Tetris - PS Edition

Gerado por Doxygen 1.8.11

Terça, 7 de Junho de 2016 23:58:12

Sumário

1 Índice das Estruturas de Dados

1.1 Estruturas de Dados

Aqui estão as estruturas de dados, uniões e suas respectivas descrições:

2 Índice dos Arquivos

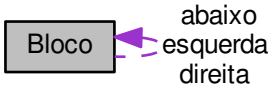
2.1 Lista de Arquivos

Esta é a lista de todos os arquivos documentados e suas respectivas descrições:

3 Estruturas

3.1 Referência da Estrutura Bloco

Diagrama de colaboração para Bloco:



Campos de Dados

- char bolinha
- unsigned short int cor
- int pos_x
- int pos_y
- unsigned short int move
- struct Bloco ∗ esquerda
- struct Bloco ∗ direita
- struct Bloco ∗ abaixo

3.1.1 Descrição Detalhada

Definição na linha 5 do arquivo bloco.h.

3.1.2 Campos

3.1.2.1 struct Bloco∗ Bloco::abaixo

Ponteiro para vizinho abaixo.

Definição na linha 13 do arquivo bloco.h.

3.1.2.2 char Bloco::bolinha

Caractere atual da peça.

Definição na linha 6 do arquivo bloco.h.

3.1.2.3 unsigned short int Bloco::cor

Cor da peça.

Definição na linha 7 do arquivo bloco.h.

3.1.2.4 struct Bloco* Bloco::direita

Ponteiro para vizinho à direita.

Definição na linha 12 do arquivo bloco.h.

3.1.2.5 struct Bloco* Bloco::esquerda

Ponteiro para vizinho à esquerda.

Definição na linha 11 do arquivo bloco.h.

3.1.2.6 unsigned short int Bloco::move

Valor booleano que indica se o bloco está em movimento ou não.

Definição na linha 10 do arquivo bloco.h.

3.1.2.7 int Bloco::pos_x

Coordenada cartesiana horizontal do bloco.

Definição na linha 8 do arquivo bloco.h.

3.1.2.8 int Bloco::pos_y

Coordenada cartesiana vertical do bloco.

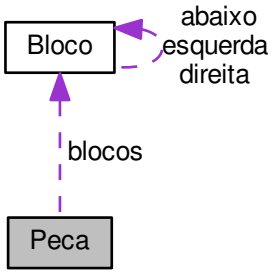
Definição na linha 9 do arquivo bloco.h.

A documentação para esta estrutura foi gerada a partir do seguinte arquivo:

- bloco.h

3.2 Referência da Estrutura Peca

Diagrama de colaboração para Peca:



Campos de Dados

- int tamanho
- unsigned short int cor_peca
- unsigned short int move_peca
- bloco ∗ blocos [ ]

3.2.1 Descrição Detalhada

Definição na linha 8 do arquivo pecas.h.

3.2.2 Campos

3.2.2.1 bloco∗ Peca::blocos[ ]

Referência para blocos na tela.

Definição na linha 12 do arquivo pecas.h.

3.2.2.2 unsigned short int Peca::cor_peca

Cor da peça.

Definição na linha 10 do arquivo pecas.h.

3.2.2.3 unsigned short int Peca::move_peca

Booleano que checa se a peça está em movimento ou não.

Definição na linha 11 do arquivo pecas.h.

3.2.2.4 int Peca::tamanho

Tamanho da peça.

Definição na linha 9 do arquivo pecas.h.

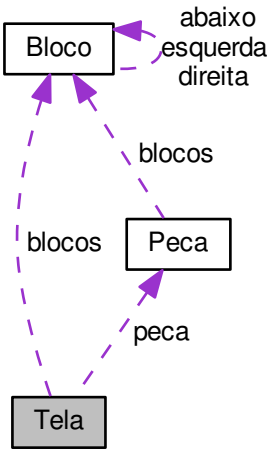
A documentação para esta estrutura foi gerada a partir do seguinte arquivo:

- pecas.h

3.3 Referência da Estrutura Tela

```
#include <tela.h>
```

Diagrama de colaboração para Tela:



Campos de Dados

- int estado
- int pontos
- int tempo_m
- int tempo_s
- int comprimento
- int largura
- WINDOW * janela
- struct Peca * peca
- bloco blocos [ ]

3.3.1 Descrição Detalhada

/struct Define a tela do jogo.

Definição na linha 13 do arquivo tela.h.

3.3.2 Campos

3.3.2.1 bloco Tela::blocos[ ]

Matriz dos blocos na tela.

Definição na linha 22 do arquivo tela.h.

3.3.2.2 int Tela::comprimento

Comprimento da tela.

Definição na linha 18 do arquivo tela.h.

3.3.2.3 int Tela::estado

Estado atual do jogo.

Definição na linha 14 do arquivo tela.h.

3.3.2.4 WINDOW∗ Tela::janela

Ponteiro para a janela do jogo.

Definição na linha 20 do arquivo tela.h.

3.3.2.5 int Tela::largura

Largura da tela.

Definição na linha 19 do arquivo tela.h.

3.3.2.6 struct Peca∗ Tela::peca

Ponteiro para a peça em movimento.

Definição na linha 21 do arquivo tela.h.

3.3.2.7 int Tela::pontos

Pontuação do jogador.

Definição na linha 15 do arquivo tela.h.

