Documentação - Trabalho Prático
“Jogo de Damas”

Universidade Federal de Minas Gerais
Programação e Desenvolvimento de Software II

Grupo: Brener Chaves Assis
Camila Viana Santos
Daniel Gonçalves Cabral Polido Lopes
Paula Teodoro de Rezende Lara
Vitor Rodrigues de Oliveira

1. INTRODUÇÃO

Em sala de aula foi proposto um trabalho prático cujo objetivo era elaborar um projeto de software de pequeno/médio porte com base nos conceitos e técnicas vistos em aula. O programa foi baseado na linguagem C++ e na biblioteca Allegro que auxilia no desenvolvimento de Video games.
O objetivo principal foi o desenvolvimento de um jogo de tabuleiro baseado em Damas. O jogo pratica-se entre dois jogadores, em um tabuleiro quadrado tamanho 8x8 (64 casas), alternadamente claras e escuras, dispondo de 12 peças brancas e 12 pretas. O objetivo do jogo é capturar ou imobilizar as peças do adversário. O jogador que conseguir capturar todas as peças do inimigo ganha a partida.
No início da partida, as peças são colocadas no tabuleiro sobre as casas escuras, da seguinte forma: nas três primeiras filas horizontais, as peças brancas; e, nas três últimas, as peças pretas. A peça movimenta-se em diagonal, sobre as casas escuras, para a frente, e uma casa de cada vez. A peça que atingir a oitava casa adversária, parando ali, será promovida a "dama", peça de movimentos mais amplos que a simples peça. A dama pode mover-se para trás e para frente em diagonal uma casa de cada vez, diferente das outras peças, que movimentam-se apenas para frente em diagonal.

2. IMPLEMENTAÇÃO (*DamasBiblioteca.h*)

2.1. BIBLIOTECAS

```
1  #ifndef DAMAS_H_
2  #define DAMAS_H_
3
4  #include <string>
5  #include <iostream>
6  #include <vector>
```

A biblioteca “iostream” implementa a capacidade de input/output.
A biblioteca “string” possibilita a utilização de funções para manipulação.
A biblioteca “vector” possibilita a manipulação de arranjo.

2.2. CLASSES

- Tabuleiro

```
8   class Tabuleiro {
9   public:
10          int Tabuleiroinicial[8][8];
11          void setTabuleiroInicial();
12  };
```

Linha 10: Define o tabuleiro como sendo uma matriz de inteiros de tamanho 8x8.

Linha 11: Método que inicializa o tabuleiro.

OBS: Os métodos estão declarados dentro do modo de visibilidade “public”, portanto podem ser acessadas fora do objeto onde estiver definido.

- Jogador

```
15  class Jogador {
16  private:
17          std::string Nome;
18          int Npecas;
19  public:
20          Jogador();
21          Jogador(std::string _Nome, int _Npecas);
22          void setname(std::string nome);
23          std::string getname();
24          void decreaseNpecas();
25          int getNpecas();
26  };
```

Linha 17: Define uma string que irá alocar o nome do jogador.
Linha 18: Define uma variável inteira para alocar o número de peças de cada jogador.
OBS: As linhas acima estão no “private” e só pode ser acessada exclusivamente por membros desta classe.
Linha 20: Método que inicializa o jogador.
Linha 21: Método que passa uma string e um inteiro como referência para que o construtor atribua os valores aos objetos.
Linha 22 a 25: Outros métodos do jogador.

- PecaNormal

```
38  class PecaNormal : public Peca {
39  private:
40          Jogador jogador;
41          int x;
42          int y;
43          int PosTab[2];
44  public:
45          PecaNormal();
46          PecaNormal(Jogador _jogador, int _x, int _y, int _PosTab[2]);
47          void movimento(Tabuleiro *tab, bool turn, int *opcoes, int *targets, int x, int y) override;
48          void opcoesMovimento(Tabuleiro tab, int opcoes[2][2], int targets[2][2], int PosTabX, int PosTabY) override;
49          int getpositionX() override;
50          int getpositionY() override;
51          int getPosTabX() override;
52          int getPosTabY() override;
53  };
```

Linha 40 a 43: define os atributos da classe PecaNormal.
Linha 45: método que inicializa uma peça normal
Linha 46: método que designa os atributos da peça.
Linha 47 a 52: métodos responsáveis por avaliar e realizar os possíveis movimento da peça.

