LERY Tetris

Eduardo de Azevêdo dos Santos
14/0136967

Luiz Felipe Carvalho Duarte
14/0151958

Rafael Cascardo Campos
14/0159401

Yero Távora
14/0166378

Brasília - DF, junho de 2016

Universidade de Brasília, Departamento de Ciência da Computação

1 Introdução

LERY Tetris é um jogo programado na linguagem C. Feito como projeto da disciplina de Programação Sistemática da Universidade de Brasília, o jogo ganha seu nome como uma sequência das iniciais de cada um dos autores de forma a ficar sonora. Segue o link do reposiório do GitHub utilizado para a confecção do programa:
https://github.com/gerudoking/lery-tetris.git
O gráfico do trabalho de cada membro do grupo pode ser encontrado aqui:
https://github.com/gerudoking/lery-tetris/graphs/contributors

2 Estrutura do Programa

O programa é composto de nove módulos, sendo quatro destes lógicos, contendo a interface das funções e a declaração dos registros, quatro sendo módulos físicos, com a implementação das funções, e um módulo contendo a função main do programa. O jogo utiliza da biblioteca ncurses para montar sua interface.

São módulos do programa os listados a seguir:

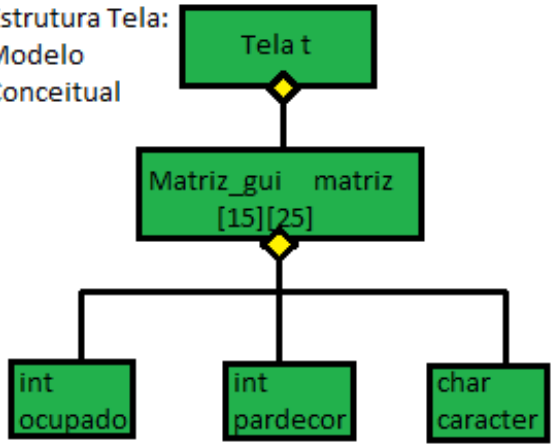
- Tela.h
- Tela.c
- Peca.h
- Peca.c
- Placar.h
- Placar.c
- Engine.h
- Engine.c
- main.c

Cada módulo será devidamente explicado em sua respectiva seção.

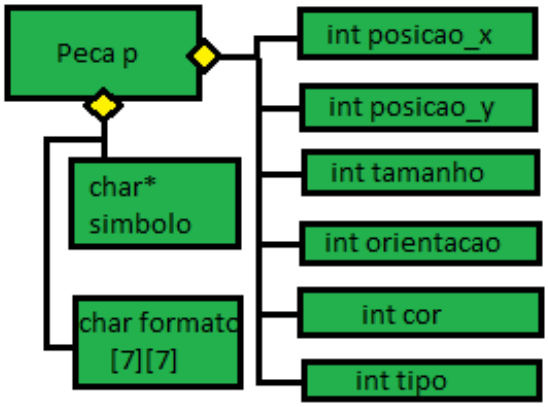
3 Modelos das Estruturas

Modelos das estruturas utilizadas no projeto:

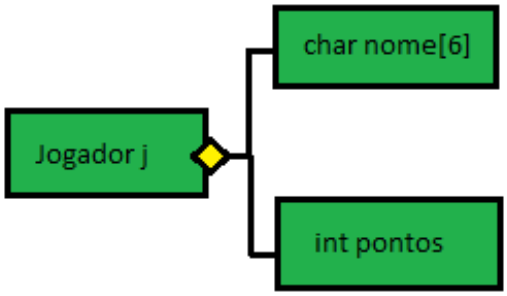
Estrutura Tela: Modelo Conceitual



Modelo Conceitual: Estrutura Peca



Modelo Conceitual: Estrutura Jogador



4 Tela

4.1 Tela.h

Arquivo do tipo header(.h) contendo a declaração da struct Tela e as interfaces das funções utilizadas no módulo Tela.c.
A struct Tela é composta por outra struct, chamada Tipomatriz. Esta, por sua vez, é composta de tres membros. Estes são:

- ocupado, do tipo inteiro;
- pardecor, do tipo inteiro;
- caracter, do tipo caractere.

As structs estão declaradas da seguinte maneira:

```
struct tipomatriz{
int ocupado;
int pardecor;
char caracter;
};
typedef struct tipomatriz Tipomatriz;

struct tela{
Tipomatriz matriz_gui[15][25];
};
typedef struct tela Tela
```

O módulo ainda possui seis interfaces:

```
WINDOW* cria_nova_janela(int alt, int larg, int starty, int startx);
```

Função que recebe quatro valores. Um correspondente a altura de uma janela padrão da biblioteca ncurses, outro a largura da mesma, e por fim os valores das posições iniciais y e x, respectivamente, da janela no terminal.

```
void destroi_janela(WINDOW* janela_local);
```

Em suma, recebe uma janela padrão da biblioteca ncurses e não possui retorno. Foi implementada porém não utilizada no programa.

```
Tela* cria_tela();
```

Funcão que não recebe parâmetros, retorna uma struct do tipo Tela.

```
int mostra_tela(Tela* t)
```

Esta função recebe uma struct do tipo Tela e retorna um valor do tipo inteiro(utilizado para testes).

```
int mostra_tela_inicial();
```

Funcão que não recebe parâmetros e retorna um valor do tipo inteiro(utilizado para testes).

```
int mostra_tela_final(int pont);
```

Funcão que recebe um inteiro representando a pontuação do jogador e retorna um inteiro(utilizado para testes).

```
int mostra_tela_placar(int pont);
```

Funcão que recebe um inteiro representando a pontuação do jogador e retorna um inteiro(utilizado para testes).

4.2 Tela.c

Arquivo do tipo .c, contém a implementação das funções declaradas em Tela.h.

cria_nova_janela: A função irá utilizar os parâmetros dados para criar uma janela padrão da biblioteca ncurses, para posteriormente desenhar uma caixa para servir como bordas da janela. Irá então dar "refresh"na janela para aplicar as mudanças. Por fim, retorna a janela criada.

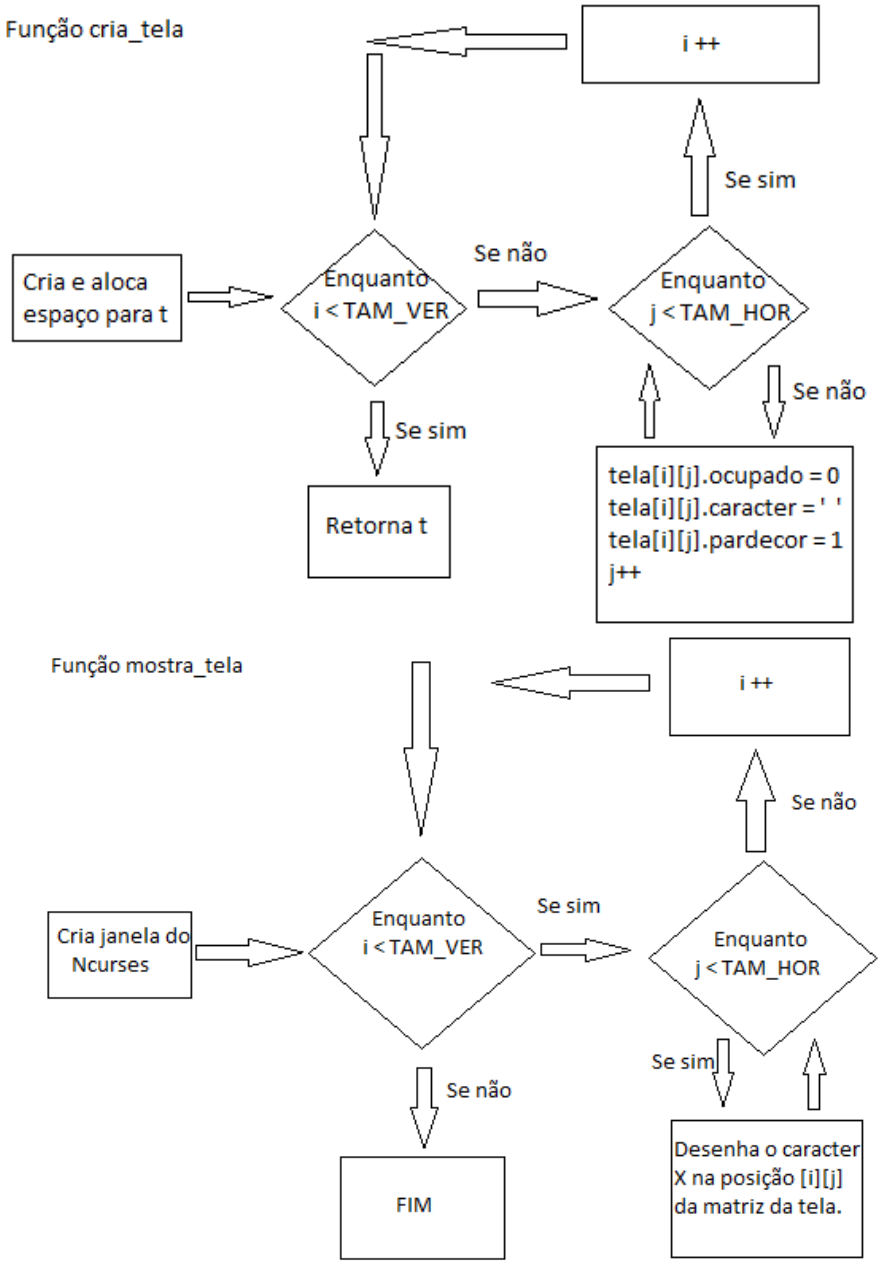
cria_tela: Esta função irá alocar espaço para a struct t do tipo Tela, testar seu sucesso, em caso de falha retornando NULL, e então irá definir cada campo de cada posição do membro matriz_gui de forma que o campo caracter seja um caractere espaço, o campo ocupado seja definido como zero, e o campo pardecor como um. Por fim retornará a tela criada.

