DOCUMENTAÇÃO JOGO TRUNFO

JOSÉ TARCÍSIO PRATA JÚNIOR
PIETRO URCINE DE ABREU
Versão 1.0
Terça, 20 de Dezembro de 2022

Sumário

Table of contents

Índice Hierárquico

Hierarquia de Classes

Esta lista de hierarquias está parcialmente ordenada (ordem alfabética):

Índice dos Componentes

Lista de Classes

Aqui estão as classes, estruturas, uniões e interfaces e suas respectivas descrições:

Índice dos Arquivos

Lista de Arquivos

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

Classes

Referência da Classe Baralho

Classe que representa um baralha. Carrega cartas de um arquivo.
`#include <Baralho.h>`

Membros Públicos

- **Baralho** (std::string arquivo, **TipoCarta** tipoCarta)
  Construtor.

- **Baralho** (**TipoCarta** tipoCarta)
  Construtor. ***Baralho*** *de jogador, inicialmente vazio.*

- void **addCarta** (**Carta** *carta)
  Adiciona carta ao baralho.

- **Carta** * **retiraCarta** ()
  Retira uma carta aleatoriamente.

- **Carta** * **retiraCarta** (unsigned int id)
  Retira uma carta com o id caso válido.

- void * **consultaCarta** (unsigned int id) const
  Consulta uma carta com o id caso válido.

- bool **validId** (unsigned int id)
  Verifica se o id da carta selecionada é válido.

- virtual **~Baralho** ()
  Destrutor.

- bool **isValid** ()
  Baralho *válido (arquivo com o baralho bem formatado).*

- **TipoCarta** **getTipoCarta** () const
  Retorna tipo de carta do baralho.

- int **size** () const
  Tamanho do baralho.

- void **shuffle** ()
  Embaralha as cartas.

Amigas

- std::ostream & **operator<<** (std::ostream &o, const **Baralho** &baralho)
  Sobrecarga do operador << para stream (escrita na saída padrão ou arquivo).

Descrição detalhada

Classe que representa um baralha. Carrega cartas de um arquivo.

Parâmetros

cartas	Lista de cartas.
tipoCarta	Tipo de carta do baralho. #param valido Se o baralho é válido.

Construtores e Destrutores

Baralho::Baralho (std::string *arquivo*, TipoCarta *tipoCarta*)

Construtor.

Parâmetros

arquivo	Arquivo com o baralho.
tipoCarta	Tipo de carta do baralho.

```
16                                                          {
17     srand (time(NULL));
18     this->valid = false;
19     this->tipoCarta = tipoCarta;
20     std::ifstream file(arquivo);
21     if(file.is_open()){
22         if (this->tipoCarta == TipoCarta::CARRO){
23             unsigned int id = 0;
24             std::string nome;
25             float potencia;
26             float cilindradas;
27             unsigned int velocidade;
28             float aceleracao;
29             unsigned int peso;
30             std::string line;
31             std::getline (file, line);
32             int count;
33             while ( std::getline (file, line) ){
34                 std::stringstream ss(line);
35                 std::string value;
36                 count = 0;
37                 id++;
38                 while (!ss.eof()) {
39                     try{
40                         std::getline(ss, value, ',');
41                         switch(count){
42                             case 0:
43                                 nome = value;
44                                 break;
45                             case 1:
46                                 potencia = std::stof(value);
47                                 break;
48                             case 2:
49                                 cilindradas = std::stof(value);
50                                 break;
51                             case 3:
52                                 velocidade = std::stoul(value);
53                                 break;
54                             case 4:
```

```
55                                  aceleracao = std::stof(value);
56                                  break;
57                              case 5:
58                                  peso = std::stoul(value);
59                                  break;
60                          }
61                          count++;
62                      } catch (const std::exception& ex) {
63                          count = 100;
64                          break;
65                      }
66                  }
67                  if (count != 6){
68                      std::cerr << "Linha " << line
69                                << " é inválida e foi ignorada." << std::endl;
70                  } else {
71                      this->cartas.push_back((Carta*) new Carro(id
72                                                              , nome
73                                                              , potencia
74                                                              , cilindradas
75                                                              , velocidade
76                                                              , aceleracao
77                                                              , peso));
78                  }
79              }
80          }
81          file.close();
82          this->valid = true;
83      } else {
84          std::cerr << "Erro ao abrir o arquivo " << arquivo << std::endl;
85      }
86  }
```