- PecaDama

```
57  class PecaDama : public Peca {
58  private:
59          Jogador jogador;
60          int x;
61          int y;
62          int PosTab[2];
63  public:
64          PecaDama(Jogador _jogador, int _x, int _y, int _PosTab[2]);
65          void movimento(Tabuleiro *tab, bool turn, int *opcoes, int *targets, int x, int y) override;
66          void opcoesMovimento(Tabuleiro tab, int opcoes[2][2], int targets[2][2], int PosTabX, int PosTabY) override;
67          int getpositionX() override;
68          int getpositionY() override;
69          int getPosTabX() override;
70          int getPosTabY() override;
71  };
```

Linha 40 a 43: define os atributos da classe PecaDama.
Linha 45: método que inicializa uma peça dama.
Linha 46: método que designa os atributos da peça.
Linha 47 a 52: métodos responsáveis por avaliar e realizar os possíveis movimento da peça.

- Peca

```
28  class Peca {
29  public:
30          virtual void movimento(Tabuleiro *tab, bool turn, int *opcoes, int *targets, int x, int y);
31          virtual void opcoesMovimento(Tabuleiro tab, int opcoes[2][2], int targets[2][2], int PosTabX, int PosTabY);
32          virtual int getpositionX();
33          virtual int getpositionY();
34          virtual int getPosTabX();
35          virtual int getPosTabY();
36  };
```

A peça é uma herança da PecaNormal e da PecaDama.

2.3. OUTROS MÉTODOS

- TransformandoDama

```
278  void TransformandoDama(Peca **Vetorpeca,int pos)
279  {
280      int _PosTab[2];
281      _PosTab[0] = Vetorpeca[pos]->getPosTabX();
282      _PosTab[1] = Vetorpeca[pos]->getPosTabY();
283      PecaDama* Dama = new PecaDama(Vetorpeca[pos]->getjogador(), Vetorpeca[pos]->getpositionX(), Vetorpeca[pos]->getpositionY(), _PosTab[2]);
284      delete Vetorpeca[pos];
285      Vetorpeca[pos] = Dama;
286  };
```

3. IMPLEMENTAÇÃO (*DamasBiblioteca.cpp*)

3.1. BIBLIOTECAS

```
1   # include  " DamasBiblioteca.h "
2   # include  < iostream >
3   # include  < string >
4   # include  " allegro5 / allegro.h "
5   # include  " allegro5 / allegro_image.h "
6   # include  " allegro5 / allegro_primitives.h "
7   # include  " allegro5 / allegro_font.h "
8   # include  " allegro5 / allegro_ttf.h "
9   # include  " allegro5 / allegro_audio.h "
10  # include  " allegro5 / allegro_acodec.h "
11  # include  < vector >
```

A biblioteca “DamasBiblioteca.h” possui a declaração de todas as classes utilizadas ao longo do código.
A biblioteca “iostream” implementa a capacidade de input/output.
A biblioteca “string” possibilita a utilização de funções para manipulação.
A biblioteca "allegro5/allegro.h" implementa as funções basicas da biblioteca allegro.
A biblioteca "allegro5/allegro_image.h" possibilita adição de formatos de imagem (e.g., .JPEG, .PNG).
A biblioteca "allegro5/allegro_primitives.h" possibilita a implementação o desenho basico das funções.
A biblioteca "allegro5/allegro_font.h" possibilita a implementaçao de fontes basicas de bitmap.
A biblioteca "allegro5/allegro_ttf.h" possibilita o carregamento de TTF fonts.
A biblioteca "allegro5/allegro_audio.h" implementa o subsistema basico de audio.
A biblioteca "allegro5/allegro_acodec.h" possibilita a implementação de audio codecs (e.g., .OGG, .WAV)
A biblioteca “vector” possibilita a manipulação de arranjo.