mostra_tela: Função que recebe uma Tela t previamente criada. Começa limpando a tela com o comando refresh(), para então gerar uma janela padrão do ncurses para servir como bordas para a interface do jogo. Em um laço duplo, irá então reproduzir na tela os caracteres de cada posição da matriz_gui com suas respectivas cores, também definidas em matriz_gui.

mostra_tela_inicial: A função mostra_tela_inicial() nÃ£o recebe parâmetros. Esta irá criar uma tela t, mostrá-la, para então escrever por cima as mensagens de boas-vindas ao jogador em suas respectivas cores. Por fim, libera a estrutura t.

mostra_tela_final: Função idêntica a função mostra_tela_final, exceto que esta recebe um inteiro correspondente a pontuação do jogador e o imprime junto de sua respectiva mensagem.

4.3 Fluxogramas das Funções



4.4 Saída do Exame do GCov

```
-:    0:Source:Tela_Teste.c
-:    0:Graph:Tela_Teste.gcno
```

```
-:    0:Data:Tela_Teste.gcda
-:    0:Runs:1
-:    0:Programs:1
-:    1:#include "CUnit/CUnit.h"
-:    2:#include "CUnit/Basic.h"
-:    3:#include "Tela.h"
-:    4:
-:    5:void adicionar_suite(void);
-:    6:
-:    7:void teste_cria_tela(void);
-:    8:void teste_mostra_tela(void);
-:    9:
1:   10:void teste_cria_tela(void){
-:   11:        int resultado;
-:   12:        Tela* t;
1:   13:        t = cria_tela();
1:   14:        if (t == NULL) resultado = 0;
1:   15:        else resultado = 1;
1:   16:        CU_ASSERT_TRUE(resultado);
1:   17:        free(t);
1:   18:}
-:   19:
1:   20:void teste_mostra_tela(void){
-:   21:        int resultado;
-:   22:        Tela* t;
1:   23:        t = cria_tela();
1:   24:        resultado = mostra_tela(t);
1:   25:        CU_ASSERT_TRUE(!resultado);
1:   26:        free(t);
1:   27:}
-:   28:
1:   29:void teste_mostra_tela_inicial(void){
-:   30:        int resultado;
1:   31:        resultado = mostra_tela_inicial();
1:   32:        CU_ASSERT_TRUE(!resultado);
1:   33:}
-:   34:
1:   35:void teste_mostra_tela_final(void){
-:   36:        int resultado;
1:   37:        resultado = mostra_tela_final(100);
1:   38:        CU_ASSERT_TRUE(!resultado);
1:   39:}
-:   40:
```

```
        1:   41:void teste_mostra_tela_placar(void){
        -:   42:        int resultado;
        1:   43:        resultado = mostra_tela_placar(100);
        1:   44:        CU_ASSERT_TRUE(!resultado);
        1:   45:}
        -:   46:
        1:   47:void adicionar_suite(void){
        -:   48:        CU_pSuite suite;
        -:   49:
        1:   50:        suite = CU_add_suite("Testes",NULL,NULL);
        -:   51:
        1:   52:        CU_ADD_TEST(suite, teste_cria_tela);
        1:   53:        CU_ADD_TEST(suite, teste_mostra_tela);
        1:   54:        CU_ADD_TEST(suite, teste_mostra_tela_inicial);
        1:   55:        CU_ADD_TEST(suite, teste_mostra_tela_final);
        1:   56:        CU_ADD_TEST(suite, teste_mostra_tela_placar);
        -:   57:
        1:   58:}
        -:   59:
        1:   60:int main(){
        1:   61:        if (CUE_SUCCESS != CU_initialize_registry())
    #####:   62:                return CU_get_error();
        -:   63:
        1:   64:        adicionar_suite();
        -:   65:
        1:   66:        CU_basic_set_mode(CU_BRM_VERBOSE);
        -:   67:
        1:   68:        CU_basic_run_tests();
        -:   69:
        1:   70:        CU_cleanup_registry();
        -:   71:
        1:   72:        return CU_get_error();
        -:   73:}
```

5 Pecas

5.1 Peca.h

Módulo contendo a declaração da struct peca, a qual é composta pelos seguintes membros:

- posicao_x, do tipo inteiro;
- posicao_y, do tipo inteiro;
- tamanho, do tipo inteiro;
- orientacao, do tipo inteiro;
- cor, do tipo inteiro;
- simbolo, do tipo ponteiro para caractere.

Segue a declaração da struct:

```
typedef struct peca{
int posicao_x, posicao_y,tamanho,orientacao,cor,tipo;
char* simbolo;
char formato[7][7];
}peca;
```

```
peca* nova_peca(Tela* tela);
```

Funcão que recebe a tela de jogo onde a peca deve ser impressa e retorna a peca que foi criada

```
int move_peca_x(Tela* tela,peca* a,int direcao);
```

Funcão que recebe a tela de jogo onde a peca deve ser impressa, a peca que deve ser impressa e a direcao para onde ela deve se mover e retorna um inteiro usado para testes

```
int move_peca_y(Tela* tela,peca* a);
```

Funcão que recebe a tela de jogo onde a peça deve ser impressa e a peça que deve ser impressa e retorna um inteiro usado para testes

```
void rotaciona_peca(Tela* tela,peca* a);
```

Funcão que recebe a tela de jogo onde a peça deve ser impressa e a peça que deve ser impressa

5.2 Peca.c

Arquivo do tipo .c, tem as seguintes funções implementadas:

nova_peca: Função que recebe uma tela, aloca espaço para uma estrutura do tipo peça, e faz as suas devidas inicializações, atribuindo a seu campo tamanho um valor aleatório entre três e cinco, escolhendo aleatoriamente, também um tipo(horizontal, vertical, Z, L, T, ou quadrado), e definindo uma cor aleatória para a peça(vermelho, azul ou verde). Por fim, a função faz mudanças na matriz_gui da tela para que a peça possa ser mostrada. Retorna a estutura da peça ao final.

move_peca_x: Recebe uma tela, uma peça, e um inteiro direção, que pode ser -1 ou 1, indicando movimento para a esquerda ou direita, respectivamente. Faz-se, então, uma checagem quanto a orientação da peça, e assegura-se a possibilidade quanto ao movimento da peça naquela direção, ou seja, se não há colisões entre a peça em questão e outras peças. A função tem diferentes retornos do tipo inteiro,para que cada situação seja tratada de maneira diferente pela função que a chamar.

