Projeto Final

Gerado por Doxygen 1.12.0

Capítulo 1

Índice Hierárquico

1.1 Hierarquia de Classes

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

Capítulo 2

Índice dos Componentes

2.1 Lista de Classes

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

Capítulo 3

Índice dos Arquivos

3.1 Lista de Arquivos

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

Capítulo 4

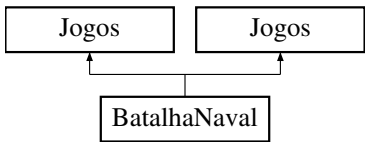
Classes

4.1 Referência da Classe BatalhaNaval

Classe que implementa o jogo de batalha naval.

`#include <BatalhaNaval.hpp>`

Diagrama de hierarquia da classe BatalhaNaval:



Membros Públicos

- BatalhaNaval ()

  Construtor da classe BatalhaNaval.
- void Jogar (Jogador &Jogador1, Jogador &Jogador2) override

  Método principal que controla o fluxo do jogo de batalha naval.

Membros Públicos herdados de Jogos

- virtual void mostrarTabuleiro ()

  Exibe o tabuleiro atual.

Membros protegidos

- void anunciarInicioPartida (Jogador &Jogador1, Jogador &Jogador2, bool &turno) override

  Anuncia o início da partida de Batalha Naval entre dois jogadores e quem irá começar.
- bool checarEmpate (int numeroJogadas, Jogador &jogador_01, Jogador &jogador_02) override
- bool checarVencedor (std::vector< std::pair< int, int > > &jogadasAtacante, std::vector< std::pair< int, int > > &barcosOponente, Jogador &vencedor, Jogador &perdedor)

  Verifica se há um vencedor com base nas jogadas do atacante e nos barcos do oponente.
- bool checarVencedor (std::vector< std::pair< int, int > > &jogadas, Jogador &vencedor, Jogador &perdedor) override
- std::pair< int, int > lerJogada (std::vector< std::vector< char > > &tabuleiroJogador)

  Lê a jogada do jogador, verificando se a posição é válida e se o tipo de dado inserido é o desejado. A inserção é feita usando (-1) nas posições para corrigir o erro proporcionado pela entrada do usuário.
- bool verificarEntrada (char tipo, int linhaInicial, int colunaInicial, int linhaFinal, int colunaFinal)

  Verifica se a entrada para posicionar o barco é válida e serve como uma função auxiliar para lerBarcos e funciona com um sistema de try-catch-throw, usando também de exceptions personalizadas para deixar os erros mais evidentes no caso de que algum deles aconteça.
- void checarPosicaoValida (std::vector< std::vector< char > > &tabuleiro)
- bool verificarSobreposicao (const std::vector< std::pair< int, int > > &barcosJogador, char tipo, int linha↩ Inicial, int colunaInicial, int linhaFinal, int colunaFinal)

  Verifica se o novo barco está sobrepondo outro barco já posicionado.
- int getTamanhoBarco (char tipo)

  Retorna o tamanho de um barco com base no tipo usando um switch-case básico.
- void lerBarcos (std::vector< std::pair< int, int > > &barcosJogador, Jogador &Jogador)

  Lê e posiciona os barcos no tabuleiro do jogador, mantendo o controle sobre a quantidade de barcos específicos já inseridos, evitando que este limite estoure. Além disso, as mensagens deixam claro como a inserção deve ser feita e o sistema loop for itera até que os 10 barcos necessários tenham sido inseridos.
- bool quantidadeBarcosDisponiveis (std::map< char, int > &countBarcos, char tipo)

  Verifica se a quantidade de barcos disponíveis para um tipo específico é suficiente e serve como auxiliar para a função de lerBarcos, impedindo que o número de barcos supere o permitido.
- bool verificarTamanhodoBarco (char tipo, int linhaInicial, int colunaInicial, int linhaFinal, int colunaFinal)

  Verifica se o tamanho do barco corresponde ao tipo informado, e é uma fução auxiliar para verificarEntrada() para garantir que o usuário não insira barcos com tamanhos além dos permitidos. O swap existe para garantir que a Final seja sempre maior que a Incial, para garantir que não hajam números negativos.
- void inserirBarcos (std::vector< std::pair< int, int > > &barcosJogador, char tipo, int linhaInicial, int coluna↩ Inicial, int linhaFinal, int colunaFinal)

  Insere os barcos no vetor de barcos do jogador. Ele implementa o mesmo sistema de swap de Inicial e Final, a inserção é feita percorrendo os limites superiores e inferiores obtidos na entrada do jogador, e introduz todas as posições ocupadas por barcos no vector com o (-1), para garantir a correspondência entre o sistema do vector e a entrada.
- void mostrarTabuleiro (const std::vector< std::vector< char > > &tabuleiroJogador)

  Exibe o tabuleiro do jogador no console, e recebe um parâmetro tabuleiro, diferentemente da classe Pai Jogos, porque há 2 tabuleiros durante a execução do jogo, e precisamos printar ambos a cada rodada, por isso, foi necessário sobrecarregar o método para usar o parâmetro em questão.
- void marcarTabuleiro (std::pair< int, int > &jogada, bool &turno, std::vector< std::vector< char > > &tabuleiroJogador, std::vector< std::pair< int, int > > &barcosJogador)

  Marca o tabuleiro com a jogada realizada, indicando acerto ('X') ou erro ('O'). A função usa da iteração no vetor de posições barcos do inimigo e a comparação com a jogada da respectiva rodada. Caso a jogada seja igual a alguma das posições, é sinalizado o acerto.

Membros protegidos herdados de Jogos

- virtual void marcarTabuleiro (std::pair< int, int > &jogada, bool &turno)

  Marca uma jogada no tabuleiro.
- virtual void limparTabuleiro ()

  Limpa o tabuleiro, preparando-o para uma nova partida.
- void anunciarTurnoJogador (Jogador &Jogador)

  Anuncia o turno do jogador atual.
- virtual bool sorteioTurno ()

  Sorteia qual jogador começa a partida.
- virtual bool checarJogadaExistente (std::vector< std::pair< int, int > > &jogadas, int linha, int coluna)

  Verifica se uma jogada já foi realizada.
- virtual bool checarPosicaoValida (int linha, int coluna)

  Verifica se uma posição no tabuleiro é válida.
- std::string gerarDivisoriaTabuleiro ()

  Gera uma divisória para separar as linhas do tabuleiro.
- virtual std::pair< int, int > lerJogada ()

Outros membros herdados

Atributos Protegidos herdados de Jogos

- std::vector< std::vector< char > > tabuleiro

4.1.1 Descrição detalhada

Classe que implementa o jogo de batalha naval.

A classe `BatalhaNaval` herda da classe base `Jogos` e é responsável por gerenciar toda a lógica do jogo de batalha naval. Isso inclui a inicialização do jogo, o posicionamento dos barcos, a leitura das jogadas dos jogadores, a verificação de vencedores e a exibição do tabuleiro.

A classe também lida com exceções específicas do jogo, como tipos de barcos inválidos e posicionamentos incorretos de barcos.

4.1.2 Construtores e Destrutores

4.1.2.1 BatalhaNaval()

```
BatalhaNaval::BatalhaNaval ()
```

Construtor da classe BatalhaNaval.

Inicializa o tabuleiro do jogo com um tamanho de 10x10, preenchendo todas as posições com o caractere '∼', que representa água. Mesmo que o tabuleiro não seja usado durante o jogo, ele é importante para estabelecer o tamanho de cada um dos 2 tabuleiros que serão usados.

4.1.3 Documentação das funções

4.1.3.1 anunciarInicioPartida()

```
void BatalhaNaval::anunciarInicioPartida (
            Jogador & Jogador1,
            Jogador & Jogador2,
            bool & turno)  [override], [protected], [virtual]
```

Anuncia o início da partida de Batalha Naval entre dois jogadores e quem irá começar.