3.3.2.8 int Tela::tempo_m

Tempo de execução em minutos.

Definição na linha 16 do arquivo tela.h.

3.3.2.9 int Tela::tempo_s

Tempo de execução em segundos.

Definição na linha 17 do arquivo tela.h.

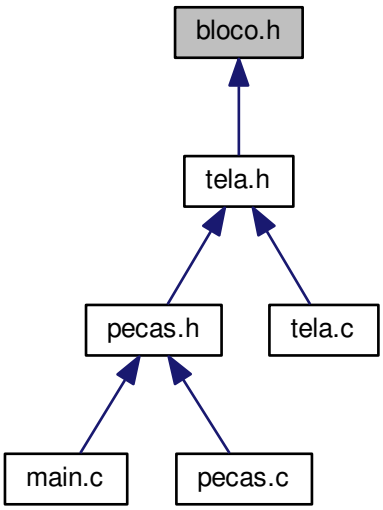
A documentação para esta estrutura foi gerada a partir do seguinte arquivo:

- tela.h

4 Arquivos

4.1 Referência do Arquivo bloco.h

Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com este arquivo:



Estruturas de Dados

- struct Bloco

Definições e Macros

- #define **COMPRIMENTO** 15
- #define **LARGURA** 25

Definições de Tipos

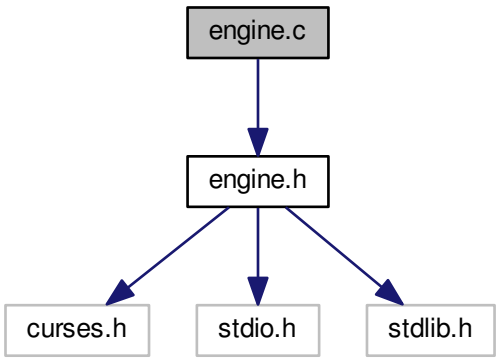
- typedef struct Bloco **bloco**

4.2 bloco.h

```
00001
00005 typedef struct Bloco{
00006   char bolinha;
00007   unsigned short int cor;
00008   int pos_x;
00009   int pos_y;
00010   unsigned short int move;
00011   struct Bloco* esquerda;
00012   struct Bloco* direita;
00013   struct Bloco* abaixo;
00014 }bloco;
00015
00017 #define COMPRIMENTO 15
00018
00019 #define LARGURA 25
00020
```

4.3 Referência do Arquivo engine.c

`#include "engine.h"`
Gráfico de dependência de inclusões para engine.c:



Funções

- void inicia_ncurses ()
- void finaliza_ncurses ()
- int pega_input (int input)

4.3.1 Funções

4.3.1.1 void finaliza_ncurses ( )

Finaliza o modo ncurses.

Definição na linha 15 do arquivo engine.c.

4.3.1.2 void inicia_ncurses ( )

Inicializa o modo ncurses e determina as funcionalidades dele que serão usadas.

Definição na linha 6 do arquivo engine.c.

4.3.1.3 int pega_input (int *input*)

Determina como interpretar a entrada do teclado.

Parâmetros

input	Entrada.

Retorna

Saída convertida.

Definição na linha 23 do arquivo engine.c.

4.4 engine.c

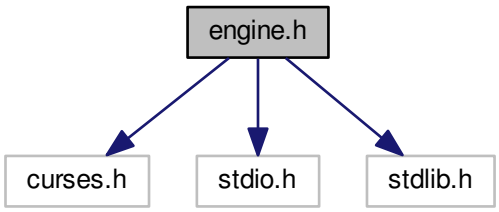
```
00001
00003 #include "engine.h"
00004
00006 void inicia_ncurses(){
00007     initscr();
00008     start_color();
00009 //  raw();
00010     keypad(stdscr,TRUE);
00011 //  noecho();
00012 }
00013
00015 void finaliza_ncurses(){
00016     endwin();
00017 }
00018
00023 int pega_input(int input){
00024     switch(input){
00025     case KEY_DOWN:
00026     case 's':
00027         return 2;
00028         break;
00029     case KEY_RIGHT:
00030     case 'd':
00031         return 3;
00032         break;
00033     case KEY_LEFT:
00034     case 'a':
00035         return 4;
00036         break;
00037     case KEY_F(4):
00038         return 0;
00039         break;
```

```
00040     default:
00041         return 1;
00042         break;
00043     }
00044
00045 }
```

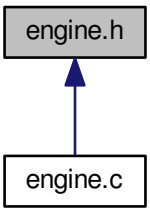
4.5 Referência do Arquivo engine.h

```
#include <curses.h>
#include <stdio.h>
#include <stdlib.h>
```

Gráfico de dependência de inclusões para engine.h:



Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com este arquivo:



Funções

- void inicia_ncurses ()
- void finaliza_ncurses ()
- int pega_input (int input)

4.5.1 Funções

4.5.1.1 void finaliza_ncurses ( )

Finaliza o modo ncurses.

Definição na linha 15 do arquivo engine.c.

4.5.1.2 void inicia_ncurses ( )

Inicializa o modo ncurses e determina as funcionalidades dele que serão usadas.

Definição na linha 6 do arquivo engine.c.

4.5.1.3 int pega_input (int *input*)

Determina como interpretar a entrada do teclado.

Parâmetros

input	Entrada.

Retorna

Saída convertida.