3.2. FUNÇÕES E CONSTRUTORES

- setTabuleiroInicial

```
15  void  Tabuleiro :: setTabuleiroInicial () {
16          int i, j;
17          para (i = 0 ; i < 8 ; i ++) {
18                  para (j = 0 ; j < 8 ; j ++) {
19                          if ((i == 0 || i == 2 ) && j% 2 == 0 ) {
20                                  this -> Tabuleiroinicial [i] [j] = 2 ;
21                          }
22                          mais {
23                                  this -> Tabuleiroinicial [i] [j] = 0 ;
24                          }
25                          if (i == 1 && j% 2 == 1 ) {
26                                  this -> Tabuleiroinicial [i] [j] = 2 ;
27                          }
28                          if ((i == 5 || i == 7 ) && j% 2 == 1 ) {
29                                  this -> Tabuleiroinicial [i] [j] = 1 ;
30                          }
31                          if (i == 6 && j% 2 == 0 ) {
32                                  this -> Tabuleiroinicial [i] [j] = 1 ;
33                          }
34                  }
35          }
36  };
```

Linha 15 a 36: Inicializa o tabuleiro, preenchendo a matriz com as posições iniciais das peças do jogador 1 e 2.

- Jogador

```
38   Jogador :: Jogador () {};
39
40   Jogador :: Jogador (std :: string _Nome, int Npecas) {
41           this -> Nome = _Nome;
42           this -> Npecas = Npecas;
43   };
```

LInha 38: Inicializa o Jogador.

Linha 40 a 43: Jogador (std :: string _Nome, int Npecas): Atribui os parâmetros passados aos parâmetros definidos na classe jogador.

```
45   void Jogador::setname(std::string nome) {
46           this->Nome = nome;
47   };
48
49   std::string Jogador::getname() {
50           return this->Nome;
51   };
```

Linha 45 a 51: Atribui as informações passadas aos jogadores.

- Npeças

```
53   void Jogador::decreaseNpecas() {
54           Npecas--;
55   }
56
57   int Jogador::getNpecas() {
58           return this->Npecas;
59   };
```

Linha 53 a 55: Indica que o jogador perdeu uma peça.
Linha 57 a 59: Atribui o novo número de peças à variável Npecas do Jogador.

- Posição e Movimento de Peças

```
62  void Peca::movimento(Tabuleiro *tab, bool turn, int *opcoes, int *targets, int x, int y) {};
63
64  void Peca::opcoesMovimento(Tabuleiro tab, int opcoes[2][2], int targets[2][2], int PosTabX, int PosTabY) {
65
66  };
67
68  int Peca::getpositionX() { return 0; };
69
70  int Peca::getpositionY() { return 0; };
71
72  int Peca::getPosTabX() { return 0; };
73
74  int Peca::getPosTabY() { return 0; };
```

Linha 62 a 74: Define as posições X e Y iniciais de cada peça.

- Atribuição de Valores

```
76  PecaNormal::PecaNormal() {};
77
78  PecaNormal::PecaNormal(Jogador _jogador, int _x, int _y, int _PosTab[2]) {
79          this->jogador = _jogador;
80          this->x = _x;
81          this->y = _y;
82  };
```

Linha 76 a 82: Atribui os valores passados às variáveis do jogador.

```
182  int PecaNormal::getpositionX() {
183          return this->x;
184  };
185
186  int PecaNormal::getpositionY() {
187          return this->y;
188  };
189
190  int PecaNormal::getPosTabX() {
191          return this->PosTab[0];
192  };
193
194  int PecaNormal::getPosTabY() {
195          return this->PosTab[1];
196  };
```

Linha 182 a 196: Retorna as posições para as peças do jogador.

- Destrutor e Final da Biblioteca.cpp

Linha 240 a 242: Destrói os espaços de memória alocados e não mais utilizados.

4. IMPLEMENTAÇÃO (*Main*)

Linha 18 a 131:

- Rotinas de Inicialização Allegro: Responsável por inicializar o jogo em si, mostrando na tela todos os bitmaps definidos do jogo, menu de ações, entre outros.

Linha 144 a 153 e 205 a 211:

- Jogadores e Variáveis: Define os jogadores e os vetores responsáveis por movimento e opções de movimento.

Linha 157 a 199:

- Criação de peças: Chama os construtores responsáveis pelas peças.

Linha 231 a 426:

- Loop Principal do Jogo: Atualiza todas as variáveis durante o tempo de execução do jogo, até um dos jogadores vencer.

Linha 430 a 442:

- Liberação de Memória: Apaga todos os espaços de memória alocados durante o jogo, que não são mais utilizados.

Linha 303:

- Fim da Main: Apresenta a opção de jogar de novo.