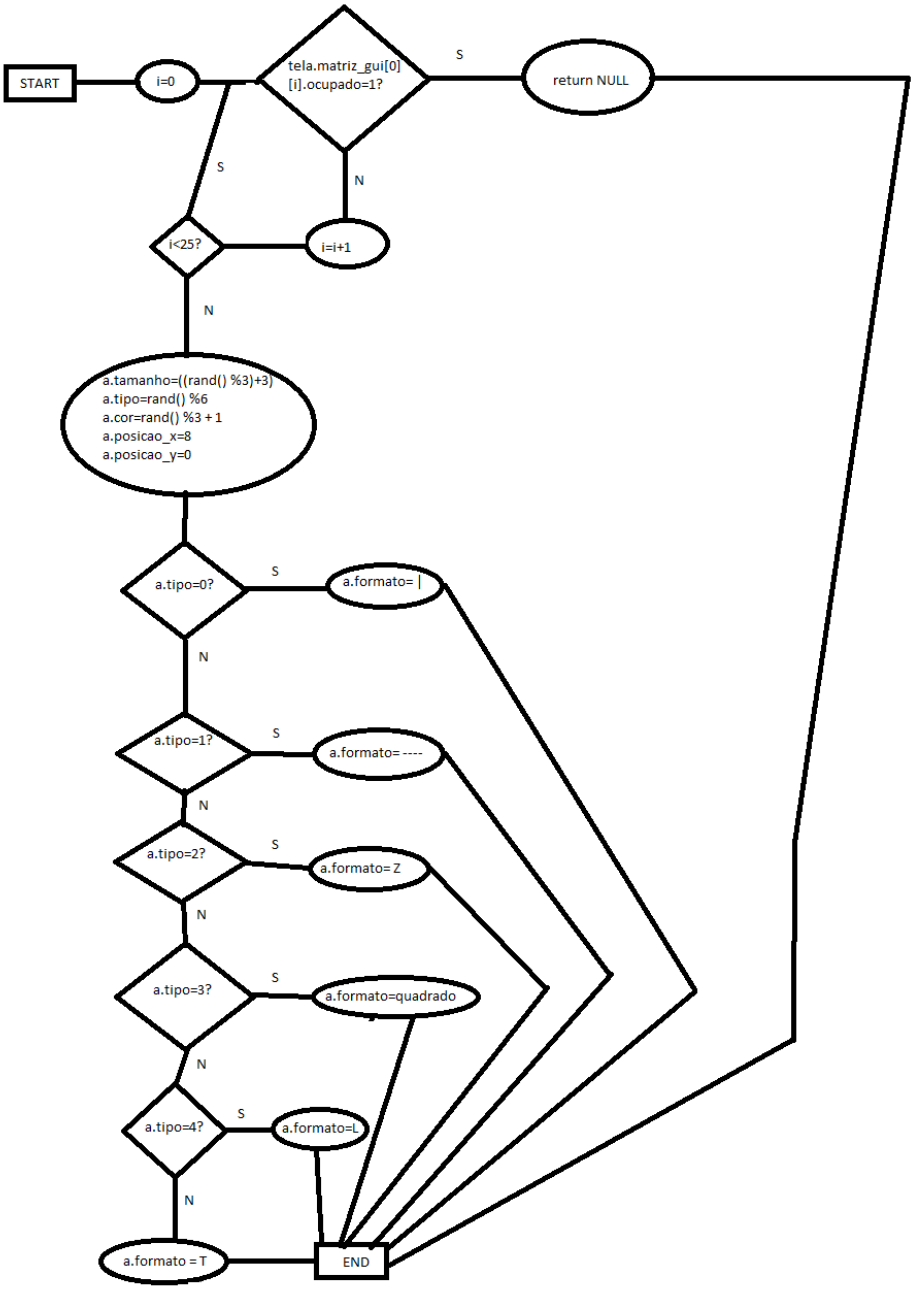
move_peca_y: Recebe uma tela e uma peça. Move a peça para baixo, checando se há colisões. Caso haja, retorna um valor. Caso contrário, move a peça para baixo, atualizando sua posição na matriz_gui.

rotaciona_peca: Recebe uma tela e uma peça. São copiados, então, os conteúdos da peça para outra peça que servirá para determinar se a rotaçáo da peça é válida. Caso seja, a peça original recebe os valores da peça alterada; se não, nada é alterado.

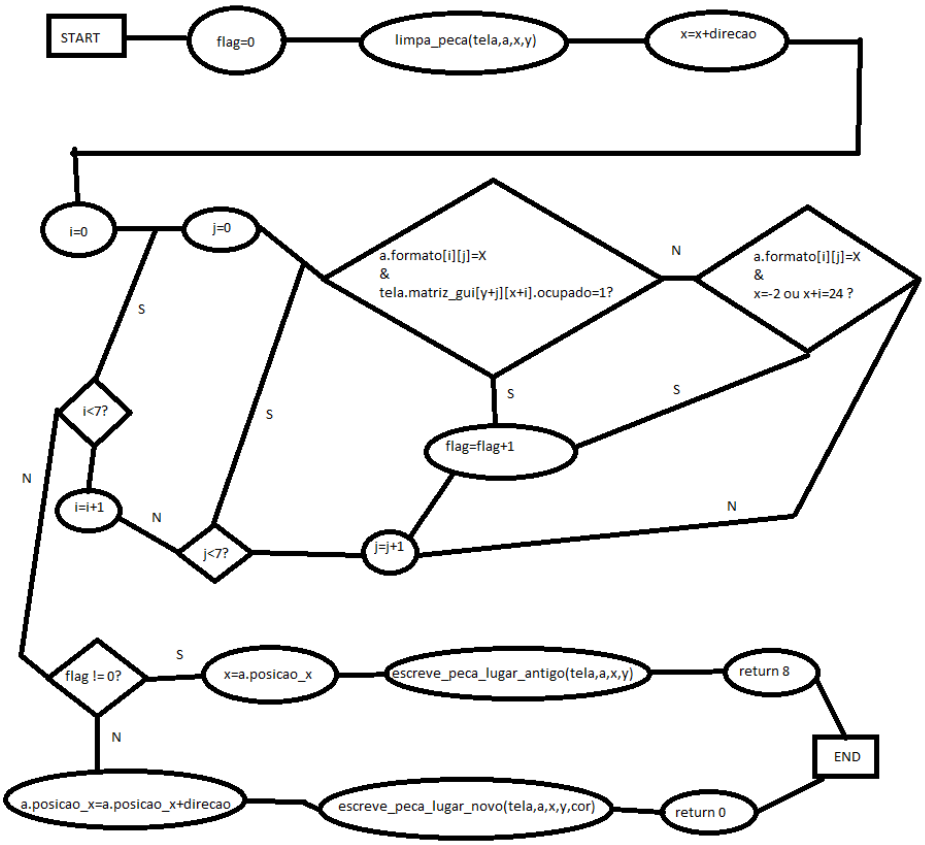
Também são utilizadas as funções escreve_peca_lugar_antigo, escreve_peca_lugar_novo e limpa_peca. Elas reduzem o tamanho total do módulo em termos de quantidade de linhas.

