UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
BACHARELADO EM SISTEMAS DA INFORMAÇÃO

HELOISE CRISTINI MAGATON
OSMARY CAMILA BORTONCELLO GLOBER
VINICIUS FRANCISCO DE PAULA

TETRIS E COLUMNS

PROJETO INTEGRADO - 1º PERÍODO

CURITIBA/2011

HELOISE CRISTINI MAGATON
OSMARY CAMILA BORTONCELLO GLOBER
VINICIUS FRANCISCO DE PAULA

TETRIS E COLUMNS

Documentação do Projeto Integrado, apresentado ao curso de Bacharelado em Sistemas de Informação do Departamento Acadêmico de Informática – DAINF - Universidade Tecnológica Federal do Paraná.

CURITIBA/2011

RESUMO

Este trabalho teve como objetivo principal a criação, implementação, desenvolvimento e funcionamento de uma união do jogo tetris e columns, além do projeto da linguagem de programação C e suas bibliotecas. O trabalho ainda traz uma especificação mais detalhada sobre os sistemas que compõe o jogo, com representação de seus modelos, usando diagramas de fluxo de dados (DFD).

Palvras-Chave: Tetris.Columns. Jogo em C. Projeto de jogo na linguagem C.

ABSTRACT

This work had as main objective the creation, implementation, development and operation of a union of tetris and columns, and design of the programming language (C) and its libraries. The work also provides a more detailed specification on the systems that make up the game, with representation of their models, using data flow diagrams (DFD).

Keywords: Tetris. Columns. Game C. Game design in C language

SUMÁRIO

INTRODUÇÃO

A proposta do Projeto Integrado, através da interação entra as matérias estudadas pelos alunos de Bacharelado em Sistemas de Informação, é combinar regras dos jogos Tetris e Columns. O jogo Tetris(considerado um dos dez jogos mais influentes de todos os tempos) foi criado ainda na União Soviética. É um jogo eletrônico cujas as regras assemelham-se a um quebra-cabeça, com o intuito de encaixar, da melhor maneira, as peças que caem do topo da tela. Notamos uma semelhanç entre esses dois jogos, uma vez que no Columns, as peças também descem do topo da tela, porém, em vez de montarmos uma espécie de quebra-cabeça, como no Tetris, deve-se montar, da melhor forma, um quebra-cabeça de cores.

CAPÍTULO 1 – Definição do Problema

1.1 Objetivo

O objetivo do jogo é de unificar os jogos Tetris e Columns mantendo seus conceitos já estabelecidos, ou seja, a idéia principal é trabalhar com modificações que resultarão num jogo totalmente novo, porém com regras conhecidas, estas oriundas do Tetris e Columns.

1.2 Escopo

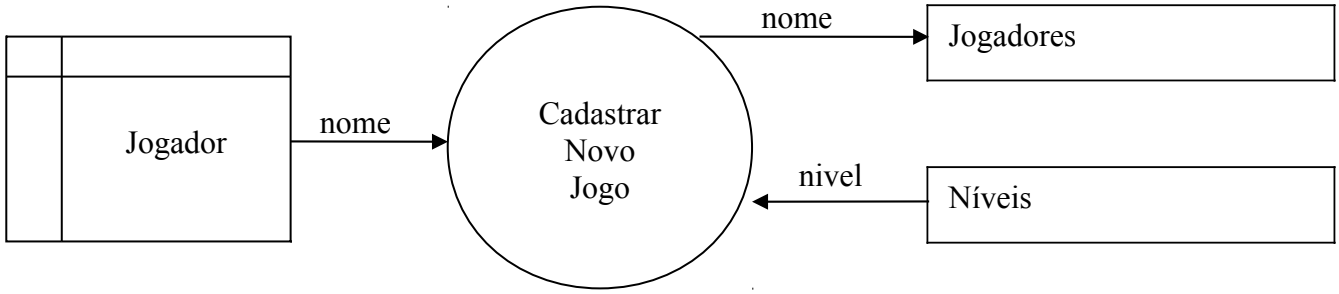
O sistema do jogo Tetris+Colums realiza as fuções necessárias para o bom funcionamento do jogo.

