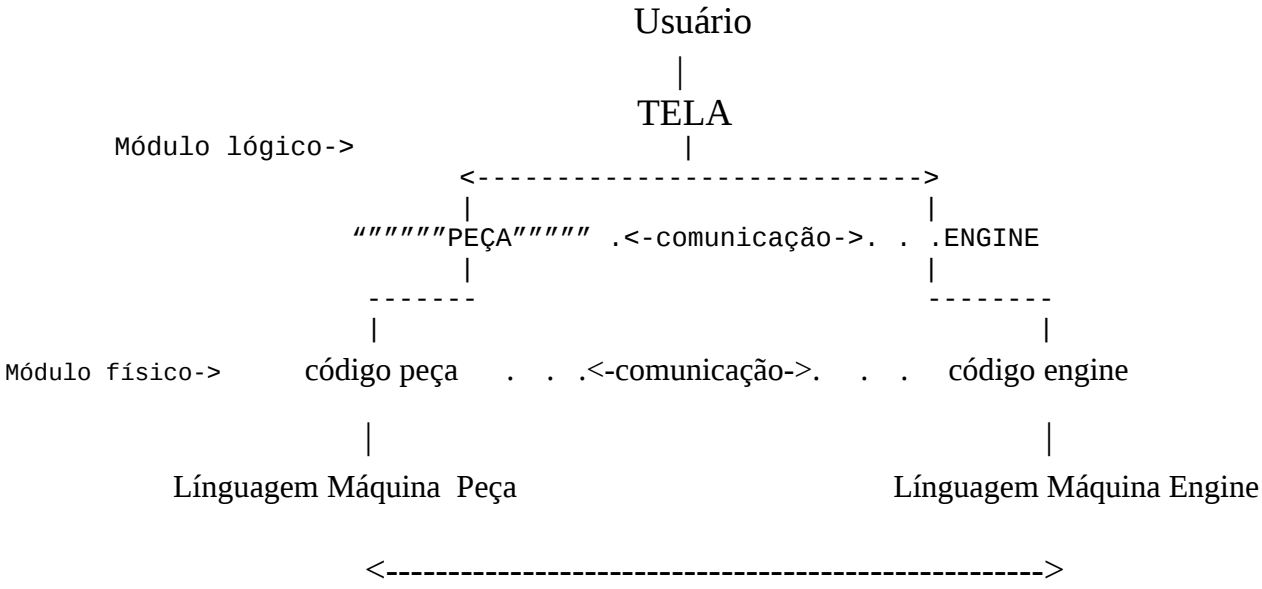
Documentação

Componentes do grupo:
Pedro Ivo Araújo – 15/0020121
Vitor Ribeiro Custodio – 15/0023502
Yago Pablo Lourenço Vaz – 15/0023928

Peças :