Definição na linha 23 do arquivo engine.c.

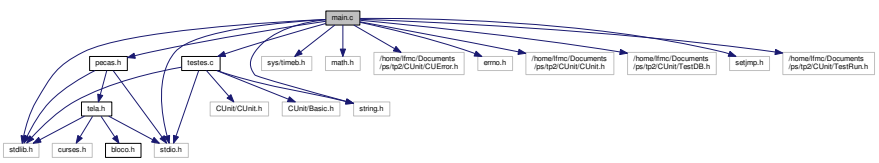
4.6 engine.h

```
00001
00003 #include <curses.h>
00004 #include <stdio.h>
00005 #include <stdlib.h>
00006
00007 void inicia_ncurses();
00008 void finaliza_ncurses();
00009 int pega_input(int input);
```

4.7 Referência do Arquivo main.c

```
#include <stdlib.h>
#include <stdio.h>
#include "pecas.h"
#include <sys/timeb.h>
#include "testes.c"
```

Gráfico de dependência de inclusões para main.c:



Funções

- int **main** ()

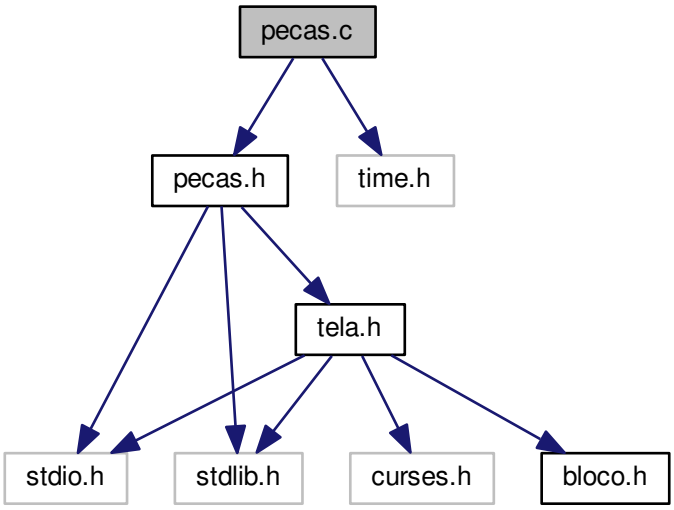
4.8 main.c

```
00001
00003 #include<stdlib.h>
00004 #include<stdio.h>
00005 #include"pecas.h"
00006 #include<sys/timeb.h>
00007 #include "testes.c"
00008
00009 int main(){
00010     if (CUE_SUCCESS != CU_initialize_registry())
00011         return CU_get_error();
00012
00013     adiciona_testes();
00014
00015         CU_basic_set_mode(CU_BRM_VERBOSE);
00016     /*Roda os testes e mostra na tela os resultados*/
00017     CU_basic_run_tests();
00018     /*Limpa o registro*/
00019     CU_cleanup_registry();
00020
00021     inicia_ncurses();
00022     Tela* tela = cria_tela();
00023     mostra_tela(tela);
00024     struct timeb inicio, atual;
00025     int pontos = 0;
00026     int get = getch();
00027     if(pega_input(get)){
00028         tela->estado = JOGO;
00029     }
00030     ftime(&inicio);
00031     nova_peca(tela);
00032     mostra_tela(tela);
00033
00034     while(pega_input(get)){
00035         ftime(&atual);
00036         mostra_tempo((atual.time - inicio.time)/60,(atual.time - inicio.time)%60);
00037         mostra_pontos(pontos);
00038         timeout(1000);
00039         get=getch();
00040
00041         if(pega_input(get) == 2){
00042             move_peca_y(tela->peca,1);
00043             mostra_tela(tela);
00044         }
00045         if(pega_input(get) == 3){
00046             move_peca_x(tela->peca,1);
00047             mostra_tela(tela);
00048         }
00049         if(pega_input(get) == 4){
00050             move_peca_x(tela->peca,-1);
00051             mostra_tela(tela);
00052         }
00053         if(!tela->peca->move_peca){
00054             pontos += verifica_linha(tela);
00055             libera_peca(tela->peca);
00056             nova_peca(tela);
00057             mostra_tela(tela);
00058         }
00059         if(checa_fim(tela)){
00060             ftime(&atual);
00061             tela->estado = FINAL;
00062             tela->pontos = pontos;
00063             tela->tempo_m = (atual.time - inicio.time)/60;
00064             tela->tempo_s = (atual.time - inicio.time)%60;
00065             mostra_tela(tela);
00066             timeout(-1);
00067             getch();
00068             get = KEY_F(4);
00069         }
00070     }
00071
00072
00073     destroi_tela(tela);
00074     finaliza_ncurses();
00075
00076     return CU_get_error();
00077     return 0;
00078
00079 }
```

4.9 Referência do Arquivo pecas.c

```
#include "pecas.h"
#include <time.h>
```

Gráfico de dependência de inclusões para pecas.c:



Funções

- void nova_peca (Tela *tela)
- void move_peca_x (peca *p, int x)
- void move_peca_y (peca *p, int y)
- void libera_peca (peca *p)

Variáveis

- unsigned short int cor_nova_peca = 4

4.9.1 Funções

4.9.1.1 void libera_peca (peca * p)

Libera a memória alocada para a peça.

Parâmetros

p	Ponteiro para a peça a ser liberada.