Baralho::Baralho (TipoCarta *tipoCarta*)

Construtor. **Baralho** de jogador, inicialmente vazio.

```
93                                      {
94      srand (time(NULL));
95      this->tipoCarta = tipoCarta;
96      this->valid=true;
97  }
```

virtual Baralho::~Baralho () `[inline], [virtual]`

Destrutor.

```
59 {};
```

Funções membros

void Baralho::addCarta (Carta * *carta*)

Adiciona carta ao baralho.

Parâmetros

carta	**Carta**.
tipoCarta	Tipo da **Carta**.

```
111                                     {
112     if (!this->validId(carta->getId()))
113         this->cartas.push_back(carta);
114 }
```

void * Baralho::consultaCarta (unsigned int *id*) const

Consulta uma carta com o id caso válido.

```
139                                                     {
140     int posicao = this->buscaCarta(id);
141     if (posicao != -1){
142         Carta * c = this->cartas[posicao];
143         if (this->tipoCarta == TipoCarta::CARRO)
144             return (void *) new Carro(*((Carro*) c));
145     } else
146         return (void* ) new Carta();
147 }
```

TipoCarta Baralho::getTipoCarta () const

Retorna tipo de carta do baralho.

Retorna

Tipo de **Carta**

```
103                                         {
104     return tipoCarta;
105 }
```

bool Baralho::isValid ()

Baralho válido (arquivo com o baralho bem formatado).

Retorna

Se válido.

```
99                      {
100     return this->valid;
101 }
```

Carta * Baralho::retiraCarta ()

Retira uma carta aleatoriamente.

```
116                             {
117     int idx = rand()%this->cartas.size();
118     Carta * c = this->cartas[idx];
119     this->cartas.erase(this->cartas.begin() + idx);
120     return c;
121 }
```

Carta * Baralho::retiraCarta (unsigned int *id*)

Retira uma carta com o id caso válido.

```
126                                             {
127     int posicao = this->buscaCarta(id);
128     if (posicao != -1){
129         Carta * c = this->cartas[posicao];
130         this->cartas.erase(this->cartas.begin() + posicao);
131         return c;
132     } else
133         return NULL;
134 }
```

void Baralho::shuffle ()

Embaralha as cartas.

```
88                      {
```

```
89      auto rng = std::default_random_engine {};
90      std::shuffle(std::begin(this->cartas), std::end(this->cartas), rng);
91  }
```

int Baralho::size () const

Tamanho do baralho.

Retorna

Tamanho.

```
107                         {
108     return this->cartas.size();
109 }
```

bool Baralho::validId (unsigned int *id*) [inline]

Verifica se o id da carta selecionada é válido.

Parâmetros

id	Id da carta.

Retorna

Se válido.

```
55 {return this->buscaCarta(id) != -1;}
```

Amigas e Funções Relacionadas

std::ostream & operator<< (std::ostream & *o*, const Baralho & *baralho*) [friend]

Sobrecarga do operador << para stream (escrita na saída padrão ou arquivo).

Parâmetros

ostream	Stream de saída.
baralho	baralho a ser enviada para o stream.

Retorna

Stream modificado com o conteúdo do baralho.

```
166                                                         {
167     if (baralho.getTipoCarta() == TipoCarta::CARRO){
168         for (Carta * carta : baralho.cartas)
169             o << *((Carro*)carta) << std::endl
170               << "------------------------" << std::endl;
171     }
172
173     return o;
174 }
```

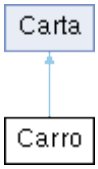
A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- **Baralho.h**
- **Baralho.cpp**

Referência da Classe Carro

`#include <Carro.h>`

Diagrama de hierarquia para Carro:



Membros Públicos

- **Carro** (unsigned int id, std::string nome, float potencia, float cilindradas, unsigned int velocidade, float aceleracao, unsigned int peso)
  Construtor.