Parâmetros

Jogador1	Referência para o primeiro jogador
Jogador2	Referência para o segundo jogador
turno	Referência para a variável que controla o turno dos jogadores

Implementa Jogos.

4.1.3.2 checarEmpate()

```
bool BatalhaNaval::checarEmpate (
            int numeroJogadas,
            Jogador & jogador_01,
            Jogador & jogador_02)  [inline], [override], [protected], [virtual]
```

Implementa Jogos.

4.1.3.3 checarPosicaoValida()

```
void BatalhaNaval::checarPosicaoValida (
            std::vector< std::vector< char > > & tabuleiro)  [protected]
```

4.1.3.4 checarVencedor() [1/2]

```
bool BatalhaNaval::checarVencedor (
            std::vector< std::pair< int, int > > & jogadas,
            Jogador & vencedor,
            Jogador & perdedor)  [inline], [override], [protected], [virtual]
```

Implementa Jogos.

4.1.3.5 checarVencedor() [2/2]

```
bool BatalhaNaval::checarVencedor (
            std::vector< std::pair< int, int > > & jogadasAtacante,
            std::vector< std::pair< int, int > > & barcosOponente,
            Jogador & vencedor,
            Jogador & perdedor)  [protected]
```

Verifica se há um vencedor com base nas jogadas do atacante e nos barcos do oponente.

Há um iterador que passa por todos os barcos do jogador inimigo e os compara com as jogadas do atacante, caso algum barco não seja encontrado, o retorno é falso e o jogo continua, caso contrário a vitória e a derrota são contabilizadas nos respectivos jogadores e o jogo termina com o print do vencedor.

Parâmetros

jogadasAtacante	Vetor de pares de inteiros representando as jogadas do atacante
barcosOponente	Vetor de pares de inteiros representando os barcos do oponente
vencedor	Referência para o jogador que venceu
perdedor	Referência para o jogador que perdeu

Retorna

true Se há um vencedor

false Se não há um vencedor

4.1.3.6 getTamanhoBarco()

```
int BatalhaNaval::getTamanhoBarco (
            char tipo)  [protected]
```

Retorna o tamanho de um barco com base no tipo usando um switch-case básico.

Parâmetros

tipo	Caractere representando o tipo de barco ('P', 'E', 'C', 'S')

Retorna

int Tamanho do barco, se for válido

-1 Se o tipo for inválido

4.1.3.7 inserirBarcos()

```
void BatalhaNaval::inserirBarcos (
            std::vector< std::pair< int, int > > & barcosJogador,
            char tipo,
            int linhaInicial,
            int colunaInicial,
            int linhaFinal,
            int colunaFinal)  [protected]
```

Insere os barcos no vetor de barcos do jogador. Ele implementa o mesmo sistema de swap de Inicial e Final, a inserção é feita percorrendo os limites superiores e inferiores obtidos na entrada do jogador, e introduz todas as posições ocupadas por barcos no vector com o (-1), para garantir a correspondência entre o sistema do vector e a entrada.

Parâmetros

barcosJogador	Vetor de pares de inteiros representando as posições dos barcos do jogador.
tipo	Caractere representando o tipo de barco ('P', 'E', 'C', 'S').
linhaInicial	Linha inicial do barco.
colunaInicial	Coluna inicial do barco.
linhaFinal	Linha final do barco.
colunaFinal	Coluna final do barco.

4.1.3.8 Jogar()

```
void BatalhaNaval::Jogar (
            Jogador & Jogador1,
            Jogador & Jogador2)  [override], [virtual]
```

Método principal que controla o fluxo do jogo de batalha naval.

Este método é responsável por gerenciar todo o ciclo de vida do jogo, desde a inicialização até a determinação do vencedor. Ele coordena as ações dos dois jogadores, alternando seus turnos e verificando se algum deles atingiu a condição de vitória.

Parâmetros

Jogador1	Referência para o primeiro jogador
Jogador2	Referência para o segundo jogador

O fluxo do jogo é o seguinte:

1. Inicializa os tabuleiros de jogadas e os vetores de barcos para ambos os jogadores.
2. Sorteia qual jogador começa a partida.
3. Cada jogador posiciona seus barcos no tabuleiro, seguindo as regras do jogo.
4. Alterna os turnos entre os jogadores, permitindo que cada um ataque o tabuleiro do oponente.
5. Verifica, após cada jogada, se todos os barcos do oponente foram afundados.
6. Declara o vencedor e registra o resultado no perfil dos jogadores.
7. Encerra o jogo.

Reimplementa Jogos.

4.1.3.9 lerBarcos()

```
void BatalhaNaval::lerBarcos (
            std::vector< std::pair< int, int > > & barcosJogador,
            Jogador & Jogador)  [protected]
```

Lê e posiciona os barcos no tabuleiro do jogador, mantendo o controle sobre a quantidade de barcos específicos já inseridos, evitando que este limite estoure. Além disso, as mensagens deixam claro como a inserção deve ser feita e o sistema loop for itera até que os 10 barcos necessários tenham sido inseridos.

A cada iteração, os dados inseridos passam por 2 sistemas de controle de qualidade, analisando se eles podem ser introduzidos no vetor de barcos do respectivo jogador e caso não possam, uma mensagem de erro aparece, o número de iterações é mantido e o loop continua.

Parâmetros

barcosJogador	Vetor de pares de inteiros representando as posições dos barcos do jogador
Jogador	Referência para o jogador

4.1.3.10 lerJogada()

```
std::pair< int, int > BatalhaNaval::lerJogada (
            std::vector< std::vector< char > > & tabuleiroJogador)  [protected]
```

Lê a jogada do jogador, verificando se a posição é válida e se o tipo de dado inserido é o desejado. A inserção é feita usando (-1) nas posições para corrigir o erro proporcionado pela entrada do usuário.

Parâmetros

tabuleiroJogador	Matriz de caracteres representando o tabuleiro do jogador

Retorna

Par de inteiros representando a jogada (linha, coluna)

4.1.3.11 marcarTabuleiro()

```
void BatalhaNaval::marcarTabuleiro (
            std::pair< int, int > & jogada,
            bool & turno,
            std::vector< std::vector< char > > & tabuleiroJogador,
            std::vector< std::pair< int, int > > & barcosJogador)  [protected]
```

Marca o tabuleiro com a jogada realizada, indicando acerto ('X') ou erro ('O'). A função usa da iteração no vetor de posições barcos do inimigo e a comparação com a jogada da respectiva rodada. Caso a jogada seja igual a alguma das posições, é sinalizado o acerto.

Parâmetros

jogada	Par de inteiros representando a jogada (linha, coluna).
turno	Referência para a variável que controla o turno dos jogadores.
tabuleiroJogador	Matriz de caracteres representando o tabuleiro do jogador.
barcosJogador	Vetor de pares de inteiros representando as posições dos barcos do jogador.

4.1.3.12 mostrarTabuleiro()

```
void BatalhaNaval::mostrarTabuleiro (
            const std::vector< std::vector< char > > & tabuleiroJogador)  [protected]
```

Exibe o tabuleiro do jogador no console, e recebe um parâmetro tabuleiro, diferentemente da classe Pai Jogos, porque há 2 tabuleiros durante a execução do jogo, e precisamos printar ambos a cada rodada, por isso, foi necessário sobrecarregar o método para usar o parâmetro em questão.

Parâmetros

tabuleiroJogador	Matriz de caracteres representando o tabuleiro do jogador.

4.1.3.13 quantidadeBarcosDisponiveis()

```
bool BatalhaNaval::quantidadeBarcosDisponiveis (
            std::map< char, int > & countBarcos,
            char tipo)  [protected]
```

Verifica se a quantidade de barcos disponíveis para um tipo específico é suficiente e serve como auxiliar para a função de lerBarcos, impedindo que o número de barcos supere o permitido.