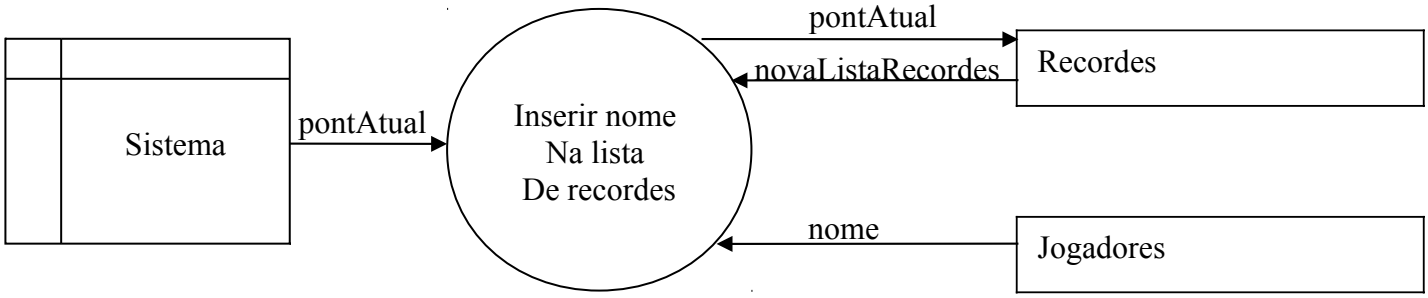
O sistemas define as peças, sua rotação, movimentação e colisão, além da atribuição de cores, pontuação e mudança de nível. As melhores pontuações também são armazenadas em arquivos recordes, os quais exibem para o usuário as cinco melhores pontuações.

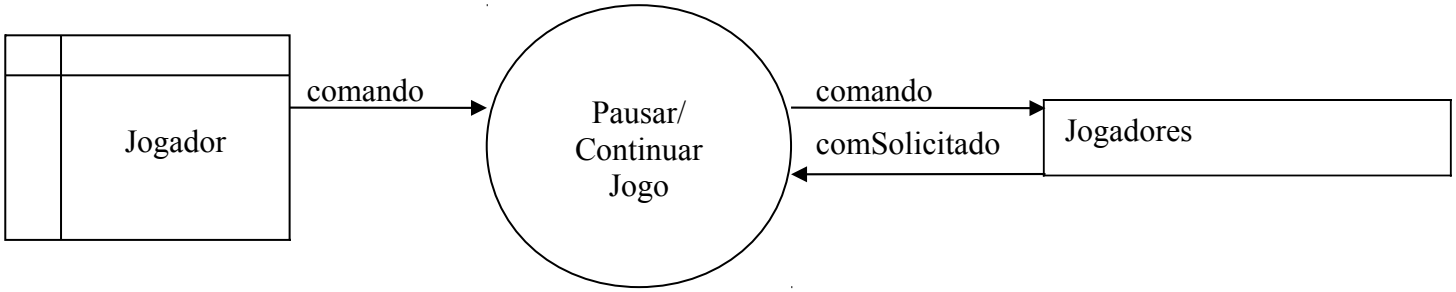
Através das definições o objetivo do jogo pode ser atingido, o qual as peças coloridas devem se encaixar e eliminar as de mesma cor.