- **Carro** (const **Carro** &carro)
  Construtor por cópia.

- virtual **~Carro** ()
  Destrutor.

- float **getAceleracao** () const
  Get para aceleração.

- void **setAceleracao** (float aceleracao)
  Set para aceleração.

- float **getCilindradas** () const
  Get para cilindradas.

- void **setCilindradas** (float cilindradas)
  Set para Cilindradas.

- unsigned int **getPeso** () const
  Get para peso.

- void **setPeso** (unsigned int peso)
  Set para Peso.

- float **getPotencia** () const
  Get para potência.

- void **setPotencia** (float potencia)
  Set para aceleração.

- unsigned int **getVelocidade** () const
  Get para aceleração.

- void **setVelocidade** (unsigned int velocidade)
  Set para velocidade.

Amigas

- std::ostream & **operator<<** (std::ostream &o, const **Carro** &carro)
  Sobrecarga do operador << para stream (escrita na saída padrão ou arquivo).

Descrição detalhada

Classe **Carro**

Parâmetros

id	Identificação da carta no baralho.
nome	Nome da carta.
potencia	Potência do carro em cavalos.
cilindradas	Cilindradas do carro em cm3.
velocidade	Velocidade máxima do carro em km/h
aceleracao	Segundos leva para ir de 0Km/h até 100Km/h.
peso	Peso do carro em Kg.

Construtores e Destrutores

Carro::Carro (unsigned int *id*, std::string *nome*, float *potencia*, float *cilindradas*, unsigned int *velocidade*, float *aceleracao*, unsigned int *peso*)

Construtor.

```
11                                : Carta(id, nome){
12     this->setPotencia(potencia);
13     this->setCilindradas(cilindradas);
14     this->setVelocidade(velocidade);
15     this->setAceleracao(aceleracao);
16     this->setPeso(peso);
17 }
```

Carro::Carro (const Carro & *carro*)

Construtor por cópia.

```
22                                    {
23     this->setId(carro.getId());
24     this->setNome(carro.getNome());
25     this->potencia = carro.getPotencia();
26     this->cilindradas = carro.getCilindradas();
27     this->velocidade = carro.getVelocidade();
28     this->aceleracao = carro.getAceleracao();
29     this->peso = carro.getPeso();
30 }
```

virtual Carro::~Carro () `[inline]`, `[virtual]`

Destrutor.

```
36 {};
```

Funções membros

float Carro::getAceleracao () const

Get para aceleração.

Retorna

Aceleração.

```
32                                  {
33     return this->aceleracao;
34 }
```

float Carro::getCilindradas () const

Get para cilindradas.

Retorna

Cilindradas.

```
43                                   {
44     return this->cilindradas;
45 }
```

unsigned int Carro::getPeso () const

Get para peso.

Retorna

Peso.

```
54                                   {
55     return this->peso;
56 }
```

float Carro::getPotencia () const

Get para potência.

Retorna

Potência.

```
62                                  {
63     return this->potencia;
64 }
```

unsigned int Carro::getVelocidade () const

Get para aceleração.

Retorna

Aceleração.

```
73                                        {
74     return this->velocidade;
```

```
75 }
```

void Carro::setAceleracao (float *aceleracao*)

Set para aceleração.

Parâmetros

aceleracao	Aceleração.

```
36                                          {
37     if (aceleracao < 0)
38         this->aceleracao = std::numeric_limits<float>::quiet_NaN();
39     else
40         this->aceleracao = aceleracao;
41 }
```

void Carro::setCilindradas (float *cilindradas*)

Set para Cilindradas.

Parâmetros

cilindradas	Cilindradas.

```
47                                            {
48     if (cilindradas < 0)
49         this->cilindradas = std::numeric_limits<float>::quiet_NaN();
50     else
51         this->cilindradas = cilindradas;
52 }
```

void Carro::setPeso (unsigned int *peso*)

Set para Peso.

Parâmetros

peso	Peso.

```
58                                        {
59     this->peso = peso;
60 }
```

void Carro::setPotencia (float *potencia*)

Set para aceleração.