Parâmetros

countBarcos	Mapa que conta a quantidade de barcos disponíveis por tipo
tipo	Caractere representando o tipo de barco ('P', 'E', 'C', 'S')

Retorna

true Se houver barcos disponíveis

false Se não houver barcos disponíveis

4.1.3.14 verificarEntrada()

```
bool BatalhaNaval::verificarEntrada (
            char tipo,
            int linhaInicial,
            int colunaInicial,
            int linhaFinal,
            int colunaFinal)  [protected]
```

Verifica se a entrada para posicionar o barco é válida e serve como uma função auxiliar para lerBarcos e funciona com um sistema de try-catch-throw, usando também de exceptions personalizadas para deixar os erros mais evidentes no caso de que algum deles aconteça.

Parâmetros

tipo	Caractere representando o tipo de barco
linhaInicial	Inteiro representando a linha inicial do barco
colunaInicial	Inteiro representando a coluna inicial do barco
linhaFinal	Inteiro representando a linha final do barco
colunaFinal	Inteiro representando a coluna final do barco

Retorna

true Se a entrada for válida

false Se a entrada não for válida

4.1.3.15 verificarSobreposicao()

```
bool BatalhaNaval::verificarSobreposicao (
            const std::vector< std::pair< int, int > > & barcosJogador,
            char tipo,
            int linhaInicial,
            int colunaInicial,
            int linhaFinal,
            int colunaFinal)  [protected]
```

Verifica se o novo barco está sobrepondo outro barco já posicionado.

Parâmetros

barcosJogador	Vetor de pares de inteiros representando as posições dos barcos do jogador
tipo	Caractere representando o tipo de barco ('P', 'E', 'C', 'S')
linhaInicial	Linha inicial do barco
colunaInicial	Coluna inicial do barco
linhaFinal	Linha final do barco
colunaFinal	Coluna final do barco

Retorna

true Se houver sobreposição

false Se não houver sobreposição

4.1.3.16 verificarTamanhodoBarco()

```
bool BatalhaNaval::verificarTamanhodoBarco (
            char tipo,
            int linhaInicial,
            int colunaInicial,
            int linhaFinal,
            int colunaFinal)  [protected]
```

Verifica se o tamanho do barco corresponde ao tipo informado, e é uma fução auxiliar para verificarEntrada() para garantir que o usuário não insira barcos com tamanhos além dos permitidos. O swap existe para garantir que a Final seja sempre maior que a Incial, para garantir que não hajam números negativos.

Parâmetros

tipo	Caractere representando o tipo de barco ('P', 'E', 'C', 'S')
linhaInicial	Linha inicial do barco
colunaInicial	Coluna inicial do barco
linhaFinal	Linha final do barco
colunaFinal	Coluna final do barco

Retorna

true Se o tamanho do barco for válido

false Se o tamanho do barco não for válido

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

- include/BatalhaNaval.hpp
- src/BatalhaNaval.cpp

4.2 Referência da Classe CentralDeJogos

```
#include <CentralDeJogos.hpp>
```

Membros Públicos

- CentralDeJogos ()
  Construtor da classe CentralDeJogos.
- ~CentralDeJogos ()
  Destrutor da classe CentralDeJogos.
- std::string validarEntrada ()
  Valida a entrada do usuário, garantindo que seja uma string.
- bool buscarJogador (std::string &apelido)
  Busca um jogador pelo apelido na lista de jogadores cadastrados.
- void cadastrarJogador (std::string &apelido, std::string &nome)
  Cadastra um novo jogador no sistema.
- void removerJogador (std::string &apelido)
  Remove um jogador da lista de jogadores cadastrados.
- void ordenarJogadores ()
  Ordena a lista de jogadores cadastrados pelo apelido.
- void listarJogadores ()
  Lista todos os jogadores cadastrados e suas estatísticas.
- void executarPartida ()
  Executa uma partida em um dos jogos disponíveis.

Atributos Privados

- std::vector< Jogador > jogadoresCadastrados
- JogoDaVelhaAi Ai
- JogoDaVelha velha
- Lig4 lig4
- Reversi reversi
- BatalhaNaval batalha

4.2.1 Construtores e Destrutores

4.2.1.1 CentralDeJogos()

```
CentralDeJogos::CentralDeJogos ()
```

Construtor da classe CentralDeJogos.

Este construtor é responsável por inicializar a classe CentralDeJogos, carregando os dados dos jogadores cadastrados a partir de um arquivo de texto. O arquivo deve estar localizado no caminho especificado (`data/↩DadosJogadoresCadastrados.txt`) e deve seguir um formato específico para que os dados sejam lidos corretamente.

O arquivo de dados deve conter, para cada jogador, as seguintes informações em ordem:

1. Nome do jogador (uma linha completa).
2. Apelido do jogador.
3. Número de vitórias no Jogo da Velha.
4. Número de derrotas no Jogo da Velha.

5. Número de empates no Jogo da Velha.
6. Número de vitórias no Lig4.
7. Número de derrotas no Lig4.
8. Número de empates no Lig4.
9. Número de vitórias no Reversi.
10. Número de derrotas no Reversi.
11. Número de empates no Reversi.
12. Número de vitórias no Batalha Naval.
13. Número de derrotas no Batalha Naval.
14. Número de empates no Batalha Naval.

Caso o arquivo não exista ou esteja corrompido, o construtor exibirá uma mensagem de erro e prosseguirá sem carregar os dados, iniciando o sistema com uma lista vazia de jogadores.

Observação

O arquivo de dados deve estar no formato correto. Caso contrário, o sistema pode falhar ao ler os dados e exibir uma mensagem de erro indicando que o arquivo está corrompido.

Aviso

Se o arquivo não puder ser aberto, o sistema exibirá uma mensagem de erro e não carregará os dados. Isso pode ocorrer se o arquivo não existir ou se houver problemas de permissão.

4.2.1.2 ~CentralDeJogos()

```
CentralDeJogos::~CentralDeJogos ()
```

Destrutor da classe CentralDeJogos.

Este destrutor é responsável por salvar os dados dos jogadores cadastrados em um arquivo de texto (`data/↩ DadosJogadoresCadastrados.txt`) antes de liberar a memória alocada para a classe. O arquivo é sobrescrito com as informações atualizadas dos jogadores, incluindo nome, apelido e estatísticas de cada jogo (vitórias, derrotas e empates).

O formato do arquivo é o seguinte para cada jogador:

1. Nome do jogador (uma linha completa).
2. Apelido do jogador, seguido pelas estatísticas de cada jogo na ordem:
   - Vitórias no Jogo da Velha.
   - Derrotas no Jogo da Velha.
   - Empates no Jogo da Velha.
   - Vitórias no Lig4.
   - Derrotas no Lig4.
   - Empates no Lig4.

- Vitórias no Reversi.
- Derrotas no Reversi.
- Empates no Reversi.
- Vitórias no Batalha Naval.
- Derrotas no Batalha Naval.
- Empates no Batalha Naval.

Caso o arquivo não possa ser aberto para escrita, o destrutor exibirá uma mensagem de erro e não salvará os dados. Isso pode ocorrer se o diretório `data` não existir ou se houver problemas de permissão.

Observação

O arquivo é sobrescrito a cada execução do programa, garantindo que os dados estejam sempre atualizados.

Aviso

Se o arquivo não puder ser aberto, os dados não serão salvos, e uma mensagem de erro será exibida. Isso pode resultar na perda de informações se o problema não for corrigido.

4.2.2 Documentação das funções

4.2.2.1 buscarJogador()

```
bool CentralDeJogos::buscarJogador (
            std::string & apelido)
```

Busca um jogador pelo apelido na lista de jogadores cadastrados.

Este método percorre a lista de jogadores cadastrados e verifica se o apelido fornecido corresponde ao apelido de algum jogador na lista.

Parâmetros

apelido	O apelido do jogador a ser buscado.

Retorna

true Se o jogador for encontrado

false Se o jogador não for encontrado

4.2.2.2 cadastrarJogador()

```
void CentralDeJogos::cadastrarJogador (
            std::string & apelido,
            std::string & nome)
```

Cadastra um novo jogador no sistema.