Definição na linha 166 do arquivo pecas.c.

4.9.1.2 void move_peca_x (peca * p, int x)

Movimenta a peça no eixo x, ou seja no sentido horizontal.

Parâmetros

p	A peça a ser movimentada.
x	Indica a direção do movimento. Se positivo, para a direita. Se negativo, para a esquerda.

Definição na linha 50 do arquivo pecas.c.

4.9.1.3 void move_peca_y (peca * p, int y)

Movimenta a peça no eixo y, ou seja, na direção vertical.

Parâmetros

p	Peça a ser movida.
y	Sempre deve ser positivo, pois a peça só pode se movimentar para baixo.

Definição na linha 117 do arquivo pecas.c.

4.9.1.4 void nova_peca (Tela * tela)

Gera nova peça do jogo. Orientação e tamanho são dados de forma pseudoaleatória. A cor é dada de forma cíclica.

Parâmetros

tela	Ponteiro para tela de jogo.

<Indica a orientação da peça. Se 1, a orientação é vertical. Se 0, a orientação é horizontal

<Indica o tamanho da peça

Definição na linha 14 do arquivo pecas.c.

4.9.2 Variáveis

4.9.2.1 unsigned short int cor_nova_peca = 4

Par de cores das peças variam entre 4 e 7.

Definição na linha 7 do arquivo pecas.c.

4.10 pecas.c

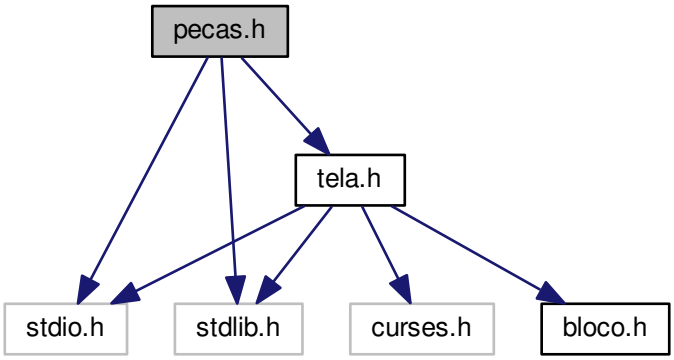
```
00001
00003 #include "pecas.h"
00004 #include <time.h>
00005
00007 unsigned short int cor_nova_peca = 4;
00008
00014 void nova_peca(Tela* tela){
00015     srand(time(NULL));
00016     int i;
00017     int orientacao = rand()%2;
00018     int tamanho = rand()%3 + 3;
00020     peca *p = malloc(sizeof(peca) + tamanho*sizeof(bloco*));
00021
00022     p->tamanho = tamanho;
00023     p->cor_peca = cor_nova_peca;
00024     p->move_peca = 1;
00025
00026     for(i = 0; i < p->tamanho; i++){
00027         if(orientacao){
00028             p->blocos[i] = &(tela->blocos[i*tela->largura + tela->
      largura/2]);
00029             p->blocos[i]->cor = p->cor_peca;
00030             p->blocos[i]->bolinha = 'o';
00031             p->blocos[i]->move = 1;
00032         }else{
00033             p->blocos[i] = &(tela->blocos[i - (p->tamanho)/2 + tela->
      largura/2]);
00034             p->blocos[i]->cor = p->cor_peca;
00035             p->blocos[i]->bolinha = 'o';
00036             p->blocos[i]->move = 1;
00037         }
00038     }
00039     if(cor_nova_peca == 7){
00040         cor_nova_peca = 4;
00041     }
00042     cor_nova_peca++;
00043     tela->peca = p;
00044 }
00045
00050 void move_peca_x(peca* p, int x){
00051
00052     unsigned short int colisao = 0;
00053     int i;
00054
00055     switch(x){
00056         case 1:
00057             for (i = 0; i < p->tamanho; i++){
00058                 if(p->blocos[i]->pos_x != 24){
00059                     if(((p->blocos[i]->direita->bolinha == 'o')&&(p->
      blocos[i]->direita->move == 0))){
00060                         colisao = 1;
00061                         break;
00062                     }
00063                 }
00064                 else{
00065                     colisao = 1;
00066                     break;
00067                 }
00068             }
00069             if(colisao != 1){
00070                 for (i = p->tamanho - 1; i >= 0; i--){
00071                     p->blocos[i]->cor = 2;
00072                     p->blocos[i]->bolinha = ' ';
00073                     p->blocos[i]->move = 0;
00074
00075                     p->blocos[i] = p->blocos[i]->direita;
00076                     p->blocos[i]->cor = p->cor_peca;
00077                     p->blocos[i]->bolinha = 'o';
00078                     p->blocos[i]->move = 1;
00079                 }
00080             }
00081             break;
00082         case -1:
00083             for (i = 0; i < p->tamanho; i++){
00084                 if((p->blocos[i]->pos_x != 0)){
00085                     if(((p->blocos[i]->esquerda->bolinha == 'o')&&(p->
      blocos[i]->esquerda->move == 0))){
00086                         colisao = 1;
00087                         break;
00088                     }
00089                 }
00090                 else{
00091                     colisao = 1;
00092                     break;
```

```
00093                 }
00094             }
00095             if(colisao != 1){
00096                 for (i = 0; i < p->tamanho; i++){
00097                     p->blocos[i]->cor = 2;
00098                     p->blocos[i]->bolinha = ' ';
00099                     p->blocos[i]->move = 0;
00100 
00101                     p->blocos[i] = p->blocos[i]->esquerda;
00102                     p->blocos[i]->cor = p->cor_peca;
00103                     p->blocos[i]->bolinha = 'o';
00104                     p->blocos[i]->move = 1;
00105                 }
00106             }
00107             break;
00108         default:
00109             break;
00110     }
00111 }
00112 
00117 void move_peca_y(peca* p, int y){
00118 
00119     int i;
00120     unsigned short int colisao = 0;
00121     unsigned short int limite_inferior =0;
00122 
00123     if (y>0){
00124         for (i = 0; i < p->tamanho; i++){
00125             if(p->blocos[i]->pos_y != 14){
00126                 if(((p->blocos[i]->abaixo->bolinha == 'o')&&(p->
      blocos[i]->abaixo->move == 0))){
00127                     colisao = 1;
00128                     break;
00129                 }
00130             }
00131             else{
00132                 colisao = 1;
00133                 break;
00134             }
00135         }
00136         if (colisao != 1){
00137             for (i = p->tamanho - 1; i >= 0; i--){
00138                 p->blocos[i]->cor = 2;
00139                 p->blocos[i]->bolinha = ' ';
00140                 p->blocos[i]->move = 0;
00141 
00142                 p->blocos[i] = p->blocos[i]->abaixo;
00143                 p->blocos[i]->cor = p->cor_peca;
00144                 p->blocos[i]->bolinha = 'o';
00145                 p->blocos[i]->move = 1;
00146                 if (p->blocos[i]->pos_y == 14){
00147                     limite_inferior = 1;
00148                 }
00149             }
00150         }
00151     }
00152 
00153     if (colisao==1 || limite_inferior == 1){
00154         for (i = 0; i < p->tamanho; i++){
00155             p->blocos[i]->move = 0;
00156         }
00157         p->move_peca = 0;
00158     }
00159 
00160 }
00161 
00166 void libera_peca(peca* p){
00167     free(p);
00168 }
```