Parâmetros

aceleracao	Aceleração.

```
66                                         {
67     if (potencia < 0)
68         this->potencia = std::numeric_limits<float>::quiet_NaN();
69     else
70         this->potencia = potencia;
71 }
```

void Carro::setVelocidade (unsigned int *velocidade*)

Set para velocidade.

Parâmetros

velocidade	Velocidade.

```
77                                                  {
78     this->velocidade = velocidade;
79 }
```

Amigas e Funções Relacionadas

std::ostream & operator<< (std::ostream & *o*, const Carro & *carro*) [friend]

Sobrecarga do operador << para stream (escrita na saída padrão ou arquivo).

Parâmetros

ostream	Stream de saída.
carro	**Carro** a ser enviado para o stream.

Retorna

Stream modificado com o conteúdo do carro.

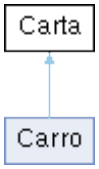
```
81                                                      {
82     o << *((Carta*)&carro) << std::endl
83       << "Potência: " << carro.getPotencia() << "cv" << std::endl
84       << "Cilindradas: " << carro.getCilindradas() << "cc" << std::endl
85       << "Velocidade: " << carro.getVelocidade() << "Km/h" << std::endl
86       << "Aceleracao: " << carro.getAceleracao() << "s" << std::endl
87       << "Peso: " << carro.getPeso() << "Kg";
88
89     return o;
90 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- **Carro.h**
- **Carro.cpp**

Referência da Classe Carta

Classe que representa uma carta.
`#include <Carta.h>`
Diagrama de hierarquia para Carta:



Membros Públicos

- **Carta** ()
  Construtor padrão.

- **Carta** (const **Carta** &other)
  Construtor por cópia.

- **Carta** (unsigned int id, std::string nome)
  Construtor.

- virtual **~Carta** ()
  Destrutor.

- unsigned int **getId** () const
  Get para id.

- void **setId** (unsigned int id)
  Set para id.

- std::string **getNome** () const
  Get para nome.

- void **setNome** (std::string nome)
  Set para nome.

Amigas

- std::ostream & **operator<<** (std::ostream &o, const **Carta** &carta)
  Sobrecarga do operador << para stream (escrita na saída padrão ou arquivo).

Descrição detalhada

Classe que representa uma carta.

Parâmetros

id	Identificação da carta no baralho.
nome	Nome da carta.

Construtores e Destrutores

Carta::Carta ()`[inline]`

Construtor padrão.

```
21 : id(0), nome(""){};
```

Carta::Carta (const Carta & *other*)

Construtor por cópia.

```
8                                {
9     this->id = other.id;
10     this->nome = other.nome;
11 }
```

Carta::Carta (unsigned int *id*, std::string *nome*)`[inline]`

Construtor.

```
29 : id(id), nome(nome){};
```

virtual Carta::~Carta ()`[inline]`, `[virtual]`

Destrutor.

```
33 {};
```

Funções membros

unsigned int Carta::getId () const

Get para id.

Retorna

Id

```
13                                  {
14     return id;
15 }
```

std::string Carta::getNome () const

Get para nome.

Retorna

```
21                                  {
22     return nome;
23 }
```

void Carta::setId (unsigned int *id*)

Set para id.

Parâmetros

id	Id da carta.

```
17                                  {
18     this->id = id;
19 }
```

void Carta::setNome (std::string *nome*)

Set para nome.

Parâmetros

nome	Nome da carta.

```
25                                   {
26     this->nome = nome;
27 }
```

Amigas e Funções Relacionadas

std::ostream & operator<< (std::ostream & *o*, const Carta & *carta*) `[friend]`

Sobrecarga do operador << para stream (escrita na saída padrão ou arquivo).

Parâmetros

ostream	Stream de saída.
carta	**Carta** a ser enviada para o stream.

Retorna

Stream modificado com o conteúdo da carta.