Este método verifica se o jogador já está cadastrado (pelo apelido) e, caso não esteja, cria um novo jogador e o adiciona à lista de jogadores cadastrados.

Parâmetros

apelido	O apelido do jogador a ser cadastrado
nome	O nome do jogador a ser cadastrado

4.2.2.3 executarPartida()

```
void CentralDeJogos::executarPartida ()
```

Executa uma partida em um dos jogos disponíveis.

Este método permite que dois jogadores (ou um jogador e a inteligência artificial) disputem uma partida em um dos jogos disponíveis: Reversi (R), Lig4 (L), Jogo da Velha (V), Batalha Naval (B) ou Jogo da Velha contra a AI (A).

O método solicita o jogo escolhido e os apelidos dos jogadores. Caso o jogo escolhido seja contra a AI, o segundo jogador é automaticamente definido como "AI". Se algum jogador não for encontrado, uma mensagem de erro é exibida.

4.2.2.4 listarJogadores()

```
void CentralDeJogos::listarJogadores ()
```

Lista todos os jogadores cadastrados e suas estatísticas.

Este método ordena a lista de jogadores pelo apelido e, em seguida, exibe o apelido, o nome e as estatísticas de cada jogador para todos os jogos disponíveis (Jogo da Velha, Lig4, Reversi e Batalha Naval).

4.2.2.5 ordenarJogadores()

```
void CentralDeJogos::ordenarJogadores ()
```

Ordena a lista de jogadores cadastrados pelo apelido.

Este método utiliza a função `std::sort` para ordenar a lista de jogadores em ordem alfabética com base no apelido de cada jogador.

4.2.2.6 removerJogador()

```
void CentralDeJogos::removerJogador (
            std::string & apelido)
```

Remove um jogador da lista de jogadores cadastrados.

Este método busca um jogador pelo apelido e, caso encontrado, o remove da lista de jogadores cadastrados. Se o jogador não for encontrado, exibe uma mensagem de erro.

Parâmetros

apelido	O apelido do jogador a ser removido

4.2.2.7 validarEntrada()

```
std::string CentralDeJogos::validarEntrada ()
```

Valida a entrada do usuário, garantindo que seja uma string.

Este método solicita uma entrada do usuário e verifica se o tipo de dado fornecido é uma string. Caso a entrada seja inválida, uma mensagem de erro é exibida, e o usuário é solicitado a tentar novamente.

Retorna

std::string A entrada válida fornecida pelo usuário.

4.2.3 Atributos

4.2.3.1 Ai

```
JogoDaVelhaAi CentralDeJogos::Ai  [private]
```

4.2.3.2 batalha

```
BatalhaNaval CentralDeJogos::batalha  [private]
```

4.2.3.3 jogadoresCadastrados

```
std::vector<Jogador> CentralDeJogos::jogadoresCadastrados  [private]
```

4.2.3.4 lig4

```
Lig4 CentralDeJogos::lig4  [private]
```

4.2.3.5 reversi

```
Reversi CentralDeJogos::reversi  [private]
```

4.2.3.6 velha

```
JogoDaVelha CentralDeJogos::velha  [private]
```

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

- include/CentralDeJogos.hpp
- src/CentralDeJogos.cpp

4.3 Referência da Classe Estatisticas

`#include <Estatisticas.hpp>`

Membros Públicos

- Estatisticas ()
  Construtor padrão da classe Estatisticas.
- Estatisticas (int vitorias, int derrotas, int empates)
  Construtor parametrizado da classe Estatisticas.
- void registrarVitoria ()
  Registra uma vitória.
- void registrarDerrota ()
  Registra uma derrota.
- void registrarEmpate ()
  Registra um empate.
- int getVitorias () const
  Retorna o número de vitórias.
- int getDerrotas () const
  Retorna o número de derrotas.
- int getEmpates () const
  Retorna o número de empates.
- std::vector< char > getHistorico () const
- void mostrarEstatisticas () const
  Exibe as estatísticas formatadas.

Atributos Privados

- int vitorias
- int derrotas
- int empates

4.3.1 Construtores e Destrutores

4.3.1.1 Estatisticas() [1/2]

```
Estatisticas::Estatisticas ()
```

Construtor padrão da classe Estatisticas.

Inicializa as estatísticas com zero vitórias, zero derrotas e zero empates.

4.3.1.2 Estatisticas() [2/2]

```
Estatisticas::Estatisticas (
            int vitorias,
            int derrotas,
            int empates)
```

Construtor parametrizado da classe Estatisticas.

Inicializa as estatísticas com os valores fornecidos para vitórias, derrotas e empates.

Parâmetros

vitorias	Número inicial de vitórias
derrotas	Número inicial de derrotas
empates	Número inicial de empates.

4.3.2 Documentação das funções

4.3.2.1 getDerrotas()

```
int Estatisticas::getDerrotas () const
```

Retorna o número de derrotas.

Retorna

int Número de derrotas.

4.3.2.2 getEmpates()

```
int Estatisticas::getEmpates () const
```

Retorna o número de empates.

Retorna

int Número de empates

4.3.2.3 getHistorico()

```
std::vector< char > Estatisticas::getHistorico () const
```

4.3.2.4 getVitorias()

```
int Estatisticas::getVitorias () const
```

Retorna o número de vitórias.

Retorna

int Número de vitórias

4.3.2.5 mostrarEstatisticas()

```
void Estatisticas::mostrarEstatisticas () const
```

Exibe as estatísticas formatadas.

Mostra o número de vitórias, derrotas e empates no formato:

- V: <vitórias> D: <derrotas> E: <empates>

4.3.2.6 registrarDerrota()

```
void Estatisticas::registrarDerrota ()
```

Registra uma derrota.

Incrementa o contador de derrotas.

4.3.2.7 registrarEmpate()

```
void Estatisticas::registrarEmpate ()
```

Registra um empate.

Incrementa o contador de empates.

4.3.2.8 registrarVitoria()

```
void Estatisticas::registrarVitoria ()
```

Registra uma vitória.

Incrementa o contador de vitórias.

4.3.3 Atributos

4.3.3.1 derrotas

```
int Estatisticas::derrotas  [private]
```

4.3.3.2 empates

```
int Estatisticas::empates  [private]
```

4.3.3.3 vitorias

```
int Estatisticas::vitorias  [private]
```

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

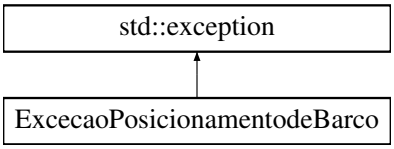
- include/Estatisticas.hpp
- src/Estatisticas.cpp

4.4 Referência da Classe ExcecaoPosicionamentodeBarco

Exceção lançada quando o posicionamento do barco é inválido.

```
#include <BatalhaNaval.hpp>
```

Diagrama de hierarquia da classe ExcecaoPosicionamentodeBarco:



Membros Públicos

- const char * what () const override throw ()

4.4.1 Descrição detalhada

Exceção lançada quando o posicionamento do barco é inválido.

Esta exceção é usada para indicar que o posicionamento de um barco no tabuleiro não segue as regras do jogo, como estar fora dos limites do tabuleiro ou não ser alinhado horizontalmente ou verticalmente.

4.4.2 Documentação das funções

4.4.2.1 what()

```
const char * ExcecaoPosicionamentodeBarco::what () const throw ( )   [inline], [override]
```

A documentação para essa classe foi gerada a partir do seguinte arquivo:

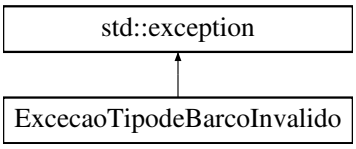
- include/BatalhaNaval.hpp

4.5 Referência da Classe ExcecaoTipodeBarcoInvalido

Exceção lançada quando um tipo de barco inválido é fornecido.

`#include <BatalhaNaval.hpp>`

Diagrama de hierarquia da classe ExcecaoTipodeBarcoInvalido:



Membros Públicos

- const char ∗ what () const override throw ()

4.5.1 Descrição detalhada

Exceção lançada quando um tipo de barco inválido é fornecido.