5.3 Fluxogramas das Funções

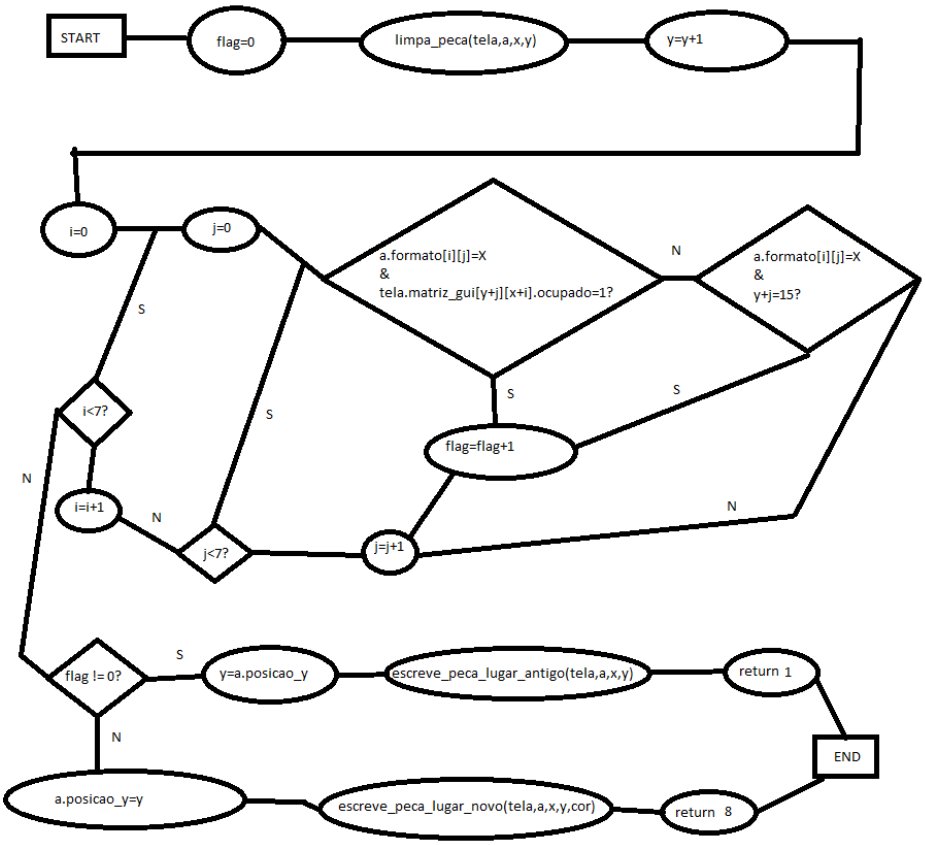
nova_peca:



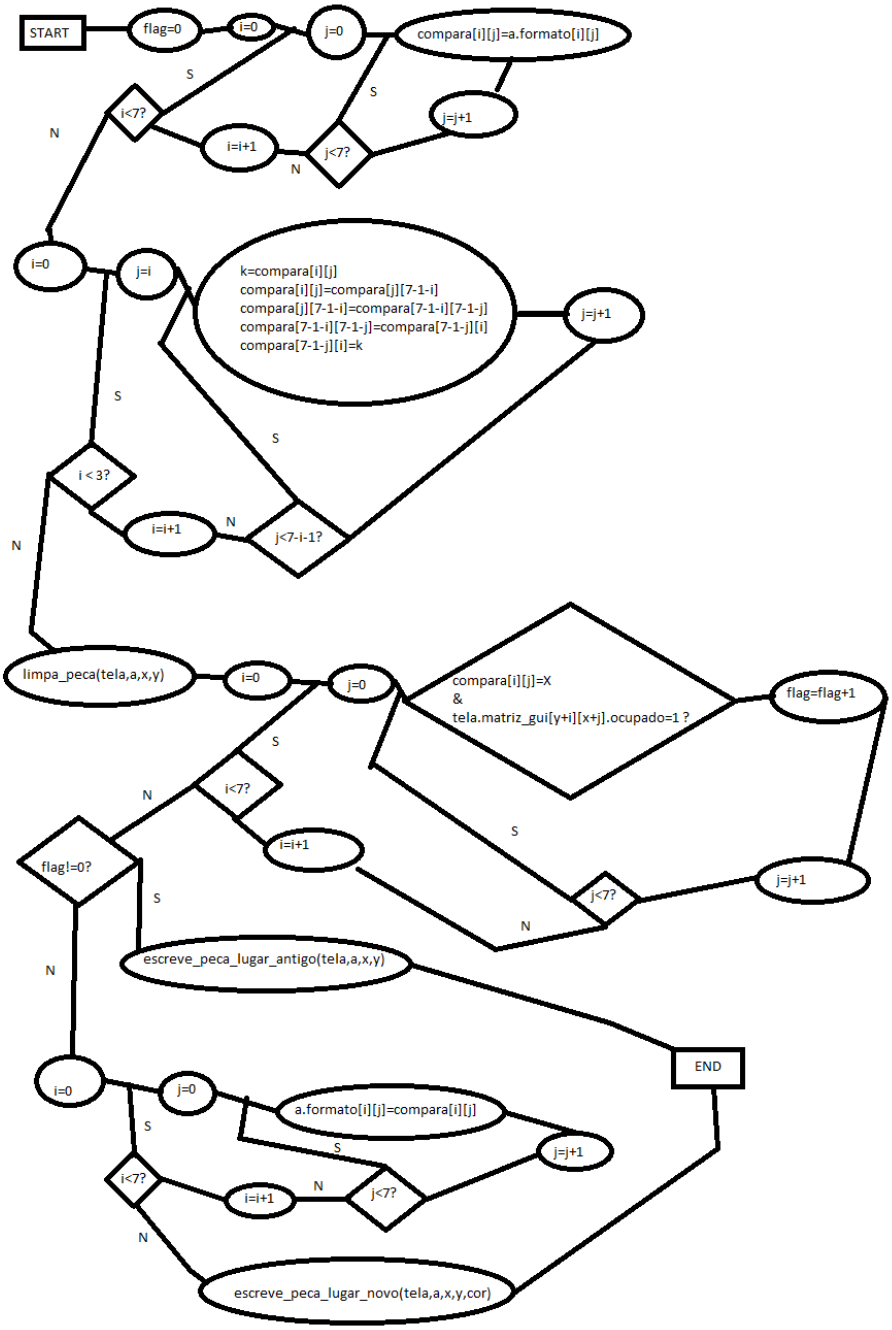
move_peca_x:


START
flag=0
limpa_peca(tela,a,x,y)
x=x+direcao
i=0
j=0
a.formato[i][j]=X
&
tela.matriz_gui[y+j][x+i].ocupado=1?
N
a.formato[i][j]=X
&
x=-2 ou x+i=24 ?
S
S
S
i<7?
flag=flag+1
N
i=i+1
N
j<7?
j=j+1
N
flag != 0?
S
x=a.posicao_x
escreve_peca_lugar_antigo(tela,a,x,y)
return 8
END
N
a.posicao_x=a.posicao_x+direcao
escreve_peca_lugar_novo(tela,a,x,y,cor)
return 0

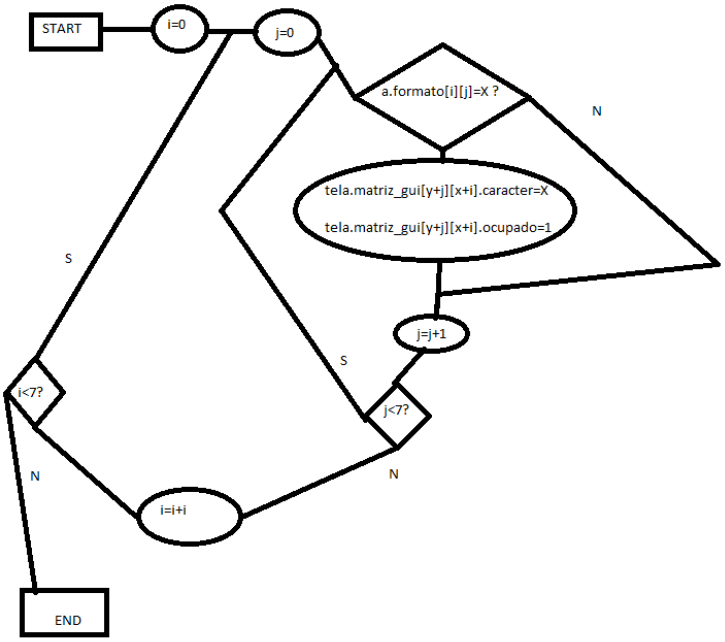
move_peca_y:


START
flag=0
limpa_peca(tela,a,x,y)
y=y+1
i=0
j=0
a.formato[i][j]=X
&
tela.matriz_gui[y+j][x+i].ocupado=1?
N
a.formato[i][j]=X
&
y+j=15?
S
S
S
i<7?
flag=flag+1
N
i=i+1
N
j<7?
j=j+1
N
flag != 0?
S
y=a.posicao_y
escreve_peca_lugar_antigo(tela,a,x,y)
return 1
END
N
a.posicao_y=y
escreve_peca_lugar_novo(tela,a,x,y,cor)
return 8

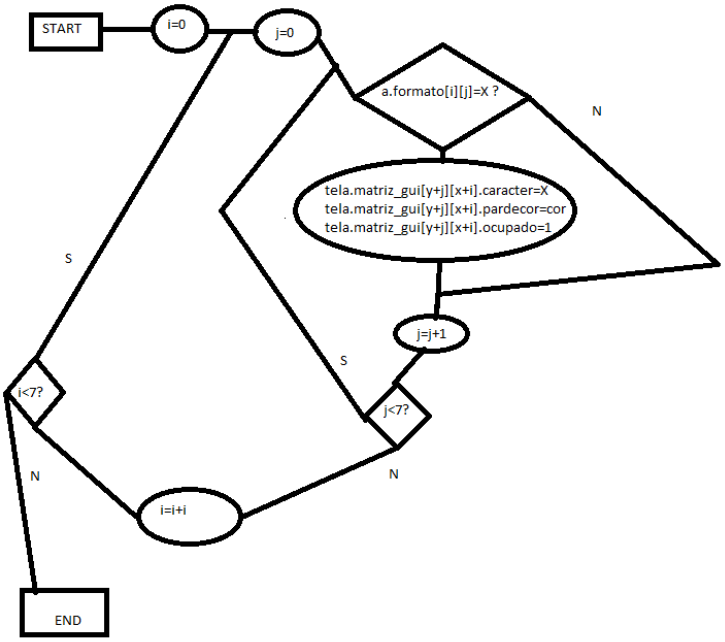
rotaciona_peca:



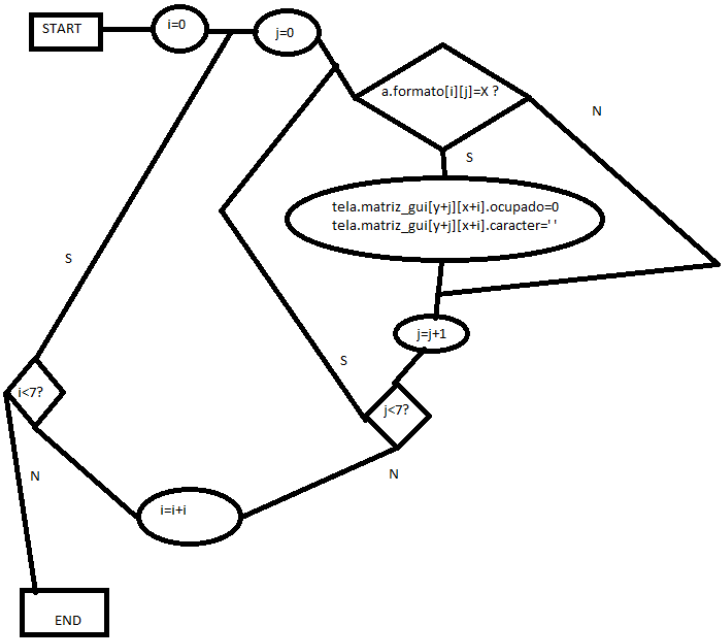
escreve_peca_lugar_antigo:



escreve_peca_lugar_novo:



limpa_peca:


START
i=0
j=0
a.formato[i][j]=X ?
N
S
tela.matriz_gui[y+j][x+i].ocupado=0
tela.matriz_gui[y+j][x+i].caracter=' '
S
j=j+1
S
i<7?
j<7?
N
N
i=i+i
END

5.4 Saída do Exame do GCov

```
    -:    0:Source:Pecas_Teste.c
-:    0:Graph:Pecas_Teste.gcno
-:    0:Data:Pecas_Teste.gcda
-:    0:Runs:1
-:    0:Programs:1
-:    1:#include "Pecas.c"
-:    2:#include "CUnit/CUnit.h"
-:    3:#include "CUnit/Basic.h"
-:    4:/*
-:    5:*        \author Luiz Felipe Carvalho Duarte
-:    6:*        \since 30/06/2016
-:    7:*        \version 1.0
-:    8:*/
-:    9:/*       Testamos primeiramente a criaÃ§Ã£o de novas peÃ
-:   10:*        Nos asseguramos de que a funÃ§Ã£o de criaÃ§Ã£o
-:   11:*
-:   12:*/
1:   13:void teste_nova_peca(){
-:   14:        Tela b;
1:   15:        peca* a=nova_peca(&b);
1:   16:        CU_ASSERT_PTR_NOT_NULL(a);
1:   17:        free(a);
1:   18:}
-:   19:/*
-:   20:*        Aqui nos asseguramos de que a funÃ§Ã£o de movim
-:   21:*
-:   22:*/
1:   23:void teste_move_peca_x_direita(){
-:   24:        int resultado;
-:   25:        Tela b;
1:   26:        peca* a=nova_peca(&b);
1:   27:        resultado=move_peca_x(&b,a,1);
1:   28:        free(a);
1:   29:        CU_ASSERT_EQUAL(resultado,0);
1:   30:}
1:   31:void teste_move_peca_x_esquerda(){
-:   32:        int resultado;
-:   33:        Tela b;
1:   34:        peca* a=nova_peca(&b);
1:   35:        resultado=move_peca_x(&b,a,-1);
1:   36:        free(a);
```

```
        1:   37:        CU_ASSERT_EQUAL(resultado,8);
        1:   38:}
        -:   39:/*
        -:   40:*       Como com o movimento horizontal, nos asseguramo
        -:   41:*       o valor que indica o funcionamento correto.
        -:   42:*/
        1:   43:void teste_move_peca_y(){
        -:   44:        int resultado;
        -:   45:        Tela b;
        1:   46:        peca* a=nova_peca(&b);
        1:   47:        resultado=move_peca_y(&b,a);
        1:   48:        free(a);
        1:   49:        CU_ASSERT_EQUAL(resultado,1);
        1:   50:}
        1:   51:void adicionar_suite(){
        -:   52:        CU_pSuite suite;
        1:   53:        suite=CU_add_suite("Testes_de_Pecas.c",NULL,NUL
        1:   54:        CU_ADD_TEST(suite, teste_nova_peca);
        1:   55:        CU_ADD_TEST(suite, teste_move_peca_x_direita);
        1:   56:        CU_ADD_TEST(suite, teste_move_peca_x_esquerda);
        1:   57:        CU_ADD_TEST(suite, teste_move_peca_y);
        1:   58:}
        1:   59:int main(void){
        -:   60:
        1:   61:        if (CUE_SUCCESS != CU_initialize_registry())
    #####:   62:                return CU_get_error();
        -:   63:
        1:   64:        adicionar_suite();
        -:   65:
        1:   66:        CU_basic_set_mode(CU_BRM_VERBOSE);
        1:   67:        CU_basic_run_tests();
        1:   68:        CU_cleanup_registry();
        -:   69:
        1:   70:        return CU_get_error();
        -:   71:}
```

6 Engine

6.1 Engine.h

O módulo Engine.h possui a interface de diversas funções que serão implementadas no Engine.c. São as seguinte:

```
void inicia_ncurses();
```

Uma função sem parâmetros e sem retorno.

```
void finaliza_ncurses();
```

Uma função sem parâmetros e sem retorno.

```
int deleta_linha(Tela* tela, int linha);
```

Esta função recebe um ponteiro para Tela, e um inteiro correspondente a uma linha.

```
int movimento(Tela* tela);
```

Função que recebe ponteiro de tela e retorna um inteiro.