```
29                                                          {
30     o << "Nome: " << carta.getNome() << std::endl
31       << "Id: " << carta.getId();
32
33     return o;
34 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- **Carta.h**
- **Carta.cpp**

Arquivos

Referência do Arquivo .dep.inc

Referência do Arquivo Baralho.cpp

```
#include "Baralho.h"
#include "Carta.h"
#include "Carro.h"
#include <iostream>
#include <fstream>
#include <limits>
#include <string>
#include <sstream>
#include <algorithm>
#include <random>
```

Funções

- std::ostream & **operator<<** (std::ostream &o, const **Baralho** &baralho)

Funções

std::ostream & operator<< (std::ostream & *o*, const Baralho & *baralho*)

Parâmetros

ostream	Stream de saída.
baralho	baralho a ser enviada para o stream.

Retorna

Stream modificado com o conteúdo do baralho.

```
166                                                             {
167     if (baralho.getTipoCarta() == TipoCarta::CARRO){
168         for (Carta * carta : baralho.cartas)
169             o << *((Carro*)carta) << std::endl
170               << "------------------------" << std::endl;
171     }
172 
173     return o;
174 }
```

Referência do Arquivo Baralho.h

```
#include <vector>
#include "Carta.h"
#include "TipoCarta.h"
#include <time.h>
```

Componentes

- class **Baralho**
  Classe que representa um baralha. Carrega cartas de um arquivo.

Baralho.h

Vá para a documentação desse arquivo.

```
5 #ifndef BARALHO_H
6 #define BARALHO_H
7
8 #include <vector>
9 #include "Carta.h"
10 #include "TipoCarta.h"
11 #include <time.h>
12
20 class Baralho {
21 public:
27     Baralho(std::string arquivo, TipoCarta tipoCarta);
31     Baralho(TipoCarta tipoCarta);
37     void addCarta(Carta * carta);
41     Carta * retiraCarta();
45     Carta * retiraCarta(unsigned int id);
49     void * consultaCarta(unsigned int id) const;
55     inline bool validId(unsigned int id){return this->buscaCarta(id) != -1;}
59     inline virtual ~Baralho(){};
64     bool isValid();
69     TipoCarta getTipoCarta() const;
77     friend std::ostream& operator<<(std::ostream& o, const Baralho& baralho);
82     int size() const;
86     void shuffle();
87 private:
88     std::vector<Carta*> cartas;
89     TipoCarta tipoCarta;
90     bool valid;
95     int buscaCarta(unsigned int id) const;
96 };
97
98 #endif /* BARALHO_H */
99
```

Referência do Arquivo Carro.cpp

```
#include "Carro.h"
#include <iostream>
#include <limits>
```

Funções

- std::ostream & **operator<<** (std::ostream &o, const **Carro** &carro)

Funções

std::ostream & operator<< (std::ostream & *o*, const Carro & *carro*)

Parâmetros

ostream	Stream de saída.
carro	**Carro** a ser enviado para o stream.

Retorna

Stream modificado com o conteúdo do carro.

```
81                                                      {
82     o << *((Carta*)&carro) << std::endl
83       << "Potência: " << carro.getPotencia() << "cv" << std::endl
84       << "Cilindradas: " << carro.getCilindradas() << "cc" << std::endl
85       << "Velocidade: " << carro.getVelocidade() << "Km/h" << std::endl
86       << "Aceleracao: " << carro.getAceleracao() << "s" << std::endl
87       << "Peso: " << carro.getPeso() << "Kg";
88
89     return o;
90 }
```

Referência do Arquivo Carro.h

```
#include "Carta.h"
```

Componentes

- class **Carro**

Carro.h

Vá para a documentação desse arquivo.

```
5 #ifndef CARRO_H
6 #define CARRO_H
7
8 #include "Carta.h"
9
21 class Carro: public Carta {
22 public:
26     Carro(unsigned int id, std::string nome, float potencia, float cilindradas
27          , unsigned int velocidade, float aceleracao
28          , unsigned int peso);
32     Carro(const Carro & carro );
36     inline virtual ~Carro(){};
37
42     float getAceleracao() const;
43
48     void setAceleracao(float aceleracao);
49
54     float getCilindradas() const;
55
60     void setCilindradas(float cilindradas);
61
66     unsigned int getPeso() const;
67
72     void setPeso(unsigned int peso);
73
78     float getPotencia() const;
79
84     void setPotencia(float potencia);
85
90     unsigned int getVelocidade() const;
91
96     void setVelocidade(unsigned int velocidade);
104     friend std::ostream& operator<<(std::ostream& o, const Carro& carro);
105 private:
106     float potencia;
107     float cilindradas;
108     unsigned int velocidade;
109     float aceleracao;
110     unsigned int peso;
111 };
112
113 #endif /* CARRO_H */
114
```