Esta exceção é usada para indicar que o tipo de barco fornecido pelo jogador não é válido no contexto do jogo.

4.5.2 Documentação das funções

4.5.2.1 what()

```
const char * ExcecaoTipodeBarcoInvalido::what () const throw ( )   [inline], [override]
```

A documentação para essa classe foi gerada a partir do seguinte arquivo:

- include/BatalhaNaval.hpp

4.6 Referência da Classe Jogador

`#include <Jogador.hpp>`

Membros Públicos

- Jogador (const std::string &apelido, const std::string &nome)

  Construtor padrão da classe Jogador.
- Jogador (const std::string &apelido, const std::string &nome, int vitoriasJogoDaVelha, int derrotasJogo↩ DaVelha, int empatesJogoDavelha, int vitoriasLig4, int derrotasLig4, int empatesLig4, int vitoriasReversi, int derrotasReversi, int empatesReversi, int vitoriasBatalhaNaval, int derrotasBatalhaNaval, int empates↩ BatalhaNaval)

  Construtor parametrizado da classe Jogador.
- void registrarVitoria (const std::string &nomeJogo)

  Registra uma vitória para o jogador em um jogo específico.
- void registrarDerrota (const std::string &nomeJogo)

  Registra uma derrota para o jogador em um jogo específico.
- void registrarEmpate (const std::string &nomeJogo)

  Registra um empate para o jogador em um jogo específico.
- std::string getApelido () const

  Retorna o apelido do jogador.
- std::string getNome () const

  Retorna o apelido do jogador.
- int getVitorias (std::string jogo)

  Retorna o nome do jogador.
- int getDerrotas (std::string jogo)

  Retorna o número de empates do jogador em um jogo específico.
- int getEmpates (std::string jogo)

  Retorna o número de empates do jogador em um jogo específico.
- void mostrarEstatisticas (const std::string &nomeJogo) const

  Exibe as estatísticas do jogador em um jogo específico.

Atributos Privados

- std::string apelido
- std::string nome
- std::unordered_map< std::string, Estatisticas > estatisticasPorJogo

4.6.1 Construtores e Destrutores

4.6.1.1 Jogador() [1/2]

```
Jogador::Jogador (
            const std::string & apelido,
            const std::string & nome)
```

Construtor padrão da classe Jogador.

Inicializa o jogador com um apelido e nome, e cria estatísticas zeradas para todos os jogos disponíveis.

Parâmetros

apelido	Apelido único do jogador
nome	Nome completo do jogador

4.6.1.2 Jogador() [2/2]

```
Jogador::Jogador (
            const std::string & apelido,
            const std::string & nome,
            int vitoriasJogoDaVelha,
            int derrotasJogoDaVelha,
            int empatesJogoDavelha,
            int vitoriasLig4,
            int derrotasLig4,
            int empatesLig4,
            int vitoriasReversi,
            int derrotasReversi,
            int empatesReversi,
            int vitoriasBatalhaNaval,
            int derrotasBatalhaNaval,
            int empatesBatalhaNaval)
```

Construtor parametrizado da classe Jogador.

Inicializa o jogador com um apelido, nome e estatísticas pré-definidas para todos os jogos disponíveis.

Parâmetros

apelido	Apelido único do jogador
nome	Nome completo do jogador
vitoriasJogoDaVelha	Número de vitórias no Jogo da Velha
derrotasJogoDaVelha	Número de derrotas no Jogo da Velha
empatesJogoDavelha	Número de empates no Jogo da Velha
vitoriasLig4	Número de vitórias no Lig4
derrotasLig4	Número de derrotas no Lig4
empatesLig4	Número de empates no Lig4
vitoriasReversi	Número de vitórias no Reversi
derrotasReversi	Número de derrotas no Reversi
empatesReversi	Número de empates no Reversi
vitoriasBatalhaNaval	Número de vitórias no Batalha Naval
derrotasBatalhaNaval	Número de derrotas no Batalha Naval
empatesBatalhaNaval	Número de empates no Batalha Naval

4.6.2 Documentação das funções

4.6.2.1 getApelido()

```
std::string Jogador::getApelido () const
```

Retorna o apelido do jogador.

Retorna

std::string Apelido do jogador

4.6.2.2 getDerrotas()

```
int Jogador::getDerrotas (
            std::string jogo)
```

Retorna o número de empates do jogador em um jogo específico.

Parâmetros

jogo	Nome do jogo (ex: "VELHA", "LIG4")

Retorna

int Número de empates no jogo especificado

4.6.2.3 getEmpates()

```
int Jogador::getEmpates (
            std::string jogo)
```

Retorna o número de empates do jogador em um jogo específico.

Parâmetros

jogo	Nome do jogo (ex: "VELHA", "LIG4")

Retorna

int Número de empates no jogo especificado

4.6.2.4 getNome()

```
std::string Jogador::getNome () const
```

Retorna o apelido do jogador.

Retorna

std::string Apelido do jogador

4.6.2.5 getVitorias()

```
int Jogador::getVitorias (
            std::string jogo)
```

Retorna o nome do jogador.

Retorna

std::string Nome do jogador

4.6.2.6 mostrarEstatisticas()

```
void Jogador::mostrarEstatisticas (
            const std::string & nomeJogo) const
```

Exibe as estatísticas do jogador em um jogo específico.

Parâmetros

nomeJogo	Nome do jogo (ex: "VELHA", "LIG4")

4.6.2.7 registrarDerrota()

```
void Jogador::registrarDerrota (
            const std::string & nomeJogo)
```

Registra uma derrota para o jogador em um jogo específico.

Parâmetros

nomeJogo	Nome do jogo no qual a derrota será registrada (ex: "VELHA", "LIG4")

4.6.2.8 registrarEmpate()

```
void Jogador::registrarEmpate (
            const std::string & nomeJogo)
```

Registra um empate para o jogador em um jogo específico.

Parâmetros

nomeJogo	Nome do jogo no qual o empate será registrado (ex: "VELHA", "LIG4")

4.6.2.9 registrarVitoria()

```
void Jogador::registrarVitoria (
            const std::string & nomeJogo)
```

Registra uma vitória para o jogador em um jogo específico.

Parâmetros

nomeJogo	Nome do jogo no qual a vitória será registrada (ex: "VELHA", "LIG4")

4.6.3 Atributos

4.6.3.1 apelido

```
std::string Jogador::apelido  [private]
```

4.6.3.2 estatisticasPorJogo

```
std::unordered_map<std::string, Estatisticas> Jogador::estatisticasPorJogo  [private]
```

4.6.3.3 nome

```
std::string Jogador::nome  [private]
```

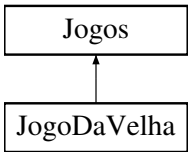
A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- include/Jogador.hpp
- src/Jogador.cpp

4.7 Referência da Classe JogoDaVelha

```
#include <JogoDaVelha.hpp>
```

Diagrama de hierarquia da classe JogoDaVelha:



Membros Públicos

- JogoDaVelha ()

  Construtor padrão da classe JogoDaVelha.
- JogoDaVelha (int tamanhoTabuleiro)

  Construtor parametrizado da classe JogoDaVelha.

Membros Públicos herdados de Jogos

- virtual void mostrarTabuleiro ()

  Exibe o tabuleiro atual.
- virtual void Jogar (Jogador &Jogador1, Jogador &Jogador2)

  Executa uma partida entre dois jogadores.