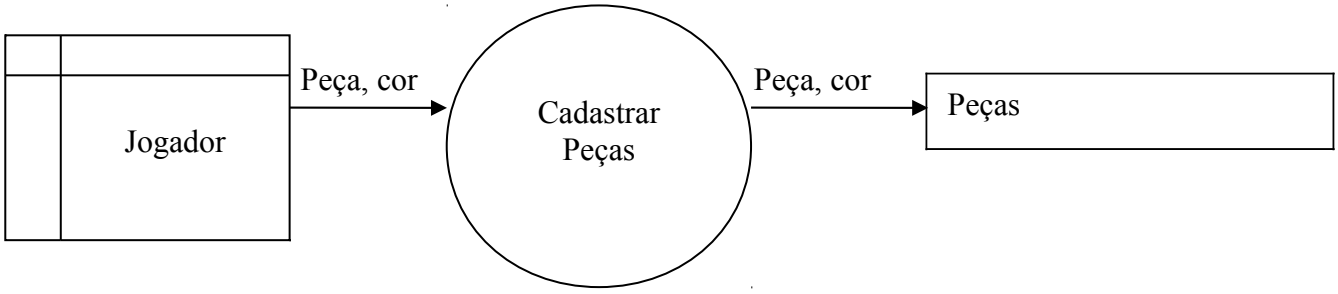
CAPÍTULO 2 – Especificação usando DFD's

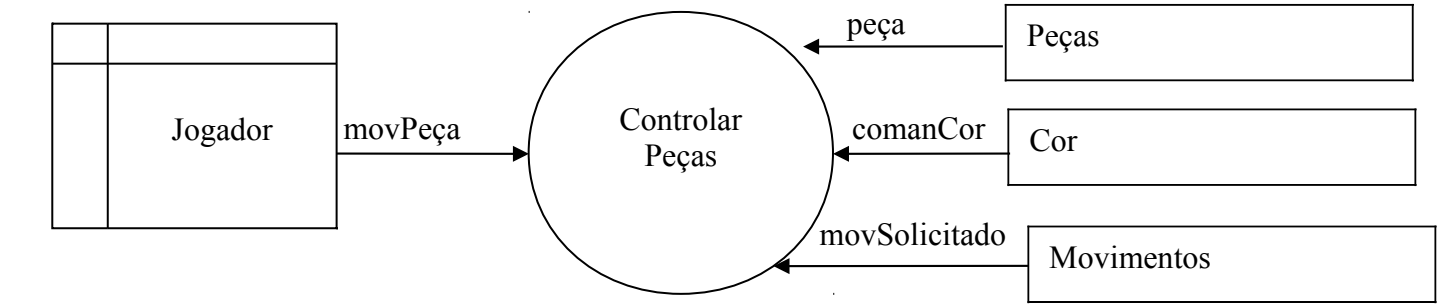
2.1 DFD's de nível 1

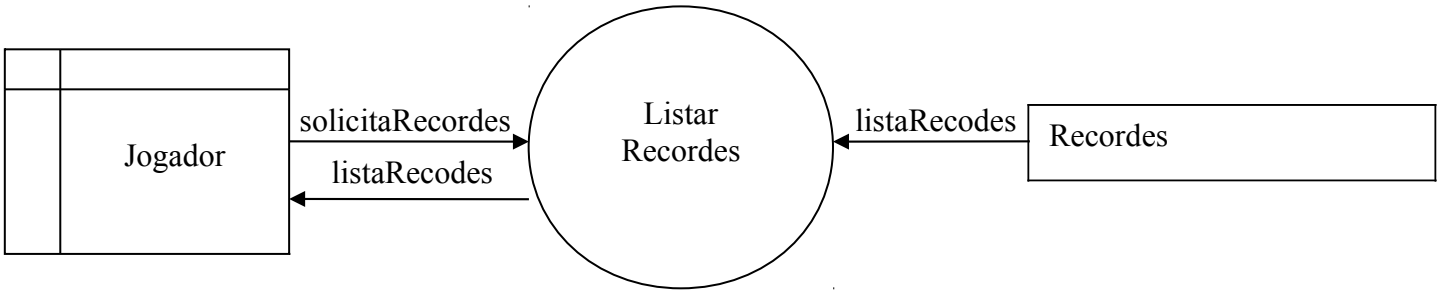


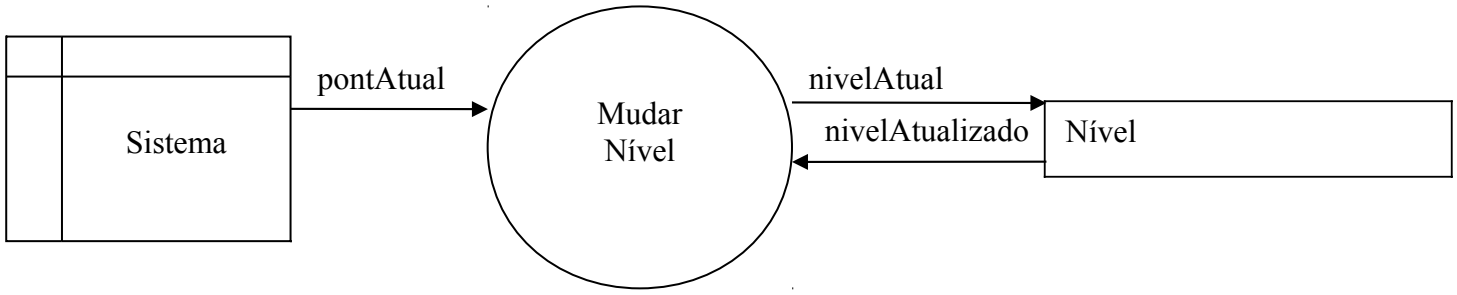


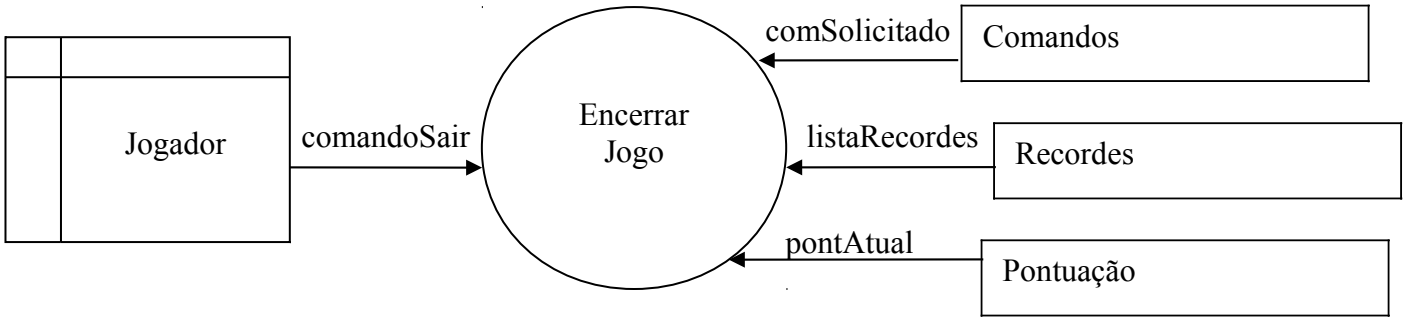












- Depósitos:

Jogadores: [nomeJogador].

Nível: [nível1, nível2, nível3, nível4, nível5].

Recordes: [listaRecordes].

Pontuação: [pontAtual].

Peças: [pec1, pec2, pec3, pec4, pec5, pec6, peça].

Cores: [cor1, cor2, cor3, cor4].

Movimentos: [mov1, mov2, mov3, mov4, movSolicitado].

Comandos: [comPausar, comContinuar, comListaRecordes, comEncerrarJogo].

2.2 Descrição informal de cada processo

Cadastrar Jogo: Recebe o fluxo nome, efetua o cadastro do jogador, enviando este fluxo para o depósito jogadores e inicia o nível.

Inserir nome Na lista De recordes: O sistema envia o fluxo pontAtual e o processo insere o nome do jogador no depósito jogadores e envia e atualiza a pontuação e a lista de recordes.

Pausar/Continua Jogo: O jogador envia o fluxo comando para o processo, que retorna o comando solicitado.

Cadastrar Peças: O jogador envia o fluxo Peça, cor e o processo efetua o cadastro com este fluxo no depósito Peças.

Controlar Peças: O jogador envia o fluxo movPeça e o processo controla as peças retornando dos fluxos peça, cor, movSolicitado dos depósitos Peças, Cores, Movimento.

Listar Recordes: O jogador envia o comando para o processo. Este retorna uma lista com os recordes para o jogador , utilizando o fluxo listaRecordes do depósito Recordes.