Referência do Arquivo Carta.cpp

```
#include "Carta.h"
#include <iostream>
```

Funções

- std::ostream & **operator<<** (std::ostream &o, const **Carta** &carta)

Funções

std::ostream & operator<< (std::ostream & *o*, const Carta & *carta*)

Parâmetros

ostream	Stream de saída.
carta	**Carta** a ser enviada para o stream.

Retorna

Stream modificado com o conteúdo da carta.

```
29                                                          {
30     o << "Nome: " << carta.getNome() << std::endl
31       << "Id: " << carta.getId();
32
33     return o;
34 }
```

Referência do Arquivo Carta.h

```
#include <string>
```

Componentes

- class **Carta**
  Classe que representa uma carta.

Carta.h

Vá para a documentação desse arquivo.

```
5 #ifndef CARTA_H
6 #define CARTA_H
7
8 #include <string>
9
16 class Carta {
17 public:
21     inline Carta(): id(0), nome(""){};
25     Carta(const Carta& other);
29     inline Carta(unsigned int id, std::string nome): id(id), nome(nome){};
33     inline virtual ~Carta(){};
38     unsigned int getId() const;
43     void setId(unsigned int id);
48     std::string getNome() const;
53     void setNome(std::string nome);
61     friend std::ostream& operator<<(std::ostream& o, const Carta& carta);
62 private:
63     unsigned int id;
64     std::string nome;
65 };
66
67 #endif /* CARTA_H */
68
```

Referência do Arquivo main.cpp

```
#include <cstdlib>
#include <iostream>
#include "Carta.h"
#include "Carro.h"
#include "Baralho.h"
#include "TipoCarta.h"
```

Definições e Macros

- #define **DEBUG** true

Funções

- float **getValor** (**Carro** *carro, int atributo)
- int **main** (int argc, char **argv)
  Engine do jogo.

Definições e macros

#define DEBUG true

Mostra as cartas da máquina.

Funções

float getValor (Carro * *carro*, int *atributo*)

Retorna o valor de um atributo de uma **Carta Carro** pelo número.

Parâmetros

carro	Referência para uma **Carta Carro**.
atributo	Número do atributo.

Retorna

Valor do atributo.

```
136                                             {
137     float valor = -1;
138     switch(atributo){
139         case 1:
140             valor = carro->getPotencia();
141             break;
142         case 2:
143             valor = carro->getCilindradas();
144             break;
145         case 3:
146             valor = carro->getVelocidade();
147             break;
148         case 4:
149             valor = carro->getAceleracao();
150             break;
151         case 5:
152             valor = carro->getPeso();
153             break;
154     }
155     return valor;
156 }
```

int main (int *argc*, char ** *argv*)

Engine do jogo.

Parâmetros

argc	Quantidade de parâmetros passados por linha de comando.
argv	Parâmetros passados por linha de comando.