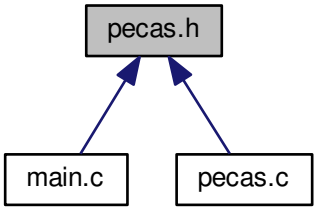
4.11 Referência do Arquivo pecas.h

```
#include <stdio.h>
#include <stdlib.h>
#include "tela.h"
```

Gráfico de dependência de inclusões para pecas.h:



Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com este arquivo:



Estruturas de Dados

- struct Peca

Definições de Tipos

- typedef struct Peca **peca**

Funções

- void nova_peca (Tela ∗tela)
- void move_peca_x (peca ∗peca, int x)
- void move_peca_y (peca ∗peca, int y)
- void libera_peca (peca ∗p)

4.11.1 Funções

4.11.1.1 void libera_peca (peca * p)

Libera a memória alocada para a peça.

Parâmetros

p	Ponteiro para a peça a ser liberada.

Definição na linha 166 do arquivo pecas.c.

4.11.1.2 void move_peca_x (peca * p, int x)

Movimenta a peça no eixo x, ou seja no sentido horizontal.

Parâmetros

p	A peça a ser movimentada.
x	Indica a direção do movimento. Se positivo, para a direita. Se negativo, para a esquerda.

Definição na linha 50 do arquivo pecas.c.

4.11.1.3 void move_peca_y (peca * p, int y)

Movimenta a peça no eixo y, ou seja, na direção vertical.

Parâmetros

p	Peça a ser movida.
y	Sempre deve ser positivo, pois a peça só pode se movimentar para baixo.

Definição na linha 117 do arquivo pecas.c.

4.11.1.4 void nova_peca (Tela * tela)

Gera nova peça do jogo. Orientação e tamanho são dados de forma pseudoaleatória. A cor é dada de forma cíclica.

Parâmetros

tela	Ponteiro para tela de jogo.

<Indica a orientação da peça. Se 1, a orientação é vertical. Se 0, a orientação é horizontal

<Indica o tamanho da peça

Definição na linha 14 do arquivo pecas.c.

4.12 pecas.h

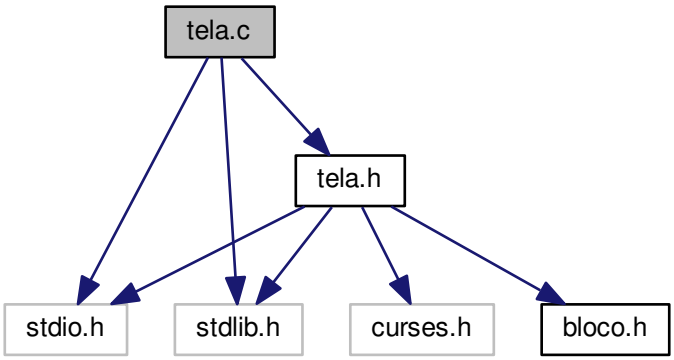
```
00001
00003 #include <stdio.h>
00004 #include <stdlib.h>
00005 #include "tela.h"
```

```
00006
00008 typedef struct Peca{
00009   int tamanho;
00010   unsigned short int cor_peca;
00011   unsigned short int move_peca;
00012   bloco* blocos[];
00013 }peca;
00014
00015 void nova_peca(Tela* tela);
00016 void move_peca_x(peca* peca, int x);
00017 void move_peca_y(peca* peca, int y);
00018 void libera_peca(peca* p);
```