Mudar de Nível: O sistema envia o fluxo pontAtual para o sistema, que efetua a mudança do nível, utilizando os fluxos nivelAtual, nivelAtualizado, estes do depósito Nível.

Encerrar Jogo: O jogador envia o fluxo comandoSair para o processo, que encerra o jogo, utilizando os fluxos comSolicitado, pontAtual, listaRecordes estes dos depósitos Comandos, Recordes e Pontuação.

CAPÍTULO 3 - Especificação Formal

3.1 Nome do sistema

Tetris+Columns

3.2 Tipos do Sistema

Tipos básicos utilizados: **STRING, INTEIRO**

MATRIZ = INTEIRO × INTEIRO → INTEIRO

RECORDISTA = <nome: STRING, pontuacao: INTEIRO>

3.3 Estado do Sistema

- pontuacao: INTEIRO
- larguraCampo: INTEIRO
- alturaCampo: INTEIRO
- campoDeJogo: MATRIZ
- nivel: INTEIRO
- tempoQuedaMilisegundos: INTEIRO
- nomeJogador: STRING
- recordistas: INTEIRO → RECORDISTA

3.3.1 Inicialização do Estado do Sistema

// valores que se mantêm constantes ao longo do jogo

larguraCampo = 10

alturaCampo = 40

numeroIndicadorCelulaVazia = 0

// valores que variam ao longo do jogo

pontuacao = 0

nivel = 1

tempoQuedaMilisegundos = 1500

campoDeJogo = *nova* MATRIZ [larguraCampo, alturaCampo]

nomeJogador = “”

recordistas = { } // lista vazia

3.4 Funções e predicados definidos

3.4.1 Funções definidas

- tamanho(**palavra**: STRING): retorna a quantidade de caracteres de **palavra**.

- quocienteInteiro (**dividendo**: INTEIRO, **divisor**: INTEIRO): retorna o quociente da divisão inteira de **dividendo** por **divisor**.