Retorna

Status do processo.

```
31                                  {
32      /* Semente do gerador de números pseudo-aleatórios. */
33      srand (time(NULL));
34
35      /* Cria baralho */
36      Baralho mesa("baralho2.txt", TipoCarta::CARRO);
37      if (!mesa.isValid()){
38          std::cerr << "Arquivo de baralho inválido!" << std::endl;
39          return -1;
40      }
41
42      /* Embaralha */
43      mesa.shuffle();
44
45      /* Cada jogador recebe 1/3 das cartas do baralho.
46       * A é o jogador humano.
47       * B é a máquina.
48       */
49      int numCartasA = mesa.size()/3;
50      int numCartasB = mesa.size()/3;
51      Baralho a(TipoCarta::CARRO);
52      Baralho b(TipoCarta::CARRO);
53      while (a.size() != numCartasA)
54          a.addCarta(mesa.retiraCarta());
55      while (b.size() != numCartasB)
56          b.addCarta(mesa.retiraCarta());
57
58      bool vezHumano = true;
59      unsigned int id;
60      int atributo;
61      float valor;
62      float maquinaValor;
63      unsigned int maquinaId;
64      Carro * lerCarta;
65      int round = 1;
66      if (DEBUG)
67          std::cout << "VOCÊ ESTÁ JOGANDO EM MODO DEBUG!" << std::endl;
68      while (a.size() != 0 && b.size() != 0){
69          std::cout << "++++++++++++\nRound " << round
70                    << "\n++++++++++++++" << std::endl;
71          /* Mostra as cartas do humano.*/
72          std::cout << "Suas cartas são:\n" << a << std::endl;
73          if (DEBUG)
74              std::cout << "As cartas da máquina são:\n" << b << std::endl;
75
76          /* Escolhas do humano. */
77          std::cout << "Escolha sua carta por id: ";
78          std::cin >> id;
79          if (a.validId(id)){
80              if (vezHumano){
81                  std::cout << "Escolha o atributo (1 a 5)" << std::endl
82                            << "1: Potência" << std::endl
83                            << "2: Cilindradas" << std::endl
84                            << "3: Velocidade" << std::endl
85                            << "4: Aceleração" << std::endl
86                            << "5: Peso" << std::endl;
87                  std::cin >> atributo;
88              } else {
89                  atributo = rand() % 5 + 1;
90              }
91              if (atributo >= 1 && atributo <= 5){
```

```
92                  lerCarta = (Carro*)a.consultaCarta(id);
93                  valor = getValor(lerCarta, atributo);
94
95                  /* Escolhas da máquina após escolhas válidas pelo humano. */
96                  lerCarta = (Carro *) b.retiraCarta();
97                  maquinaValor = getValor(lerCarta, atributo);
98                  maquinaId = lerCarta->getId();
99                  b.addCarta((Carta*) lerCarta);
100                  if (DEBUG)
101                      std::cout << "A máquina escolheu a carta de id "
102                                << maquinaId << std::endl;
103
104                  if ((atributo == 4 && valor < maquinaValor)
105                      || ((atributo != 4 && valor > maquinaValor))) {
106                      a.addCarta((Carta*) b.retiraCarta(maquinaId));
107                      std::cout << "Parabéns, você ganhou a carta\n"
108                                << *((Carro*) a.consultaCarta(maquinaId))
109                                << std::endl;
110                      vezHumano = true;
111                  } else if (valor != maquinaValor) {
112                      b.addCarta((Carta*) a.retiraCarta(id));
113                      std::cout << "Você perdeu a carta\n"
114                                << *((Carro*) b.consultaCarta(id))
115                                << std::endl;
116                      vezHumano = false;
117                  } else {
118                      std::cout << "Empate." << std::endl;
119                      vezHumano = !vezHumano;
120                  }
121                  round++;
122              } else
123                  std::cout << "Atributo inválido!" << std::endl;
124          } else
125              std::cout << "Carta inválida!" << std::endl;
126      }
127
128      if (b.size() == 0)
129          std::cout << "Você Ganhou!!!" << std::endl;
130      else
131          std::cout << "Você Perdeu." << std::endl;
132
133      return 0;
134  }
```

Referência do Arquivo TipoCarta.h

Enumerações

- enum class **TipoCarta** { **CARRO** }
  *Tipos de **Carta**.*

Enumerações

enum class TipoCarta `[strong]`

Tipos de **Carta**.

Enumeradores:

CARRO	

```
11 {
12     CARRO
13 };
```

TipoCarta.h

Vá para a documentação desse arquivo.

```
4 #ifndef TIPOCARTA_H
5 #define TIPOCARTA_H
6
10 enum class TipoCarta
11 {
12     CARRO
13 };
14
15 #endif /* TIPOCARTA_H */
16
```

Sumário

INDEX