4.13 Referência do Arquivo tela.c

```
#include <stdio.h>
#include <stdlib.h>
#include "tela.h"
```

Gráfico de dependência de inclusões para tela.c:



Funções

- Tela ∗ cria_tela ()
- void mostra_tela (Tela ∗t)
- void mostra_pontos (int pontos)
- void mostra_tempo (int minutos, int segundos)
- int verifica_linha (Tela ∗t)
- void limpa_linha (Tela ∗t, int y)
- void desce_linhas (Tela ∗t, int y)
- int checa_fim (Tela ∗t)
- void destroi_tela (Tela ∗t)

4.13.1 Funções

4.13.1.1 int checa_fim (Tela ∗ *t*)

Verifica se as peças ultrapassaram o limite superior do jogo.

Parâmetros

t	Ponteiro para a tela de jogo.

Retorna

Verdadeiro se ultrapassou o limite. Falso caso contrário.

Definição na linha 210 do arquivo tela.c.

4.13.1.2 Tela* cria_tela ( )

Cria uma tela de jogo com os parâmetros corretos.

Retorna

Retorna um ponteiro para tal tela.

Definição na linha 9 do arquivo tela.c.

4.13.1.3 void desce_linhas (Tela * *t*, int *y*)

Desce determinada linha da tela.

Parâmetros

t	Ponteiro para a tela de jogo.
y	Posição para a linha.

Definição na linha 186 do arquivo tela.c.

4.13.1.4 void destroi_tela (Tela * *t*)

Libera o espaço de memória reservado para a tela de jogo.

Parâmetros

t	Ponteiro para a tela de jogo.

Definição na linha 224 do arquivo tela.c.

4.13.1.5 void limpa_linha (Tela * *t*, int *y*)

Limpa uma determinada linha do jogo.

Parâmetros

t	Ponteiro para a tela do jogo.
y	Posição da linha a ser eliminada.

Definição na linha 171 do arquivo tela.c.

4.13.1.6 void mostra_pontos (int *pontos*)

Mostra a pontuação do jogados.

Parâmetros

pontos	O escore atual.

Definição na linha 110 do arquivo tela.c.

4.13.1.7 void mostra_tela (Tela * *t*)

Mostra a tela de jogo, conforme seu atual estado. Também inicializa os pares de cores a serem utilizados.

Parâmetros

t	Ponteiro para tela a ser mostrada.

Definição na linha 48 do arquivo tela.c.

4.13.1.8 void mostra_tempo (int *minutos*, int *segundos*)

Mostra o tempo da partida.

Parâmetros

minutos	Tempo em minutos.
segundos	Tempo em segundos.

Definição na linha 125 do arquivo tela.c.

4.13.1.9 int verifica_linha (Tela * *t*)

Verifica se uma linha horizontal do jogo está completamente preenchida.

Parâmetros

t	Ponteiro para a tela do jogo.

Retorna

100 se a linha estiver preenchida. 0 caso contrário.

Definição na linha 144 do arquivo tela.c.

4.14 tela.c

```
00001
00003 #include <stdio.h>
00004 #include <stdlib.h>
00005 #include "tela.h"
00006
00009 Tela* cria_tela(){
00010
00011     int i,j;
00012     Tela* t = malloc(sizeof(Tela) + COMPRIMENTO*LARGURA*sizeof(bloco));
00013
00014     t->comprimento = COMPRIMENTO;
00015     t->largura = LARGURA;
00016     t->estado = INICIO;
00017     for(i=0; i < t->comprimento; i++){
00018         for(j=0; j < t->largura; j++){
00019             t->blocos[j+i*t->largura].bolinha = ' ';
00020             t->blocos[j+i*t->largura].move = 0;
00021             t->blocos[j+i*t->largura].pos_x = j;
00022             t->blocos[j+i*t->largura].pos_y = i;
00023             if(j!=0)
00024                 t->blocos[j+i*t->largura].esquerda = &(t->
      blocos[j-1+i*t->largura]);
00025             else
00026                 t->blocos[j+i*t->largura].esquerda = NULL;
00027             if(j!= t->largura - 1)
00028                 t->blocos[j+i*t->largura].direita = &(t->
      blocos[j+1+i*t->largura]);
00029             else
00030                 t->blocos[j+i*t->largura].direita = NULL;
00031             if(i!= t->comprimento - 1)
00032                 t->blocos[j+i*t->largura].abaixo = &(t->
      blocos[j+(i+1)*t->largura]);
00033             else
00034                 t->blocos[j+i*t->largura].abaixo = NULL;
00035         }
00036     }
00037
00038     t->janela = newwin(t->comprimento + 2, t->largura + 2, 5, 10);
00039     box(t->janela, ACS_VLINE, ACS_HLINE);
00040     return t;
00041 }
00042
00048 void mostra_tela(Tela* t){
00049
00050     init_pair(1,COLOR_BLACK,COLOR_BLUE);
00051     init_pair(2,COLOR_WHITE,COLOR_BLACK);
00052     init_pair(3,COLOR_BLACK,COLOR_WHITE);
00053     init_pair(4,COLOR_YELLOW,COLOR_BLACK);
00054     init_pair(5,COLOR_BLUE,COLOR_BLACK);
00055     init_pair(6,COLOR_RED,COLOR_BLACK);
00056     init_pair(7,COLOR_GREEN,COLOR_BLACK);
00057
00058
00059     clear();
00060     refresh();
00061
00062     if(t->estado == INICIO){
00063
00064         bkgd(COLOR_PAIR(1));
00065         move(1,1);
00066         printw("Pressione Qualquer Tecla Para Iniciar o Jogo.");
00067
00068     }
00069
00070     else if(t->estado == JOGO){
00071         bkgd(COLOR_PAIR(2));
00072         refresh();
00073         wbkgd(t->janela,COLOR_PAIR(2));
00074         int i,j;
00075         mvprintw(11,8,"=>");
00076         mvprintw(11,37,"<=");
00077         refresh();
00078         move(0,0);
00079         for(i = 0; i < t->comprimento; i++){
00080             for(j = 0; j < t->largura; j++){
00081                 wattron(t->janela,COLOR_PAIR(t->blocos[j+i*t->
      largura].cor));
00082                 mvwprintw(t->janela,i+1,j+1,&(t->blocos[j+i*t->
      largura].bolinha));
00083                 wrefresh(t->janela);
00084             }
00085             /*wprintw(t->janela,"\n");*/
00086         }
00087     }
```