- potencia (**base:** INTEIRO, **expoente:** INTEIRO): retorna o resultado da multiplicação da **base** pelo número de vezes do **expoente.**

3.5 Especificação Formal das Operações

3.5.1 Inicia Novo Jogo

ΔTetris+Columns

nomeJogador?: STRING

Pré-condições:

// o nome do jogador deve ter no mínimo 1 caracter

tamanho(nomeJogador?)>=1

// o nome do jogador deve ter no máximo 20 caracteres

tamanho(nomeJogador?)<=20

Pós-condições:

// A especificação deve verificar se a operação reiniciou os valores que variam ao longo do jogo de acordo com a Inicialização do Sistema

// a pontuacao deve ser zero

pontuacao == 0

// o nivel deve ser 1

nivel == 1

// o tempo de queda deve ser 1500 ms

tempoQuedaMilisegundos == 1500

// o campo de jogo deve estar vazio

∀x ∀y

(x >= 1 ∧ x <= larguraCampo ∧ y >= 1 ∧ y <= alturaCampo)

→

campoDeJogo[x,y] == numeroIndicadorCelulaVazia

// o nome do jogador no estado do sistema é o nome recebido como entrada

nomeJogador == nomeJogador?

3.5.2 Muda Nível Jogador

// operação executada toda vez que a pontuacao do jogador muda

ΔTetris+Columns

Pré-condições:

pontuacao >= 500

Pós-condições:

nivel == quocienteInteiro(pontuacao, 500) + 1

tempoQuedaMilisegundos = potencia(0.9, nivel-1) * 1500

3.5.3 Ver recordes

ΞTetris+Columns
saida!: STRING

Pré-condições:
// o tamanho da lista de recordistas deve ser ao menos um
#recordes! >=1

Pós-condições:
// saida! deve conter os nomes e a pontuacao dos 5 melhores jogadores
saida! -> [nome,pontuação] = 5

CAPÍTULO 4 – Algoritmos Implementados e suas Descrições

- Incluindo bibliotecas necessárias para a implementação do programa:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>
#include <math.h>
#include “pecas.h”
```

- Definindo valores inicias para variáveis:

```
#define BORDA 178
#define VINICIAL 1500
#define TROCANIVEL 500
```

- Procedimentos e funções utilizados na implementação do jogo:

- Procedimento para mostrar o menu do usuário.Não recebe nada como parâmetro. O algoritmo mostra o menu inicial para o jogador.

```
void imprimeMenu ()
{
    printf("\t\t TETRIS+COLLUMS\n\n\n");
    printf("Selecione uma opcao.\n\n");
    printf("1 - Novo jogo.\n");
    printf("2 - Continuar jogo.\n");
    printf("3 - Ver recordes.\n");
    printf("4 - Modo Automático.\n");
    printf("5 - Sair.\n\n");

}
```

- Procedimento para inicializar o tabuleiro, incluindo as bordas, recebendo como parâmetro duas variáveis apontadoras do tipo inteiro. O algoritmo percorre toda a matriz e atribui valores iniciais às suas posições para iniciar o jogo.

```
void inicializaTabuleiro(int *pontos, int *nivel)
{
int l, c;
```

```
 *pontos=0;
   *nivel=1;

   for(c=0; c<12; c++){
      tabuleiro[41][c]=9;
      tabuleiro[0][c]=9;
   }

   for(l=0; l<42; l++) {
      tabuleiro[l][0]=9;
      tabuleiro[l][11]=9;
   }
   for(l=0; l<41; l++){
      for(c=1; c<11; c++){
         tabuleiro[l][c]=0;
      }
   }
}
```

- Procedimento utilizado para zerar o tabuleiro, não recebendo nenhum valor como parâmetro. Ele percorre a matriz inteira e compara as posições para zerar somente a matriz e não as peças nem as bordas, utilizando os critérios de que a posição a ser comparada tenha que ser diferente dos números 5, 6, 7, 8 e 9.

```
void zerarTabuleiro()
{
int l, c;

   for(l=0; l<42; l++)
      for(c=0; c<12; c++)
         if((tabuleiro[l][c]!=5)&&(tabuleiro[l][c]!=6)&&(tabuleiro[l][c]!=7)&&(tabuleiro[l][c]!
=8)&&(tabuleiro[l][c]!=9))
         tabuleiro[l][c]=0;
}
```

- Procedimento que muda o valor das peças para depois poder comparar. Não recebe nada como parâmetro. Muda o valor das posições da matriz para poder depois comparar em outras funções.O algoritmo percorre a matriz e muda o valor da posição.

Se o valor da posição for igual ao número 1,

esta receberá 5. Se for igual ao número 2, esta receberá 6. Se for igual ao número 3 esta receberá 7.

Se for igual ao número 4, esta receberá 8.