Membros protegidos

- void anunciarInicioPartida (Jogador &Jogador1, Jogador &Jogador2, bool &turno) override

  Anuncia o início da partida e define o jogador que começa.
- std::pair< int, int > lerJogada () override

  Lê a jogada do jogador atual.
- bool checarDiagonal (std::vector< std::pair< int, int > > &jogadas)

  Verifica se há um vencedor nas diagonais do tabuleiro.
- bool checarColunas (std::vector< std::pair< int, int > > &jogadas)

  Verifica se há um vencedor nas colunas do tabuleiro.
- bool checarLinhas (std::vector< std::pair< int, int > > &jogadas)

  Verifica se há um vencedor nas linhas do tabuleiro.
- bool checarVencedor (std::vector< std::pair< int, int > > &jogadas, Jogador &vencedor, Jogador &perdedor) override

  Verifica se há um vencedor no jogo.
- bool checarEmpate (int numeroJogadas, Jogador &jogador_01, Jogador &jogador_02) override

  Verifica se o jogo terminou em empate.

Membros protegidos herdados de Jogos

- virtual void marcarTabuleiro (std::pair< int, int > &jogada, bool &turno)

  Marca uma jogada no tabuleiro.
- virtual void limparTabuleiro ()

  Limpa o tabuleiro, preparando-o para uma nova partida.
- void anunciarTurnoJogador (Jogador &Jogador)

  Anuncia o turno do jogador atual.
- virtual bool sorteioTurno ()

  Sorteia qual jogador começa a partida.
- virtual bool checarJogadaExistente (std::vector< std::pair< int, int > > &jogadas, int linha, int coluna)

  Verifica se uma jogada já foi realizada.
- virtual bool checarPosicaoValida (int linha, int coluna)

  Verifica se uma posição no tabuleiro é válida.
- std::string gerarDivisoriaTabuleiro ()

  Gera uma divisória para separar as linhas do tabuleiro.

Amigos

- class JogoDaVelhaAi
- class JogoDaVelhaTests

Outros membros herdados

Atributos Protegidos herdados de Jogos

- std::vector< std::vector< char > > tabuleiro

4.7.1 Construtores e Destrutores

4.7.1.1 JogoDaVelha() [1/2]

```
JogoDaVelha::JogoDaVelha ()
```

Construtor padrão da classe JogoDaVelha.

Inicializa o tabuleiro com tamanho 3x3 e todas as posições vazias.

4.7.1.2 JogoDaVelha() [2/2]

```
JogoDaVelha::JogoDaVelha (
            int tamanhoTabuleiro)
```

Construtor parametrizado da classe JogoDaVelha.

Inicializa o tabuleiro com um tamanho personalizado e todas as posições vazias.

Parâmetros

tamanhoTabuleiro	Tamanho do tabuleiro (ex: 3 para um tabuleiro 3x3)

4.7.2 Documentação das funções

4.7.2.1 anunciarInicioPartida()

```
void JogoDaVelha::anunciarInicioPartida (
            Jogador & Jogador1,
            Jogador & Jogador2,
            bool & turno)  [override], [protected], [virtual]
```

Anuncia o início da partida e define o jogador que começa.

Exibe uma mensagem de boas-vindas e informa qual jogador começa a partida.

Parâmetros

Jogador1	Referência para o primeiro jogador
Jogador2	Referência para o segundo jogador
turno	Referência para a variável que controla o turno dos jogadores

Implementa Jogos.

4.7.2.2 checarColunas()

```
bool JogoDaVelha::checarColunas (
            std::vector< std::pair< int, int > > & jogadas)  [protected]
```

Verifica se há um vencedor nas colunas do tabuleiro.

Parâmetros

jogadas	Vetor de jogadas realizadas

Retorna

true Se houver um vencedor

false Se não houver um vencedor

4.7.2.3 checarDiagonal()

```
bool JogoDaVelha::checarDiagonal (
            std::vector< std::pair< int, int > > & jogadas)  [protected]
```

Verifica se há um vencedor nas diagonais do tabuleiro.

Parâmetros

jogadas	Vetor de jogadas realizadas

Retorna

bool True se houver um vencedor, False caso contrário

4.7.2.4 checarEmpate()

```
bool JogoDaVelha::checarEmpate (
            int numeroJogadas,
            Jogador & jogador_01,
            Jogador & jogador_02)  [override], [protected], [virtual]
```

Verifica se o jogo terminou em empate.

Parâmetros

numeroJogadas	Número total de jogadas realizadas
jogador_01	Referência para o primeiro jogador
jogador_02	Referência para o segundo jogador

Retorna

true Se o jogo terminou em empate

false Se o jogo não terminou em empate

Implementa Jogos.

4.7.2.5 checarLinhas()

```
bool JogoDaVelha::checarLinhas (
            std::vector< std::pair< int, int > > & jogadas)  [protected]
```

Verifica se há um vencedor nas linhas do tabuleiro.

Parâmetros

jogadas	Vetor de jogadas realizadas

Retorna

true Se houver um vencedor

false Se não houver um vencedor

4.7.2.6 checarVencedor()

```
bool JogoDaVelha::checarVencedor (
            std::vector< std::pair< int, int > > & jogadas,
            Jogador & vencedor,
            Jogador & perdedor)  [override], [protected], [virtual]
```

Verifica se há um vencedor no jogo.

Parâmetros

jogadas	Vetor de jogadas realizadas
vencedor	Referência para o jogador vencedor
perdedor	Referência para o jogador perdedor

Retorna

true Se houver um vencedor

false Se não houver um vencedor

Implementa Jogos.

4.7.2.7 lerJogada()

```
std::pair< int, int > JogoDaVelha::lerJogada ()  [override], [protected], [virtual]
```

Lê a jogada do jogador atual.

Solicita ao jogador que insira as coordenadas da jogada (linha e coluna) e valida a entrada.

Retorna

std::pair<int, int> Coordenadas (linha, coluna) da jogada válida

Reimplementa Jogos.

4.7.3 Documentação dos símbolos amigos e relacionados

4.7.3.1 JogoDaVelhaAi

```
friend class JogoDaVelhaAi  [friend]
```

4.7.3.2 JogoDaVelhaTests

```
friend class JogoDaVelhaTests  [friend]
```

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

- include/JogoDaVelha.hpp
- src/JogoDaVelha.cpp

4.8 Referência da Classe JogoDaVelhaAi

```
#include <JogoDaVelhaAi.hpp>
```

Membros Públicos

- JogoDaVelhaAi ()

  Construtor que inicializa o tabuleiro com células vazias.
- void Jogar (Jogador &Jogador1, Jogador &Jogador2)

  Controla o fluxo principal do jogo.

Membros privados

- bool checarVitoria (char jogador) const

  Verifica se o jogador especificado alcançou uma condição de vitória.
- bool isTabuleiroCheio () const

  Verifica se todas as posições do tabuleiro estão ocupadas.
- int minimax (bool isMaximizador, int profundidade)

  Implementa o algoritmo Minimax para avaliação de jogadas.
- int getMelhorMovimento ()

  Calcula a melhor jogada para a IA usando Minimax.
- std::pair< int, int > jogarHumano (bool turno)

  Processa e registra uma jogada do jogador humano.
- std::pair< int, int > jogarAI (bool turno)

  Calcula e executa a jogada da IA.

Atributos Privados

- int MAX_PROFUNDIDADE
- JogoDaVelha jogo
- std::vector< char > tabuleiro

4.8.1 Construtores e Destrutores

4.8.1.1 JogoDaVelhaAi()

```
JogoDaVelhaAi::JogoDaVelhaAi ()
```

Construtor que inicializa o tabuleiro com células vazias.

4.8.2 Documentação das funções

4.8.2.1 checarVitoria()

```
bool JogoDaVelhaAi::checarVitoria (
            char jogador) const  [private]
```

Verifica se o jogador especificado alcançou uma condição de vitória.

Checa todas as combinações possíveis de vitória (linhas, colunas e diagonais)

Parâmetros

player	Jogador a ser verificado (JOGADOR_X ou JOGADOR_O)

Retorna

true Se o jogador tem três símbolos consecutivos em alguma linha/coluna/diagonal

false Caso não haja vitória

4.8.2.2 getMelhorMovimento()

```
int JogoDaVelhaAi::getMelhorMovimento ()  [private]
```

Calcula a melhor jogada para a IA usando Minimax.