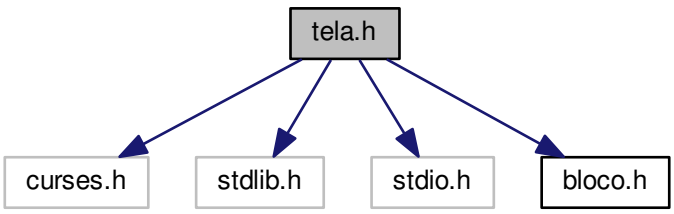
```
00088
00089     else if(t->estado == FINAL){
00090         char tempo_m[15], tempo_s[15], pontos[15];
00091         sprintf(tempo_m,"%d",t->tempo_m);
00092         sprintf(tempo_s,"%d",t->tempo_s);
00093         sprintf(pontos,"%d",t->pontos);
00094         bkgd(COLOR_PAIR(3));
00095         mvprintw(1,1,"Fim de Jogo :c");
00096         mvprintw(2,1,"Pontuação-> ");
00097         mvprintw(2,15,pontos);
00098         mvprintw(3,1,"Tempo->");
00099         mvprintw(3,9,tempo_m);
00100         mvprintw(3,11,":");
00101         mvprintw(3,13,tempo_s);
00102         mvprintw(5,1,"Pressione Qualquer Tecla para finalizar o jogo.");
00103     }
00104     refresh();
00105 }
00106
00110 void mostra_pontos(int pontos){
00111     WINDOW* janela;
00112     char str[15];
00113     snprintf(str,15,"%d",pontos);
00114     janela = newwin(1,20,5,50);
00115     wclear(janela);
00116     mvwprintw(janela,0,0,"Pontuação:");
00117     mvwprintw(janela,0,15,str);
00118     wrefresh(janela);
00119 }
00120
00125 void mostra_tempo(int minutos,int segundos){
00126     WINDOW* janela;
00127     char str_m[15],str_s[15];
00128     sprintf(str_m,"%d",minutos);
00129     sprintf(str_s,"%d",segundos);
00130     janela = newwin(1,20,6,50);
00131     wclear(janela);
00132     mvwprintw(janela,0,0,"Tempo:");
00133     mvwprintw(janela,0,10,str_m);
00134     mvwprintw(janela,0,12,":");
00135     mvwprintw(janela,0,14,str_s);
00136     wrefresh(janela);
00137 }
00138
00144 int verifica_linha (Tela* t){
00145
00146     int x, y;
00147     int counter = 0;
00148     int points = 0;
00149
00150     for(y = t->comprimento - 1;  y >= 0; y--){
00151         for(x = 0; x < t->largura; x++){
00152             if (t->blocos[x+y*t->largura].bolinha == 'o'){
00153                 counter++;
00154             }
00155         }
00156         if (counter == 25){
00157             limpa_linha(t, y);
00158             desce_linhas(t, y);
00159             points += 100;
00160             y++;
00161         }
00162         counter = 0;
00163     }
00164     return points;
00165 }
00166
00171 void limpa_linha (Tela* t, int y){
00172
00173     int x;
00174
00175     for (x = 0; x < t->largura; x++){
00176         t->blocos[x+y*t->largura].bolinha = ' ';
00177         t->blocos[x+y*t->largura].cor = 2;
00178     }
00179 }
00180
00186 void desce_linhas (Tela* t, int y){
00187
00188     int i,j;
00189     int counter=0;
00190
00191     for(i = y-1;  i >= 0; i--){
00192         for(j = 0; j < t->largura; j++){
00193             if(t->blocos[j+i*t->largura].bolinha == 'o'){
00194                 counter++;
00195             }
```

```
00196             t->blocos[j+(i+1)*t->largura].bolinha = t->
      blocos[j+i*t->largura].bolinha;
00197             t->blocos[j+(i+1)*t->largura].cor = t->blocos[j+i*t->
      largura].cor;
00198         }
00199         if(counter==0){
00200             break;
00201         }
00202         counter = 0;
00203     }
00204 }
00205
00210 int checa_fim(Tela* t){
00211     int i;
00212
00213     for(i = 0; i < t->largura; i++){
00214         if((t->blocos[i+5*t->largura].bolinha == 'o')&&(!t->
      blocos[i+5*t->largura].move)){
00215             return true;
00216         }
00217     }
00218     return false;
00219 }
00220
00224 void destroi_tela(Tela* t){
00225     free(t);
00226 }
00227
```