```
void transferirJogo()
{
   int l, c;
   for(l=0; l<42; l++)
      for(c=0; c<12; c++) {
         if(tabuleiro[l][c]==1)
            tabuleiro[l][c]=5;
         else if(tabuleiro[l][c]==2)
            tabuleiro[l][c]=6;
         else if(tabuleiro[l][c]==3)
            tabuleiro[l][c]=7;
         else if(tabuleiro[l][c]==4)
            tabuleiro[l][c]=8;
         }
}
```

- Procedimento para imprimir a matriz na tela com as bordas. Recebe por parâmetro duas variáveis do tipo inteiro.O algoritmo percorre a matriz, imprimindo a borda no lugar de 9,o número 1 no lugar do 5, o número 2 no lugar do 6, o número 3 no lugar do 7, o número 4 no lugar do 8, e espaço no lugar do 0.Imprime também os pontos do jogador e o nível em qual se encontra.

```
void imprimeMatriz(int pontos, int nivel)
{
   int l,c,i;
   printf("\tTETRIS+COLLUMS\n\n");
   //borda superior
   for(i=0;i<12;i++)
      printf("%2c", BORDA);
   printf("\n");

   for(l=0; l<42; l++) {
      for(c=0; c<12; c++) {
```

```
            if(tabuleiro[l][c]==9)
               printf("%2c", BORDA);
            else if(tabuleiro[l][c]==5)
               printf("%2d", 1);
            else if(tabuleiro[l][c]==6)
               printf("%2d", 2);
            else if(tabuleiro[l][c]==7)
               printf("%2d", 3);
            else if(tabuleiro[l][c]==8)
               printf("%2d", 4);
            else if(tabuleiro[l][c]!=0)
               printf("%2d", tabuleiro[l][c]);
            else
               printf("%2c", ' ');
         }
         printf("\n");
      }
      printf("\n");

      printf("\tPontos: %d\n", pontos);
      printf("\tNivel: %d\n\n\n", nivel);
}
```

- Função do tipo inteira para mudar a posição da peça. Recebe por parâmetro duas variáveis do tipo inteiro, uma com o número da peça e outra com sua posição. O algoritmo compara a posição da peça. Se a variável posição for igual a 3, ela receberá 0 e chamará outro procedimento.

```
int mudarPosicao(int nroPeca, int posicao)
{
   int l, c;
   if(posicao==3) {
      posicao=0;
      coordenadas(nroPeca, posicao);
      for(l=0; l<4; l++)
         for(c=5; c<12; c++)
            if(tabuleiro[posX+pos[l]+1][posY+pos[l+4]]==c)
               return posicao+3;
```

```
}
   else {
      posicao++;
      coordenadas(nroPeca, posicao);
      for(l=0; l<4; l++)
         for(c=5; c<12; c++)
            if(tabuleiro[posX+pos[l]+1][posY+pos[l+4]]==c)
               return posicao-1;
   }
   return posicao;
}
```

- Procedimento para mudar a pontuação do jogador. Recebe como parâmetro duas variáveis do tipo inteiro, sendo que uma delas é do tipo apontadora, que representa a pontuação do jogador e a outra é o caso a ser escolhido para aumentar a pontuação.O algoritmo implementado utiliza o critério proposto no projeto para a pontuação, utilizando como base o jogo Columns.

```
int modificarPontos(int *pontos, int n)
{
   switch(n){
      case 3:
         *pontos=*pontos+10;
         break;
      case 4:
         *pontos=*pontos+20;
         break;
      case 5:
         *pontos=*pontos+30;
         break;
      case 6:
         *pontos=*pontos+50;
         break;
   }
   return 0;
}
```

- Procedimento para mudar de nível, caso a pontuação seja maior que 500. Recebe como parâmetro duas variáveis do tipo apontadoras, uma com a pontuação do jogador e outra com o nível em que se encontra. O algoritmo compara os pontos com o nível, se for maior, a variável apontadora *nível é

incrementada em um.

```
void mudarNivel(int *pontos, int *nivel)
{
   if(*pontos>*nivel*TROCANIVEL){
      (*nivel)++;
   }
}
```

- Função do tipo inteira para ver se o critério de fim de jogo do Tetris acontece. Não recebe nada como parâmetro. O algoritmo percorre a matriz e verfica se há algum bloco que varia de 5 a 8. Se não houver o jogo continua. Se houver, a função retorna 1, que é o critério na estrutura de repetição utilizada para finalizar o jogo.

```
int fimJogo()
{
int c, peca;

   for(c=1; c<12; c++){
      for(peca=5; peca<9; peca++){ // verifica se há um bloco que varia de 5 a 8
         if(tabuleiro[0][c]==peca){
            return 1;
         }
      }
   }

   return 0;
}
```

- Procedimento utilizado para ler recordes. Recebe por parâmetro uma variável ponteira do tipo jogador, para poder modificar os valores nesta. O algoritmo abre o arquivo, onde ficam armazenados os recordes. Caso não haja recordes salvos, imprime-se valores “nulos do arquivo”.