6.2 Engine.c

Arquivo do tipo .c, tem as seguintes funções implementadas:

inicia_ncurses: Função que chama várias outras funções da ncurses, assim inicializando as funcionalidades da biblioteca.

finaliza_ncurses: Função que finaliza as funcionalidades da biblioteca ncurses, chamando a função endwin().

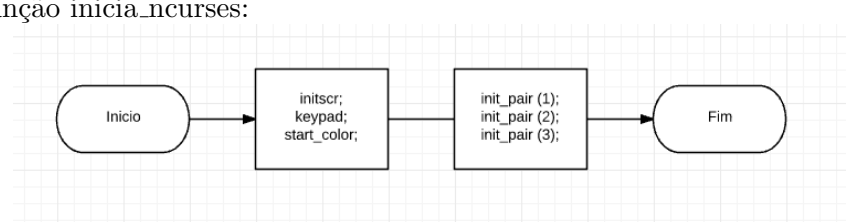
deleta_linha: Função que pega uma linha preenchida na matriz_gui, e a desocupa, posteriormente pegando todas as posições ocupadas acima e as reatribuindo uma posição abaixo.

movimento: A função que ao começar cria uma nova peça e a mostra na tela, então entra em um laço de repetição responsÃ¡vel por receber as entradas de movimento da peça até ela ser fixada na matriz_gui ao encerrar seu movimento. Dentro do laço é contado o tempo de espera por entrada do jogador e são tratado os casos de saída para as

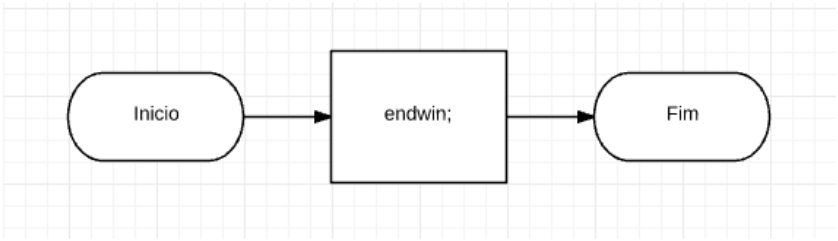
funções de movimento, e é chamada a função mostra_tela novamente. Ao final,é feito uma checagem se alguma linha está completamente ocupada. Caso esteja, é chamada a função deleta_linha.

6.3 Fluxogramas das Funções

Função inicia_ncurses:

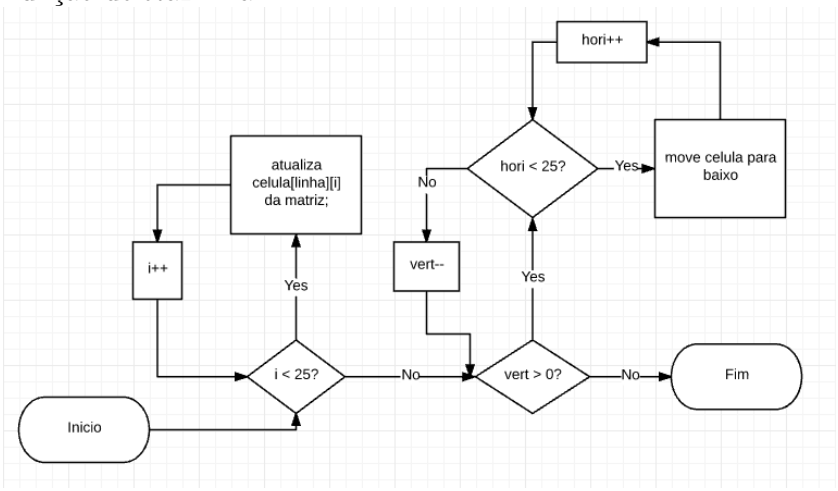


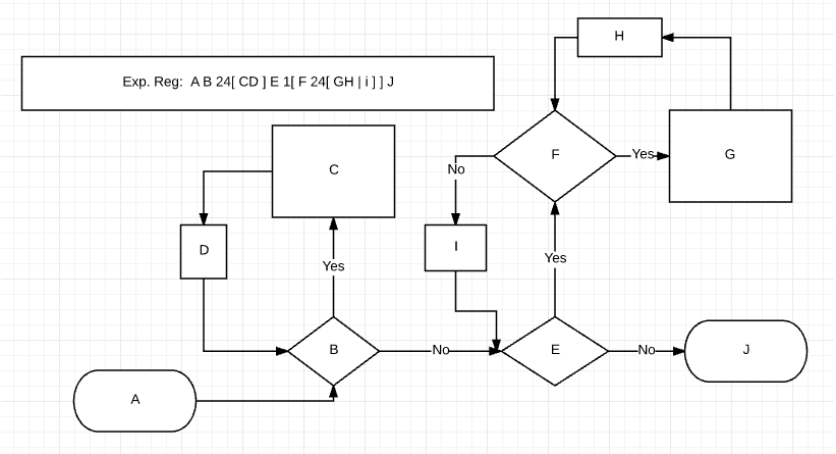
Função finaliza_ncurses:



As duas funções acima possuem expressão regular A, pois são duas funções simples e inteiramente sequenciais.

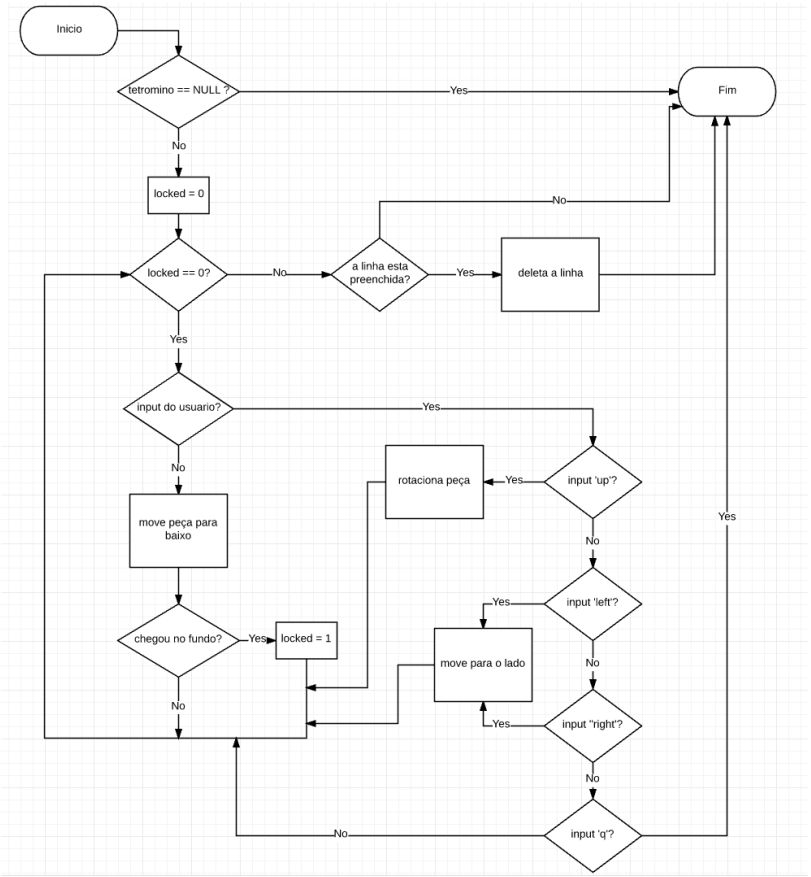
Função deleta_linha:

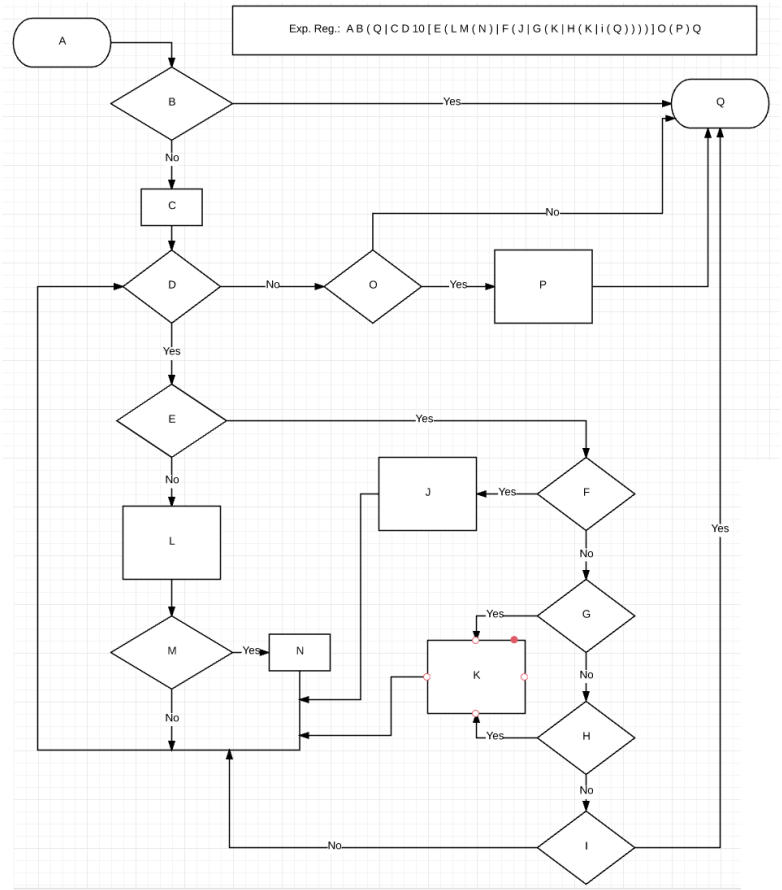




A função deleta_linha possui iterações com arrasto equivalente á largura da matriz da tela, ou seja, 24, por isso são necessárias várias repetições para atualizar os valores de todas as variáveis a ela pertencentes.

Função movimento:





A função movimento possui uma iteração D que possui arrasto 10, isso porque é o numero mínimo de movimentos da peça na tela para alcançar o fundo.

6.4 Saída do Exame do GCov

```
-:    0:Source:Engine_Teste.c
-:    0:Graph:Engine_Teste.gcno
-:    0:Data:Engine_Teste.gcda
-:    0:Runs:1
-:    0:Programs:1
-:    1:#include <time.h>
-:    2:#include <stdio.h>
-:    3:#include <stdlib.h>
-:    4:#include <string.h>
-:    5:
```

```
  -:    6:#include "CUnit/CUnit.h"
  -:    7:#include "CUnit/Basic.h"
  -:    8:#include "Engine.c"
  -:    9:
  -:   10:
  -:   11:void adicionar_suite(void);
  -:   12:void inicia_ncurses();
  -:   13:int finaliza_ncurses();
  -:   14:void deleta_linha(Tela* tela, int linha);
  -:   15:int movimento(Tela* tela, int* pontuacao);
  -:   16:
  -:   17:
  1:   18:void teste_inicia_ncurses(){
  -:   19:
  1:   20:        inicia_ncurses();
  1:   21:        CU_ASSERT(COLOR_PAIR(1));
  1:   22:        CU_ASSERT(COLOR_PAIR(2));
  1:   23:        CU_ASSERT(COLOR_PAIR(3));
  1:   24:}
  -:   25:
  1:   26:void teste_finaliza_ncurses(){
  1:   27:        CU_ASSERT_TRUE( finaliza_ncurses() == OK );
  1:   28:}
  -:   29:
  -:   30:
  1:   31:void teste_deleta_linha(){
  -:   32:        int i, j;
  1:   33:        Tela* tela = cria_tela();
 25:   34:        for(i=1; i<25; i++)
216:   35:                for(j=7; j<15; j++)
192:   36:                        tela->matriz_gui[i][j].ocupado
  -:   37:
  1:   38:        deleta_linha(tela, 10);
  -:   39:
  1:   40:        CU_ASSERT_TRUE( (tela->matriz_gui[13][6].ocupad
 16:   41:  for(i = 0; i<15; i++){
390:   42:                for(j = 0; j<25; j++){
375:   43:        tela->matriz_gui[i][j].caracter = 'X';
375:   44:        tela->matriz_gui[i][j].ocupado = 1;
  -:   45:      }
  -:   46:  }
 16:   47:  for(i = 0; i<15; i++)
 15:   48:    deleta_linha(tela, i);
```

```
 -:   49:
 1:   50:  CU_ASSERT_TRUE(tela->matriz_gui[8][16].caracter == 32
 1:   51:}
 -:   52:
 1:   53:void teste_movimento(){
 1:   54:        int i, j, end=0;
 1:   55:        int pont = 0;
 1:   56:        Tela* tela = cria_tela();
 -:   57:
 1:   58:        mostra_tela(tela);
 1:   59:        movimento(tela, &pont);
 -:   60:
 1:   61:        CU_ASSERT_TRUE(pont == 0);
 -:   62:
24:   63:        for(i = 1; i<24; i++){
92:   64:                for(j = 12; j<15; j++){
69:   65:                        tela->matriz_gui[j][i].caracter
69:   66:                        tela->matriz_gui[j][i].ocupado
 -:   67:                }
 -:   68:        }
 1:   69:        mostra_tela(tela);
 1:   70:        refresh();
 -:   71:
 1:   72:        movimento(tela, &pont);
 -:   73:
 1:   74:        CU_ASSERT_TRUE(pont != 0);
 1:   75:}
 -:   76:
 -:   77:
 1:   78:void adicionar_suite(){
 -:   79:        CU_pSuite suite;
 1:   80:        suite = CU_add_suite("lery_tetris",NULL,NULL);
 -:   81:
 1:   82:        CU_ADD_TEST(suite, teste_inicia_ncurses);
 1:   83:        CU_ADD_TEST(suite, teste_deleta_linha);
 1:   84:        CU_ADD_TEST(suite, teste_movimento);
 1:   85:        CU_ADD_TEST(suite, teste_finaliza_ncurses);
 -:   86:
 1:   87:}
 -:   88:
 1:   89:int main(void){
 -:   90:
 1:   91:        if (CUE_SUCCESS != CU_initialize_registry())
```

```
#####:   92:                return CU_get_error();
    -:   93:
    1:   94:        adicionar_suite();
    -:   95:
    1:   96:        CU_basic_set_mode(CU_BRM_VERBOSE);
    1:   97:        CU_basic_run_tests();
    1:   98:        CU_cleanup_registry();
    -:   99:
    1:  100:        return CU_get_error();
    -:  101:}
```

7 Placar

7.1 Placar.h

Módulo contendo a declaração da struct Jogador, o qual é composto pelos seguintes membros:

- vetor de caracteres do tipo char, nome;
- valor do tipo inteiro, pontos.

Segue declaração da struct:

```
struct tipojogador{
  char nome[6];
  int pontos;
};
typedef struct tipojogador Jogador;
```

```
void cria_placar(void);
```

Função que não recebe argumentos e não retorna nada.

```
void atualiza_placar(int pontuacao);
```

Função que recebe a pontuacao a ser escrita no arquivo de placares e não retorna nada.

```
void mostra_placar(void);
```

Função que não recebe argumentos e não retorna nada.

7.2 Placar.c

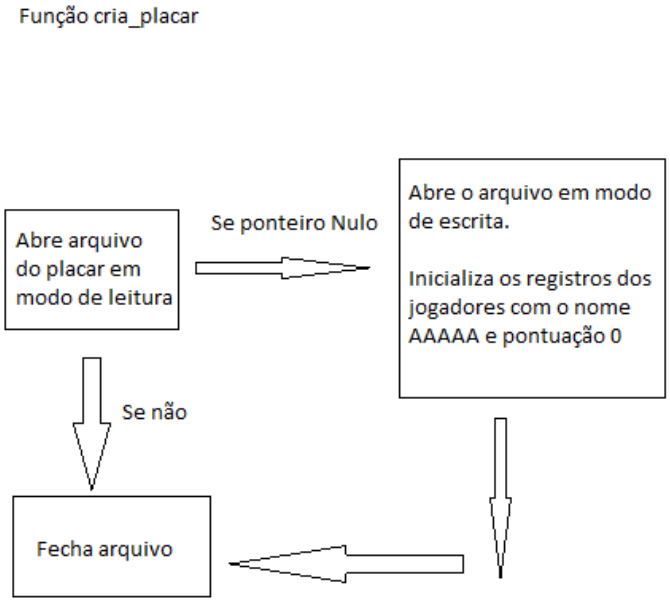
Arquivo do tipo .c, implementa as seguintes funções:

cria_placar: Esta função abre um arquivo no modo leitura, checando se ele existe. Caso não exista, abrirá o arquivo no modo escrita+, inicializando o .txt que contém as informações do placar do jogo.