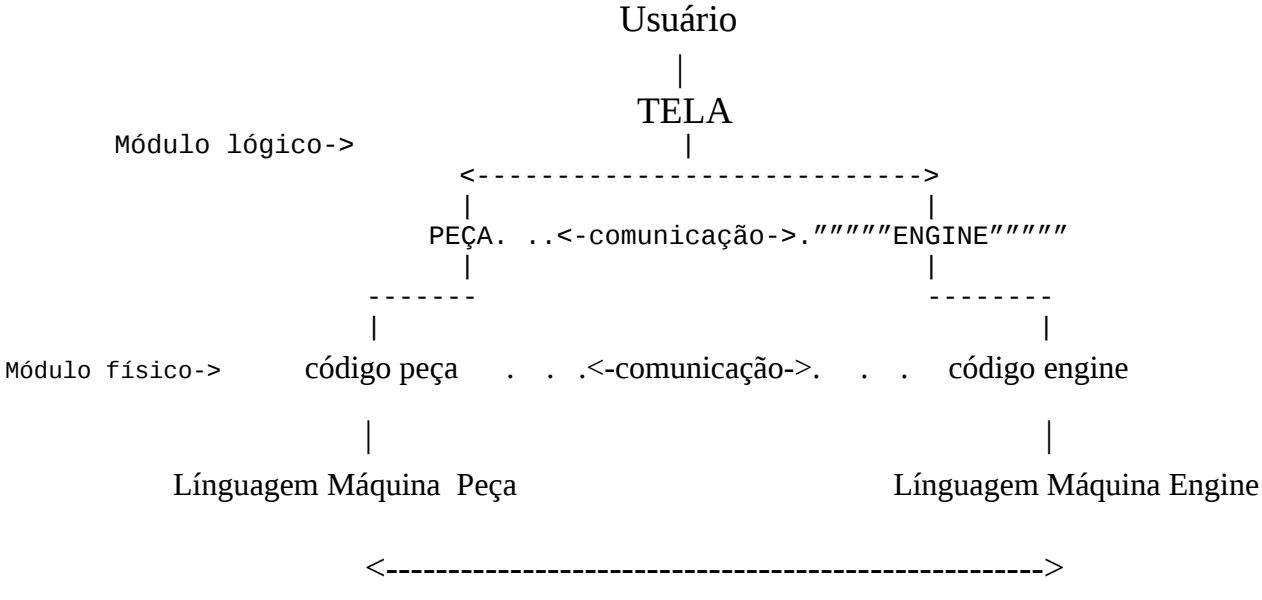
- O módulo peça é composto por 6 estruturas do tipo Struct e por 18 funções;
- Os tipos Struct são usados para definir os formatos das peças que serão usados, as especificações de localização de cada peça / bloco , as direções que podem ser movidas e as estruturas que compõe o tabuleiro do jogo;
- As peças são feitas a partir de strings , sendo necessário saber a localização da linha e coluna que ela ocupará ;
- A função get é responsável por retornar um bloco na linha e posição recebidas;
- A função set posiciona o bloco na localização desejada ;
- A função checar é responsável pela verificação de que a localização do bloco está dentro dos limites do tabuleiro ;
- A função put coloca um bloco no tabuleiro ;
- A função remover é responsável por remover um bloco do campo ;
- A função encaixe é atribuída o dever de checar se um bloco pode ser colocado em jogo ;
- A função random_peca retorna uma peça aleatória ;
- A função novo_cair criar um novo bloco e chama pelo próximo bloco aleatório que será despejado em campo;
- A função movimentar é atribuída o dever de movimentar o bloco entre esquerda e direita;
- A função cai é tem como objetivo pegar o bloco do campo e colocá-lo no fim do tabuleiro;
- A função hold tem a funcionalidade de trocar um bloco com outro que está no buffer ;
- A função handle_move realiza a movimentação especificada pela entrada do usuário , que é

atribuída as setas ;
- A função linha_cheia retorna um valor booleano caso a linha conferida esteja cheia;
- A função shift_lines limpa o tabuleiro quando está cheio;
- A função checa_linha tem a função de procurar linhas cheias, removê-las , e retornar a quantidade de linhas vazias ;
- A função atualiza_placar tem como responsabilidade o ajuste de pontos , quando cada linha é completada;
- A função jogo_terminou retorna um valor booleano informando o fim ou não da partida;

Exemplo de movimentação do jogo:

```
static void handle_move(tetris_game *obj, tetris_move move)
{
  switch (move) {
  case esquerda:
  movimentar(obj, -1);
   break;
   case direita:
  movimentar(obj, 1);
 break;
case soltar:
cai(obj);
break;
case segurar:
hold(obj);
break;
default:
  break;
 }
}
```

Engine:

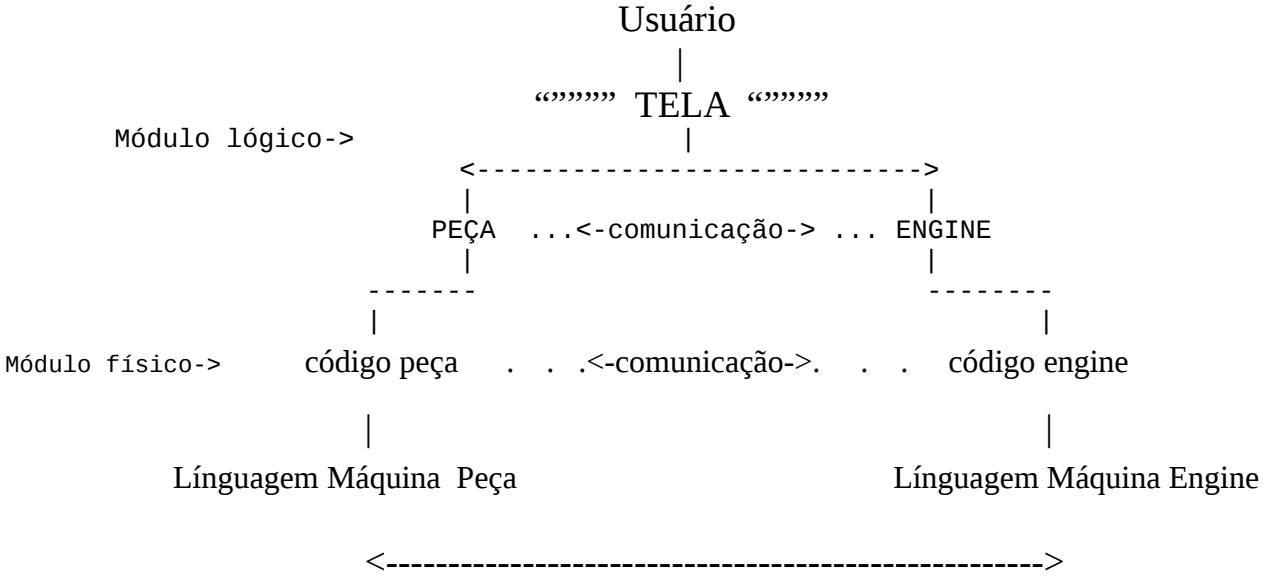


- O módulo engine tem a responsabilidade de criar e imprimir as condições impostas ao jogo, como tabuleiro , cores , placar , e tem funcionalidade de salvar o jogo na condição desejada ;
- A função tg_tick têm as funcionalidades de checar se o jogo continua sendo possível , de chamar a função para checar e atualizar o placar e de chamar as funções de movimentação das peças ;
- A função iniciar é responsável por apontar para a criação do tabuleiro e pela checagem de linhas vazias ;
- A função apagar apaga o tabuleiro no fim do jogo;
- A função delete tem a responsabilidade de apagar as peças do jogo;
- A função salvar criar um arquivo onde é salvo o seu jogo;
- A função print tem a responsabilidade de imprimir o tabuleiro no campo;
- A função mostra_tabuleiro utiliza de funções do ncurses para personalizar e melhorar o tabuleiro
- A função mostra_peca aponta para a criação das peças do jogo;
- A função mostra_pontuacao é responsável por retornar a pontuação do jogo;
- A função salva_e_sai cria um arquivo onde o jogo é salvo e logo depois o jogo é ejetado ;
- A função init_color é responsável por dar cores as peças do jogo;

Exemplo de uma das funcionalidade do engine que é mostrar o tabuleiro:

```
void print(tetris_game *obj, FILE *f) {
  int i, j;
  for (i = 0; i < obj->rows; i++) {
  for (j = 0; j < obj->cols; j++) {
  if (PC_IS_EMPTY(get(obj, i, j))) {
    fputs(PC_EMPTY_STR, f);
  } else {
    fputs(PC_BLOCK_STR, f);
  }
  }
  fputc('\n', f);
   }
}
```

Tela:



- O módulo tela é onde a função main se encontra , nele é declarado as bibliotecas usadas para a criação do código , assim como retorna os dois módulos descritos acima. As suas funcionalidades é garantir que as funções para o funcionamento do jogo serão chamadas , assim como a realizar um checagem na entrada do usuário para que as teclas certas sejam usadas com suas definidas usabilidades.

Bibiotecas usadas:

```
#include <stdio.h>
#include <ncurses.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <time.h>
#include "CUnit/CUnit.h"
#include "CUnit/Basic.h"
#include "peca.c"
#include "engine.c"
```

Exemplo de código da tela em favor da movimentação em geral:

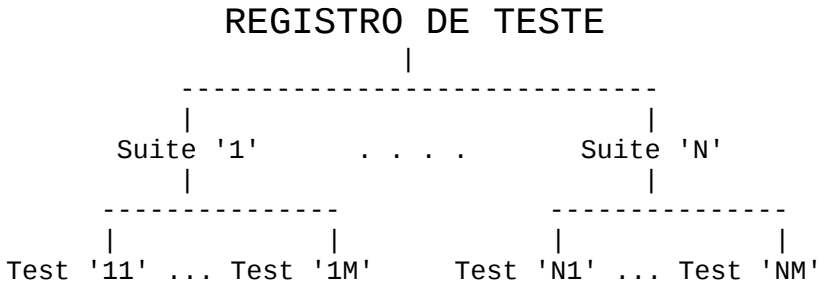
```
switch (getch()) {
   case KEY_LEFT:
     move = esquerda;
break;
   case KEY_RIGHT:
     move = direita;
     break;
   case KEY_DOWN:
     move = soltar;
     break;
     case 's':
     salva_e_sai(tg, board);
     move = none;
     break;
   case 'q':
     running = false;
     move = none;
     break;
   case 'p':
     wclear(board);
     box(board, 0, 0);
     wmove(board, tg->rows/2, (tg->cols*COLS_PER_CELL-6)/2);
     wprintw(board, "PAUSED");
     wrefresh(board);
```