```
void lerRecordes(Jogador *player)
{
int i;
FILE *record;

   record=fopen("Recordes.bin", "rb");

   if(record==NULL){
```

```
      for(i=0; i<5; i++) {
         strcpy(player[i].nome, "-");
         player[i].pontos=0;
      }
      record=fopen("Recordes.bin", "wb");
      fwrite(player,sizeof(Jogador),5,record);
   }

   else
      fread(player,sizeof(Jogador),5,record);

   fclose(record);
}
```

- Procedimento que grava o recorde de um jogador. Recebe como parâmetro duas variáveis, uma do tipo caractere e outra o tipo inteiro. O algoritmo faz a comparação entre a pontuação atingida pelo jogador e as pontuações que estão armazenadas em um arquivo. Este procedimento chama um outro procedimento para ler os recordes e efetuar a comparação.

```
void gravarRecordes(char *nome, int pontos)
{
int i, k;
Jogador player[5];
FILE *record;

   lerRecordes(player);

   for(i=0; i<5; i++) {
      if(pontos>player[i].pontos){
         for(k=4; k>i; k--){
            strcpy(player[k].nome, player[k-1].nome);
            player[k].pontos=player[k-1].pontos;
         }
         strcpy(player[i].nome, nome);
         player[i].pontos=pontos;
         break;
      }
   }
```

```
    record=fopen("Recordes.txt", "wb");
    fwrite(&player,sizeof(Jogador),5,record);
    fclose(record);
}
```

- Procedimento que mostra os recordes na tela. Recebe como parâmetro uma estrutura, que contém os nomes e os pontos dos jogadores. O algoritmo mostra a colocação, o nome dos jogadores e os pontos feitos por cada.

```
void escreveRecordes(Jogador *player)
{
int i;

    printf("\tRECORDES:\n posicao/nome/pontuacao\n\n");

    for(i=0; i<5; i++){
        printf("%d- %s %d\n",i+1,player[i].nome, player[i].pontos);
    }
}
```

- Procedimento principal para o funcionamento do jogo. Recebe por parâmetro 4 variáveis apontadoras do tipo inteiro, uma com o número da peça, uma com a posição em que esta se encontra, uma com a pontuação do jogador e uma com o nível em que este se encontra. Recebe ainda uma variável do tipo caractere com o nome do jogador. O procedimento ainda declara como variável local do tipo caractere . O algoritmo possui uma estrutura de repetição principal com a condição de que se uma função retorna algum número diferente do número 1, o programa continua. Se não ele para a repetição e dá a mensagem de fim de jogo. O primeiro comando dentro desta estrutura compara a posição X,. Se esta for igual a 0, o procedimento chama outro procedimento para gerar nova peça apenas de houve colisão, e as variáveis do tipo apontadoras recebem o procedimento para o sorteio da peça que irá cair e outra variável deste mesmo tipo recebe um procedimento para o sorteio da rotação desta peça. Caso contrário, ou seja, se a variável comparada for diferente de 0, significa pausa do jogo. Depois desta estrutura de comparação o procedimento principal chama outro para as coordenadas das peças. Depois deste, o próximo comando é uma estrutura de comparação com um loop infinito, ou seja, a condição de parada desta estrutura é infinita, porém ela só irá para quando encontrar um comando do tipo else e break, para sair do loop. Dentro desta, é chamado um procedimento para zerar a matriz pra comparação, depois é realizada as comparações para identificar se o jogador quer pausar o jogo, rotacionar a peça, mover para a direita ou para a esquerda. Quando o jogador aperta a tecla p, a variável tecla, recebe 'p', com isto o jogo pausa. Quando o jogador aperta a tecla w, a variável tecla recebe 'w', com isto a variável apontadora recebe o procedimento para mudar a posição da peça. Depois é chamado outros dois procedimentos para armazenar a peça e ver as coordenadas desta. Se o jogador aperta a tecla d, a variável tecla recebe 'd', e com isto chama um procedimento para deslocar a peça para a diretira e a variável que possui a posição Y é incrementada em 1. Se o jogador aperta a tecla a, a variável tecla recebe 'a', e com isto chama um procedimento para deslocar a peça para a esquerda e a variável que possui a posição Y é decrementada em 1. Ainda dentro da estrutura de repetição principal, o

próximo comando é a chamada de um procedimento para armazenar a peça, depois efetua uma comparação para verificar a colisão em baixo, na matriz, comprando o valor de retorno de um procedimento. Se este retornar 0, a posição X e incrementada em 1. Se não é efetuada a chamada de outro procedimento para trocar os valores e as variáveis da posição X e da posição Y recebem 0. Depois desta comparação, o compilador encontrará um break pra parar a estrutura de repetição principal e gerar uma nova peça. Depois desta parada, o procedimento principal chama outro procedimento para imprimir a matriz principal na tela, e depois deste é chamado outro que aumenta o nível conforme a pontuação do jogador. Depois destes comandos, é chamado dois procedimentos para pagar as peças, conforme as regras do Columns e outro para mudar o nível. Por último aparece a pontuação feia pelo jogador, e o procedimento principal chama outro para ver ou armazenar os recordes.