Utiliza uma ordem otimizada de verificação de movimentos (cantos primeiro, centro depois, bordas por último) para acelerar a busca pela jogada ideal e para evitar jogadas subótimas, uma vez que, como jogadores perfeitos sempre empatam, uma má ordenação dos movimentos pode dificultar a IA na escolha da melhor jogada.

Retorna

int Índice da melhor jogada no tabuleiro (0-8)

4.8.2.3 isTabuleiroCheio()

```
bool JogoDaVelhaAi::isTabuleiroCheio () const  [private]
```

Verifica se todas as posições do tabuleiro estão ocupadas.

Retorna

true Se não há mais espaços vazios no tabuleiro

false Se há pelo menos um espaço vazio restante

4.8.2.4 Jogar()

```
void JogoDaVelhaAi::Jogar (
            Jogador & Jogador1,
            Jogador & Jogador2)
```

Controla o fluxo principal do jogo.

Gerencia todo o ciclo de vida do jogo, incluindo:

- Configuração inicial de jogadores e dificuldade
- Alternância de turnos entre jogadores
- Verificação de condições de término (vitória/empate)
- Reinicialização do jogo ao final

Parâmetros

Jogador1	Primeiro jogador (normalmente humano)
Jogador2	Segundo jogador (normalmente IA)

Exceções

std::invalid_argument	Se entrada numérica inválida
std::out_of_range	Se valores fora dos intervalos permitidos

4.8.2.5 jogarAI()

```
std::pair< int, int > JogoDaVelhaAi::jogarAI (
            bool turno)  [private]
```

Calcula e executa a jogada da IA.

A jogada é marcada no tabuleiro da AI e no objeto JogoDaVelha, uma vez que a exibição do jogo depende do estado do tabuleiro do objeto JogoDaVelha.

Parâmetros

turno	Indica de qual jogador é o turno (não utilizado na implementação atual)

Retorna

std::pair<int, int> Coordenadas (linha, coluna) da jogada

4.8.2.6 jogarHumano()

```
std::pair< int, int > JogoDaVelhaAi::jogarHumano (
            bool turno)  [private]
```

Processa e registra uma jogada do jogador humano.

A jogada é marcada no tabuleiro da AI e no objeto JogoDaVelha, uma vez que a exibição do jogo depende do estado do tabuleiro do objeto JogoDaVelha.

Parâmetros

turno	Indica de qual jogador é o turno (não utilizado na implementação atual)

Retorna

std::pair<int, int> Coordenadas (linha, coluna) da jogada

4.8.2.7 minimax()

```
int JogoDaVelhaAi::minimax (
            bool isMaximizador,
            int profundidade)  [private]
```

Implementa o algoritmo Minimax para avaliação de jogadas.

Avalia recursivamente todas as jogadas possíveis até a profundidade máxima configurada, alternando entre jogadores de maximização (IA) e minimização (jogador humano)

Parâmetros

isMaximizador	Indica se é o turno do jogador maximizador (IA)
profundidade	Profundidade atual da recursão

Retorna

int Valor heurístico da posição (1 para vitória IA, -1 para derrota, 0 para neutro)

4.8.3 Atributos

4.8.3.1 jogo

```
JogoDaVelha JogoDaVelhaAi::jogo  [private]
```

4.8.3.2 MAX_PROFUNDIDADE

```
int JogoDaVelhaAi::MAX_PROFUNDIDADE  [private]
```

4.8.3.3 tabuleiro

```
std::vector<char> JogoDaVelhaAi::tabuleiro  [private]
```

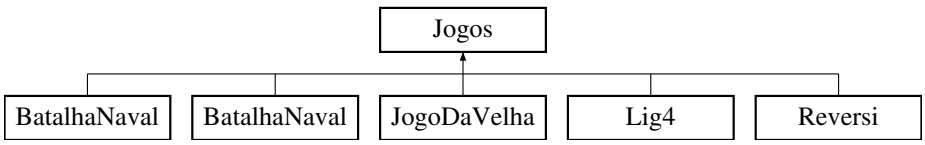
A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- include/JogoDaVelhaAi.hpp
- src/JogoDaVelhaAi.cpp

4.9 Referência da Classe Jogos

`#include <Jogos.hpp>`

Diagrama de hierarquia da classe Jogos:



Membros Públicos

- virtual void mostrarTabuleiro ()

  Exibe o tabuleiro atual.
- virtual void Jogar (Jogador &Jogador1, Jogador &Jogador2)

  Executa uma partida entre dois jogadores.

Membros protegidos

- virtual bool checarVencedor (std::vector< std::pair< int, int > > &jogadas, Jogador &vencedor, Jogador &perdedor)=0
- virtual bool checarEmpate (int numeroJogadas, Jogador &jogador_01, Jogador &jogador_02)=0
- virtual void anunciarInicioPartida (Jogador &Jogador1, Jogador &Jogador2, bool &turno)=0
- virtual void marcarTabuleiro (std::pair< int, int > &jogada, bool &turno)

  Marca uma jogada no tabuleiro.
- virtual void limparTabuleiro ()

  Limpa o tabuleiro, preparando-o para uma nova partida.
- void anunciarTurnoJogador (Jogador &Jogador)

  Anuncia o turno do jogador atual.
- virtual bool sorteioTurno ()

  Sorteia qual jogador começa a partida.
- virtual bool checarJogadaExistente (std::vector< std::pair< int, int > > &jogadas, int linha, int coluna)

  Verifica se uma jogada já foi realizada.
- virtual bool checarPosicaoValida (int linha, int coluna)

  Verifica se uma posição no tabuleiro é válida.
- std::string gerarDivisoriaTabuleiro ()

  Gera uma divisória para separar as linhas do tabuleiro.
- virtual std::pair< int, int > lerJogada ()

Atributos Protegidos

- std::vector< std::vector< char > > tabuleiro

4.9.1 Documentação das funções

4.9.1.1 anunciarInicioPartida()

```
virtual void Jogos::anunciarInicioPartida (
            Jogador & Jogador1,
            Jogador & Jogador2,
            bool & turno)  [protected], [pure virtual]
```

Implementado por BatalhaNaval, JogoDaVelha, Lig4 e Reversi.

4.9.1.2 anunciarTurnoJogador()

```
void Jogos::anunciarTurnoJogador (
            Jogador & Jogador)  [protected]
```

Anuncia o turno do jogador atual.

Parâmetros

Jogador	Referência para o jogador cujo turno será anunciado

4.9.1.3 checarEmpate()

```
virtual bool Jogos::checarEmpate (
            int numeroJogadas,
            Jogador & jogador_01,
            Jogador & jogador_02)  [protected], [pure virtual]
```

Implementado por BatalhaNaval, JogoDaVelha, Lig4 e Reversi.

4.9.1.4 checarJogadaExistente()

```
bool Jogos::checarJogadaExistente (
            std::vector< std::pair< int, int > > & jogadas,
            int linha,
            int coluna)  [protected], [virtual]
```

Verifica se uma jogada já foi realizada.

Parâmetros

jogadas	Vetor de jogadas realizadas
linha	Linha da jogada a ser verificada
coluna	Coluna da jogada a ser verificada

Retorna

true Se a jogada já foi realizada

false Se a jogada não foi realizada

4.9.1.5 checarPosicaoValida()

```
bool Jogos::checarPosicaoValida (
            int linha,
            int coluna)  [protected], [virtual]
```

Verifica se uma posição no tabuleiro é válida.

Parâmetros

linha	Linha da posição a ser verificada
coluna	Coluna da posição a ser verificada

Retorna

true Se a posição for válida

false Se a posição for inválida

4.9.1.6 checarVencedor()

```
virtual bool Jogos::checarVencedor (
            std::vector< std::pair< int, int > > & jogadas,
            Jogador & vencedor,
            Jogador & perdedor)  [protected], [pure virtual]
```

Implementado por BatalhaNaval, JogoDaVelha, Lig4 e Reversi.