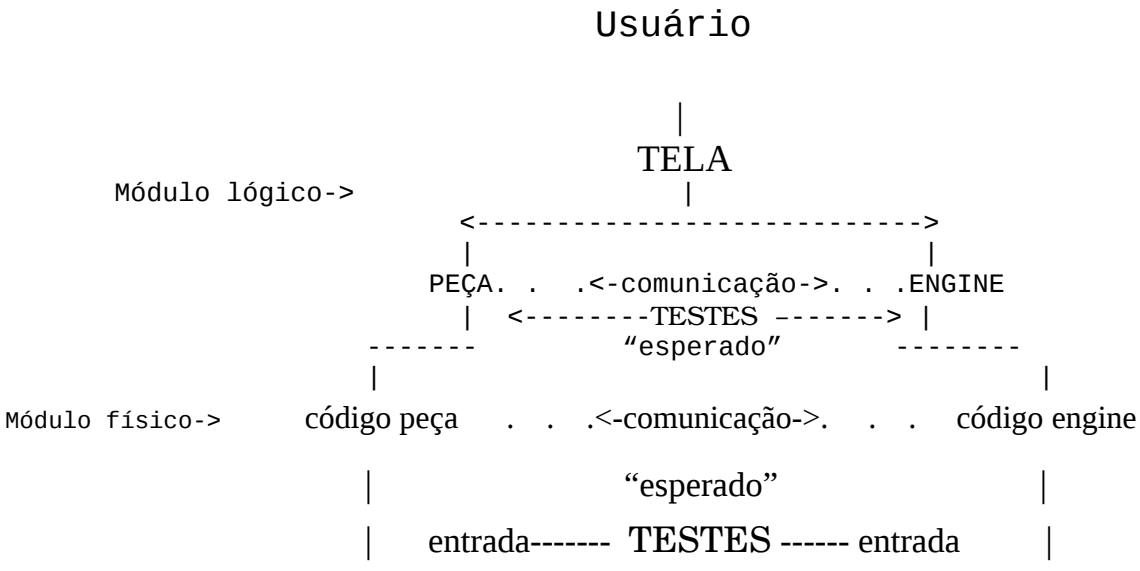
```
    timeout(-1);
    getch();
    timeout(0);
    move = none;
    break;

  case ' ':
    move = segurar;
    break;
  default:
    move = none;
  }
}
```

Testes:

- Os testes foram feitos visando á funcionalidade de cada função:





Línguagem Máquina Peça Línguagem Máquina Engine

<--->

- Casos de teste individual são empacotados em suítes, que são registrados com o registro de teste ativo. Suites podem ter funções de instalação e desmontagem que são automaticamente chamados antes e após a execução de testes da suíte . Todas as suites / testes no registro pode ser executado usando uma única chamada de função, ou selecionados suites ou testes podem ser executados .

Uma sequência típica de passos para utilizar o framework Cunit é:
Escrever funções para testes (e inicialização suite / limpeza se necessário).
Inicializar o registro de teste - CU_initialize_registry ()
Adicionar suites para o registro de teste - CU_add_suite ()
Adicionar testes para as suites - CU_add_test ()
Executar testes usando uma interface apropriada , por exemplo, CU_console_run_tests
Limpeza do registro de teste – CU_cleanup_registry

Exemplo de testes:

```
/*Testa se a função checar é falsa*/
void teste_DT_checar_Checachecar2(void){
bool resultado;
resultado = checar;
resultado = false;
CU_ASSERT(!resultado);
}
CU_ADD_TEST(suite, teste_DT_checar_Checachecar2);
```

```
/*Inicializa o registro de suítes e testes do CUnit*/
if (CUE_SUCCESS != CU_initialize_registry())
        return CU_get_error();

/*Adiciona os testes ao registro*/
adicionar_suite();

/*Muda o modo do CUnit para o modo VERBOSE
 O modo VERBOSE mostra algumas informacoes a
 mais na hora da execucao. Outras opções: NORMAL e SILENT*/
CU_basic_set_mode(CU_BRM_VERBOSE);

/*Roda os testes e mostra na tela os resultados*/
CU_basic_run_tests();
//CU_console_run_tests();
//CU_automated_run_tests();


/*Limpa o registro*/
CU_cleanup_registry();

return CU_get_error();
```