```
void jogo(int *nroPeca, int *posicao, char *nome, int *pontos, int *nivel)
{

   char tecla;

   do {
      if(posX==0) {
         sortearCor();
         *nroPeca=sortearPeca();
         *posicao=sortearRotacao();
      }
      coordenadas(*nroPeca, *posicao);
      for(;;) {
         zerarTabuleiro();
         if(kbhit()){

               tecla = getch();

               if(tecla=='p')
                  return;

               if(tecla=='w') {
                  *posicao=mudarPosicao(*nroPeca, *posicao);
                  armazenarPeca(*nroPeca, *posicao);
                  coordenadas(*nroPeca, *posicao);
               }
               if(tecla=='d') { // move para a direita
```

```
                    if(colisaoDireita()==0)
                posY++;
            }
            else if(tecla=='a') { // move para a esquerda
               if(colisaoEsquerda()==0)
                posY--;
            }

        }
        armazenarPeca(*nroPeca, *posicao);
        if(colisaoBaixo()==0)
           posX++;
        else {
           transferirJogo();
           posX=0;
           posY=0;
           break;
        }

        imprimeMatriz(*pontos, *nivel);
        Sleep(pow(0.9,*nivel-1)*VINICIAL);
        system("cls");
      }
      apagarPecas(pontos);
      mudarNivel(pontos, nivel);

    } while(fimJogo()!=1);

    printf("     TETRIS + COLLUMS\n\n");
    printf("\n\n\nFim de jogo!\n\n");
    printf("Sua pontuacao final foi de %d pontos\n", *pontos);
    printf("Voce pode conferir os recordes no menu.\n");
    gravarRecordes(nome, *pontos);
    getch();
    system("cls");
}
```

- Função do tipo inteira. Não recebe nada por parâmetro. O algoritmo implementado declara três variáveis do tipo inteiro, uma do tipo caractere, duas estruturas, na qual a estrutura player é um vetor de estruturas com 5 posições. A função possui uma estrutura de repetição principal, na qual continua executando o comando enquanto a variável opc for diferente do caractere 5. Dentro desta estrutura de repetição, há os possíveis casos que um jogador pode querer. O primeiro seria inicializar novo jogo chamando os procedimentos principais para iniciar. O segundo para voltar ao jogo depois de pausar, chamando o procedimento principal e verificando se há movimentação. O terceiro para mostrar os recordes para o jogador.

```
int main()
{
int nroPeca, posicao, nivel;
char opc;

Jogador j, player[5];
   do {

      imprimeMenu();

      opc=getch();

      system("cls");

      switch(opc) {
         case '1':
            srand(time(NULL));
            printf("Entre com seu nome:\n");
            fflush(stdin);
            scanf("%30[^\n]", &j.nome);
            system("cls");
            inicializaTabuleiro(&j.pontos, &nivel);
            jogo(&nroPeca, &posicao, j.nome, &j.pontos, &nivel);
            break;
         case '2':
            if(posX!=0) // não há movimentação
               jogo(&nroPeca, &posicao, j.nome, &j.pontos, &nivel);
            else {
               printf("\tNao ha jogo em execucao para continuar.\n");
```

```
                printf("Pressione uma tecla para voltar ao menu.\n");
                getch();
                system("cls");
            }
            break;
        case '3':
            lerRecordes(player);
            escreveRecordes(player);
            getch();
            system("cls");
            break;
    }
  } while(opc!='5');
  return 0;
}
```

CAPÍTULO 5 – Considerações Finais

Através do desenvolvimento do jogo Tetris+Columns como trabalho final, desenvolveu-se os conhecimentos vistos durante todo o semestre do curso Bacharelado em Sistemas de Informação, como o uso da linguagem C e da lógica, da implementação de algoritmos, e de reconhecer e modelar sistemas.

Aprender a trabalhar em equipe e saber lidar com situações que envolvem pressão foram uns dos desafios do desenvolvimento do projeto, os quais aproximam os alunos da realidade do mercado de trabalho e do dia-dia de um profissional da área.

Referências Bibliográficas

- *http://www.dreamincode.net/forums/topic/106575-looking-for-tetris-source-code/*
- *Arquivos disponibilizados na matéria Fundamentos de Programação 1*