!\n");
    getchar();
}
```

A função **move_block** move apenas um bloco, ela é usada de maneira recursiva e se **execute** for 0, ela faz apenas uma simulação da execução. Apesar de **move_block** mover um bloco, não é informado a tecla inserida por parâmetro.

```
int move_block(int i, int j, int x, int y, int execute) {
    int newi, newj;

    newi = i+y;
    newj = j+x;

    if(newi < 0 || newi >= 4) return 0; // nao pode escapar do tabuleiro
    if(newj < 0 || newj >= 4) return 0;

    if(game.tabuleiro[newi][newj] == 0) {
        if(!execute) return 1;
        game.tabuleiro[newi][newj] = game.tabuleiro[i][j];
        game.tabuleiro[i][j] = 0;
        move_block(newi, newj, x, y, execute);
        return 1;
    } else if(game.tabuleiro[newi][newj] == game.tabuleiro[i][j]) {
        if(combinated[newi][newj]) { // combina duas vezes
            return 0;
        } else if(!execute) {
            return 1;
        }
        combinated[newi][newj] = 1;
        game.tabuleiro[newi][newj] = 2*game.tabuleiro[i][j];
        game.score += game.tabuleiro[newi][newj];
        game.tabuleiro[i][j] = 0;
        if(game.tabuleiro[newi][newj] == 2048 && game.win == 0) {
            win();
        }
        return 1;
    }

    return 0;
}
```

A função **clear_combinated** é utilizada para atribuir a variável **combinted** em todos os valores com 0.

```
void clear_combinated() {
    int i, j;
    for(i=0;i<4;i++) {
        for(j=0;j<4;j++) {
            combinated[i][j] = 0;
        }
    }
}
```

A função **set_line_col** altera os vetores **line** e **col** que são recebidos por parâmetro.

```
void set_line_col(int line[], int col[], int x, int y) {
    int i;
    if(y == 1) { // de baixo para cima
        for(i=4; i>0; i--) line[4-i] = i-1;
    } else { // de cima para baixo
        for(i=0; i<4; i++) line[i] = i;
    }

    if(x == 1) { // da direita para a esquerda
        for(i=4; i>0; i--) col[4-i] = i-1;
    } else { // da esquerda para a direita
        for(i=0; i<4; i++) col[i] = i;
    }
}
```

A função **move** é mais genérica que a **move_block**, ela recebe a tecla que informa a direção da execução e se **execute** for 0, ela faz apenas uma simulação da execução. As variáveis **x** e **y** informam como será feita a nova posição da peça, i.e., se **x** for 1, ela anda uma casa para a direita, se for -1, anda uma casa para a esquerda. As variáveis **line** e **col** são utilizados dentro dos laços, e.g., se é apertado 'D', **line** vale {0, 1, 2, 3} e **col** vale {3, 2, 1, 0}.

```
int move(int dir, int execute) {
    int x = 0, y = 0, i, j; // coordenadas para andar
    int line[4], col[4];
    int nummoves = 0;
    switch(dir) {
        case KEY_RIGHT:
            x = 1;
            break;
        case KEY_LEFT:
            x = -1;
            break;
        case KEY_UP:
            y = -1;
```

```
            break;
        case KEY_DOWN:
            y = 1;
            break;
    }

    clear_combinated();
    set_line_col(line, col, x, y);

    for(i=0; i<4; i++) {
        int li;
        int co;
        for(j=0; j<4; j++) {
            li = line[i];
            co = col[j];
            if(game.tabuleiro[li][co]) {
                nummoves += move_block(li, co, x, y, execute); // retorna se mveu
            }
        }
    }

    return nummoves;
}
```

A função **undo_move** atribui **game** com o movimento anterior.

```
void undo_move() {
    if(game.turn > 0) {
        game = undo[(game.turn-1) % 20];
    }
    render();
    draw();
}
```

A função **endgame** retorna 1 se não há movimentos possíveis, ela simula os movimentos para cima, esquerda, baixo e direita.

```
int endgame() {
    return  !( move(KEY_LEFT    , 0)
        || move(KEY_UP      , 0)
        || move(KEY_RIGHT   , 0)
        || move(KEY_DOWN    , 0));
}
```

main.c

```
#include "2048.h"

int main() {

    srand(time(0));

    system("stty cbreak"); // não sei bem o que isso faz, mas é por causa dele que getchar não espera o enter

    new_block(1); // cria um novo bloco
    while(!endgame()) { // enquanto não é o fim do jogo, faça
        new_block(1);
        render(); // renderiza o jogo
        draw(); // imprime o jogo

        undo[(game.turn) % 20] = game; // cópia de game em undo

        //insiste enquanto não moveu e é possível criar um novo bloco
        while(!move(button(), 1) && new_block(0)) {
            // nao faz nada
        }
        game.turn++;
    }

    // derrota
    render();
    draw();
    puts("Fim de jogo!");
    while(button()); // espero apertar o botao esc

    return 0;
}
```

Nesta linha é atribuido uma seed com base na hora para gerar números aleatórios.

```
srand(time(0));
```

Não sei explicar aprofundamente o que esta linha faz, mas por causa dela, o terminal não espera que seja pressionado o ENTER do teclado, melhorando a gameplay do jogo.

```
system("stty cbreak"); // não sei bem o que isso faz, mas é por causa dele que getchar não espera o enter
```

Nesta linha é criado um novo bloco.

```
new_block(1); // cria um novo bloco
```

No laço em seguida, é feita uma verificação. Enquanto não é o fim do jogo, continue jogando.

```
while(!endgame()) { // enquanto não é o fim do jogo, faça
    new_block(1);
    render(); // renderiza o jogo
    draw(); // imprime o jogo

    undo[(game.turn) % 20] = game; // cópia de game em undo

    //insiste enquanto não moveu e é possível criar um novo bloco
    while(!move(button(), 1) && new_block(0)) {
        // nao faz nada
    }
    game.turn++;
}
```

Dentro do while, esta linha serve para criar um backup de **game** em **undo.**

```
undo[(game.turn) % 20] = game; // cópia de game em undo
```

O while que não faz nada foi verbalizado da seguinte maneira: "deve insistir enquanto não foi feito nenhum movimento e é possível criar um novo bloco".

```
while(!move(button(), 1) && new_block(0)) {
    // nao faz nada
}
```

No fim do programa, no laço que não faz nada, button retorna algo diferente de zero sempre, por isso o programa se encerra apenas quando é pressionado ESC.

```
while(button()); // espero apertar o botao esc
```