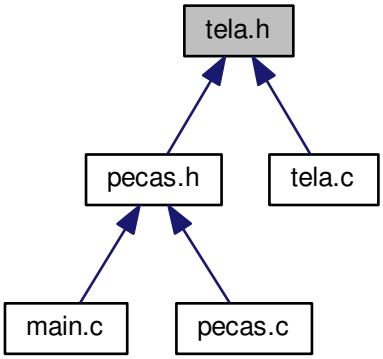
4.15 Referência do Arquivo tela.h

```
#include <curses.h>
#include <stdlib.h>
#include <stdio.h>
#include "bloco.h"
```

Gráfico de dependência de inclusões para tela.h:



Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com este arquivo:



Estruturas de Dados

- struct Tela

Definições de Tipos

- typedef struct Tela Tela

Enumerações

- enum estado { **INICIO**, **JOGO**, **FINAL** }

Funções

- Tela ∗ cria_tela ()
- void mostra_tela (Tela ∗t)
- void mostra_pontos (int pontos)
- void mostra_tempo (int minutos, int segundos)
- void destroi_tela (Tela ∗t)
- void limpa_linha (Tela ∗t, int y)
- void desce_linhas (Tela ∗t, int y)
- int verifica_linha (Tela ∗t)
- int checa_fim (Tela ∗t)

4.15.1 Definições dos tipos

4.15.1.1 typedef struct Tela Tela

/struct Define a tela do jogo.

4.15.2 Enumerações

4.15.2.1 enum estado

Variável enumerada que indica o estado do jogo.

Definição na linha 10 do arquivo tela.h.

4.15.3 Funções

4.15.3.1 int checa_fim (Tela * t)

Verifica se as peças ultrapassaram o limite superior do jogo.

Parâmetros

t	Ponteiro para a tela de jogo.

Retorna

Verdadeiro se ultrapassou o limite. Falso caso contrário.

Definição na linha 210 do arquivo tela.c.

4.15.3.2 Tela* cria_tela ( )

Cria uma tela de jogo com os parâmetros corretos.

Retorna

Retorna um ponteiro para tal tela.

Definição na linha 9 do arquivo tela.c.

4.15.3.3 void desce_linhas (Tela * t, int y)

Desce determinada linha da tela.

Parâmetros

t	Ponteiro para a tela de jogo.
y	Posição para a linha.

Definição na linha 186 do arquivo tela.c.

4.15.3.4 void destroi_tela (Tela * t)

Libera o espaço de memória reservado para a tela de jogo.

Parâmetros

t	Ponteiro para a tela de jogo.

Definição na linha 224 do arquivo tela.c.

4.15.3.5 void limpa_linha (Tela * t, int y)

Limpa uma determinada linha do jogo.

Parâmetros

t	Ponteiro para a tela do jogo.
y	Posição da linha a ser eliminada.

Definição na linha 171 do arquivo tela.c.

4.15.3.6 void mostra_pontos (int *pontos*)

Mostra a pontuação do jogados.

Parâmetros

pontos	O escore atual.

Definição na linha 110 do arquivo tela.c.

4.15.3.7 void mostra_tela (Tela * t)

Mostra a tela de jogo, conforme seu atual estado. Também inicializa os pares de cores a serem utilizados.

Parâmetros

t	Ponteiro para tela a ser mostrada.

Definição na linha 48 do arquivo tela.c.

4.15.3.8 void mostra_tempo (int *minutos*, int *segundos*)

Mostra o tempo da partida.

Parâmetros

minutos	Tempo em minutos.
segundos	Tempo em segundos.

Definição na linha 125 do arquivo tela.c.

4.15.3.9 int verifica_linha (Tela * t)

Verifica se uma linha horizontal do jogo está completamente preenchida.

Parâmetros

t	Ponteiro para a tela do jogo.

Retorna

100 se a linha estiver preenchida. 0 caso contrário.

Definição na linha 144 do arquivo tela.c.

4.16 tela.h

```
00001
00003 #include<curses.h>
00004 #include<stdlib.h>
00005 #include<stdio.h>
00006 #include"bloco.h"
00007
00008
00010 enum estado {INICIO,JOGO,FINAL};
00011
00013 typedef struct Tela{
00014   int estado;
00015   int pontos;
00016   int tempo_m;
00017   int tempo_s;
00018   int comprimento;
00019   int largura;
00020   WINDOW *janela;
00021   struct Peca* peca;
00022   bloco blocos[];
00023 }Tela;
00024
00025 Tela* cria_tela();
00026 void mostra_tela(Tela* t);
00027 void mostra_pontos(int pontos);
00028 void mostra_tempo(int minutos,int segundos);
00029 void destroi_tela(Tela* t);
00030 void limpa_linha (Tela* t, int y);
00031 void desce_linhas (Tela* t, int y);
00032 int verifica_linha(Tela* t);
00033 int checa_fim(Tela* t);
```

Índice Remissivo