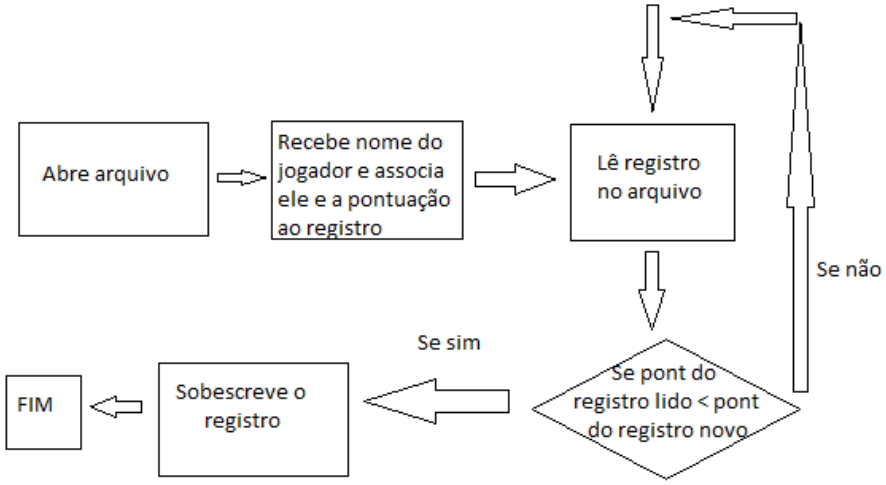
atualiza_placar: A função pede entrada ao usuário de um nome de até 5 letras, e logo após checa se a pontuação obtida no jogo supera alguma das gravadas em disco. No caso de superar, o novo registro Jogador é escrito na posição correspondente, mantendo um arquivo de registros ordenados pela pontuação. Caso contrário, não será escrito nada no arquivo .txt.

mostra_placar: A função mostra_placar abre o arquivo placar.txt em modo leitura, lê cinco linhas escritas no arquivo e as imprime na tela, sendo estas linhas o nome e a pontuação de cada jogador, ordenado por pontuação.

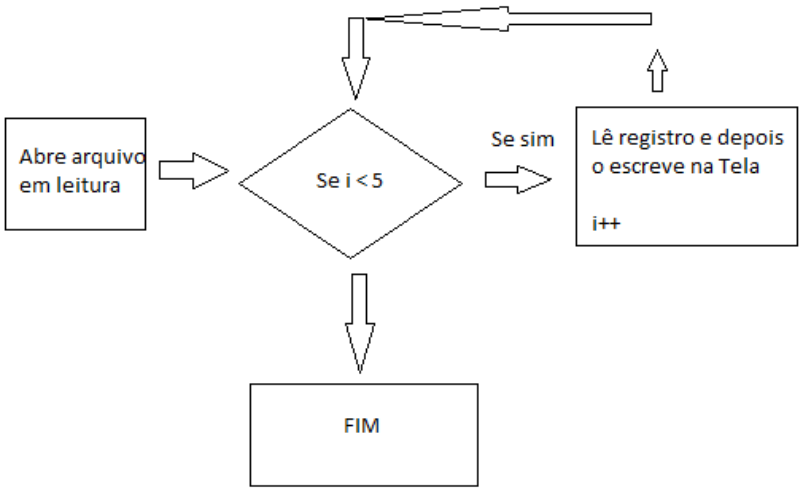
7.3 Fluxogramas das Funções



Função atualiza_placar



Função mostra_placar



7.4 Saída do Exame do GCov

```
-:    0:Source:Placar_Teste.c
-:    0:Graph:Placar_Teste.gcno
-:    0:Data:Placar_Teste.gcda
-:    0:Runs:1
```

```
-:    0:Programs:1
-:    1:
-:    2:#include "CUnit/CUnit.h"
-:    3:#include "CUnit/Basic.h"
-:    4:
-:    5:void adicionar_suite(void);
-:    6:
-:    7:void teste_cria_placar(void);
-:    8:void teste_atualiza_placar(void);
-:    9:void teste_mostra_placar(void);
-:   10:
1:   11:void teste_cria_placar(void){
-:   12:        int resultado;
1:   13:        resultado = cria_placar();
1:   14:        CU_ASSERT_TRUE(!resultado);
1:   15:}
-:   16:
1:   17:void teste_atualiza_placar_100(void){
-:   18:        int resultado;
1:   19:        resultado = atualiza_placar(100);
1:   20:        CU_ASSERT_TRUE(!resultado);
1:   21:}
-:   22:
1:   23:void teste_atualiza_placar_999999(void){
-:   24:        int resultado;
1:   25:        resultado = atualiza_placar(999999);
1:   26:        CU_ASSERT_FALSE(!resultado);
1:   27:}
-:   28:
-:   29:
1:   30:void teste_mostra_placar(void){
-:   31:        int resultado;
1:   32:        resultado = mostra_placar();
1:   33:        CU_ASSERT_TRUE(!resultado);
1:   34:}
-:   35:
1:   36:void adicionar_suite(void){
-:   37:        CU_pSuite suite;
-:   38:
1:   39:        suite = CU_add_suite("Testes",NULL,NULL);
-:   40:
1:   41:        CU_ADD_TEST(suite, teste_cria_placar);
1:   42:        CU_ADD_TEST(suite, teste_mostra_placar);
```

```
        1:   43:        CU_ADD_TEST(suite, teste_atualiza_placar_100);
        1:   44:        CU_ADD_TEST(suite, teste_atualiza_placar_999999
        -:   45:
        1:   46:}
        -:   47:
        1:   48:int main(){
        1:   49:        if (CUE_SUCCESS != CU_initialize_registry())
    #####:   50:                return CU_get_error();
        -:   51:
        1:   52:        adicionar_suite();
        -:   53:
        1:   54:        CU_basic_set_mode(CU_BRM_VERBOSE);
        -:   55:
        1:   56:        CU_basic_run_tests();
        -:   57:
        1:   58:        CU_cleanup_registry();
        -:   59:
        1:   60:        return CU_get_error();
        -:   61:}
```

8 A função Main

```
int main(){

        srand(time(NULL));

        int end = 0;
        int pontuacao = 0;

        inicia_ncurses();

        Tela* t = cria_tela();

        mostra_tela_inicial();
        getch();

        while(end == 0){

                mostra_tela(t);
                end = movimento(t, &pontuacao);
        }

        mostra_tela_final(pontuacao*100);
        getch();

        mostra_tela_placar(pontuacao*100);
        getch();

        free(t);
        finaliza_ncurses();

        return 0;
}
```

Nesta função, é inicializada a seed para as funções que usam aleatoriedade, é criada uma estrutura Tela, e então se mostra a tela de boas vindas, para posteriormente utilizar a estrutura criada dentro de um laço que termina quando o jogador perder o jogo. O laço irá constantemente se chama mostra_tela e movimento, sendo que esta retorna se o jogo está perdido ou não. Por fim, mostra-se a tela final, libera-se a estrutura Tela, e finaliza-se o ncurses.

9 Divisão do Trabalho

- Tela.h, Tela.c, Placar.h e Placar.c: Eduardo de Azevêdo dos Santos
- Pecas.h e Pecas.c: Luiz Felipe Carvalho Duarte
- Engine.h e Engine.c: Yero Távora; Rafael Cascardo Campos

No geral, todos ajudaram aos outros com dicas e ajustes. O makefile foi feito e atualizado por todos, e a documentação feita por Eduardo de Azevêdo dos Santos e Rafael Cascardo Campos em LaTeX.