4.9.1.7 gerarDivisoriaTabuleiro()

```
std::string Jogos::gerarDivisoriaTabuleiro ()  [protected]
```

Gera uma divisória para separar as linhas do tabuleiro.

Retorna

std::string Uma string contendo a divisória do tabuleiro.

4.9.1.8 Jogar()

```
void Jogos::Jogar (
            Jogador & Jogador1,
            Jogador & Jogador2)  [virtual]
```

Executa uma partida entre dois jogadores.

Gerencia o fluxo da partida, alternando turnos entre os jogadores, validando jogadas, verificando vitórias e empates, e exibindo o tabuleiro.

Parâmetros

Jogador1	Referência para o primeiro jogador
Jogador2	Referência para o segundo jogador

Reimplementado por BatalhaNaval e Reversi.

4.9.1.9 lerJogada()

```
virtual std::pair< int, int > Jogos::lerJogada ()  [inline], [protected], [virtual]
```

Reimplementado por JogoDaVelha e Lig4.

4.9.1.10 limparTabuleiro()

```
void Jogos::limparTabuleiro ()  [protected], [virtual]
```

Limpa o tabuleiro, preparando-o para uma nova partida.

Todas as posições do tabuleiro são redefinidas como vazias (' ').

Reimplementado por Reversi.

4.9.1.11 marcarTabuleiro()

```
void Jogos::marcarTabuleiro (
            std::pair< int, int > & jogada,
            bool & turno)  [protected], [virtual]
```

Marca uma jogada no tabuleiro.

Parâmetros

jogada	Coordenadas (linha, coluna) da jogada
turno	Indica o turno do jogador (true para jogador 1, false para jogador 2)

Reimplementado por Reversi.

4.9.1.12 mostrarTabuleiro()

```
void Jogos::mostrarTabuleiro ()  [virtual]
```

Exibe o tabuleiro atual.

Mostra o estado atual do tabuleiro, incluindo as marcações dos jogadores.

4.9.1.13 sorteioTurno()

```
bool Jogos::sorteioTurno ()  [protected], [virtual]
```

Sorteia qual jogador começa a partida.

Retorna

true Se o jogador 1 começar

false Se o jogador 2 começar

4.9.2 Atributos

4.9.2.1 tabuleiro

```
std::vector<std::vector<char> > Jogos::tabuleiro  [protected]
```

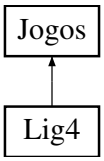
A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- include/Jogos.hpp
- src/Jogos.cpp

4.10 Referência da Classe Lig4

```
#include <Lig4.hpp>
```

Diagrama de hierarquia da classe Lig4:



Membros Públicos

- Lig4 (int tamanhoTabuleiro)
  Construtor parametrizado da classe Lig4.
- Lig4 ()
  Construtor padrão da classe Lig4.

Membros Públicos herdados de Jogos

- virtual void mostrarTabuleiro ()
  Exibe o tabuleiro atual.
- virtual void Jogar (Jogador &Jogador1, Jogador &Jogador2)
  Executa uma partida entre dois jogadores.

Membros protegidos

- void anunciarInicioPartida (Jogador &Jogador1, Jogador &Jogador2, bool &turno) override
  Anuncia o início da partida e define o jogador que começa.
- bool checarDiagonal (std::vector< std::pair< int, int > > &jogadas)
  Verifica se há um vencedor nas diagonais do tabuleiro.
- bool checarColunas (std::vector< std::pair< int, int > > &jogadas)
  Verifica se há um vencedor nas colunas do tabuleiro.
- bool checarLinhas (std::vector< std::pair< int, int > > &jogadas)
  Verifica se há um vencedor nas linhas do tabuleiro.
- bool checarVencedor (std::vector< std::pair< int, int > > &jogadas, Jogador &vencedor, Jogador &perdedor) override
  Verifica se há um vencedor no jogo.
- bool checarEmpate (int numeroJogadas, Jogador &jogador_01, Jogador &jogador_02) override
  Verifica se o jogo terminou em empate.
- std::pair< int, int > lerJogada () override
  Lê a jogada do jogador atual.

Membros protegidos herdados de Jogos

- virtual void marcarTabuleiro (std::pair< int, int > &jogada, bool &turno)

  Marca uma jogada no tabuleiro.
- virtual void limparTabuleiro ()

  Limpa o tabuleiro, preparando-o para uma nova partida.
- void anunciarTurnoJogador (Jogador &Jogador)

  Anuncia o turno do jogador atual.
- virtual bool sorteioTurno ()

  Sorteia qual jogador começa a partida.
- virtual bool checarJogadaExistente (std::vector< std::pair< int, int > > &jogadas, int linha, int coluna)

  Verifica se uma jogada já foi realizada.
- virtual bool checarPosicaoValida (int linha, int coluna)

  Verifica se uma posição no tabuleiro é válida.
- std::string gerarDivisoriaTabuleiro ()

  Gera uma divisória para separar as linhas do tabuleiro.

Amigos

- class Lig4Testes

Outros membros herdados

Atributos Protegidos herdados de Jogos

- std::vector< std::vector< char > > tabuleiro

4.10.1 Construtores e Destrutores

4.10.1.1 Lig4() [1/2]

```
Lig4::Lig4 (
            int tamanhoTabuleiro)
```

Construtor parametrizado da classe Lig4.

Inicializa o tabuleiro com um tamanho personalizado.

Parâmetros

tamanhoTabuleiro	Tamanho do tabuleiro (ex: 7 para um tabuleiro 7x6)

4.10.1.2 Lig4() [2/2]

```
Lig4::Lig4 ()
```

Construtor padrão da classe Lig4.

Inicializa o tabuleiro com tamanho padrão (7x6).

4.10.2 Documentação das funções

4.10.2.1 anunciarInicioPartida()

```
void Lig4::anunciarInicioPartida (
            Jogador & Jogador1,
            Jogador & Jogador2,
            bool & turno)  [override], [protected], [virtual]
```

Anuncia o início da partida e define o jogador que começa.

Exibe uma mensagem de boas-vindas e informa qual jogador começa a partida.

Parâmetros

Jogador1	Referência para o primeiro jogador
Jogador2	Referência para o segundo jogador
turno	Referência para a variável que controla o turno dos jogadores

Implementa Jogos.

4.10.2.2 checarColunas()

```
bool Lig4::checarColunas (
            std::vector< std::pair< int, int > > & jogadas)  [protected]
```

Verifica se há um vencedor nas colunas do tabuleiro.

Parâmetros

jogadas	Vetor de jogadas realizadas

Retorna

true se houver um vencedor

false Se não gouver um vencedor

4.10.2.3 checarDiagonal()

```
bool Lig4::checarDiagonal (
            std::vector< std::pair< int, int > > & jogadas)  [protected]
```

Verifica se há um vencedor nas diagonais do tabuleiro.

Parâmetros

jogadas	Vetor de jogadas realizadas

Retorna

true Se houver um vencedor

false Se não houver um vencedor

4.10.2.4 checarEmpate()

```
bool Lig4::checarEmpate (
            int numeroJogadas,
            Jogador & jogador_01,
            Jogador & jogador_02)  [override], [protected], [virtual]
```

Verifica se o jogo terminou em empate.

Parâmetros

numeroJogadas	Número total de jogadas realizadas
jogador_01	Referência para o primeiro jogador
jogador_02	Referência para o segundo jogador

Retorna

true Se o jogo empatou

false Se o jogo não está empatado

Implementa Jogos.

4.10.2.5 checarLinhas()

```
bool Lig4::checarLinhas (
            std::vector< std::pair< int, int > > & jogadas)  [protected]
```

Verifica se há um vencedor nas linhas do tabuleiro.

Parâmetros

jogadas	Vetor de jogadas realizadas

Retorna

true Se houver um vencedor

false Se não houver um vencedor

4.10.2.6 checarVencedor()

```
bool Lig4::checarVencedor (
            std::vector< std::pair< int, int > > & jogadas,
            Jogador & vencedor,
            Jogador & perdedor)  [override], [protected], [virtual]
```

Verifica se há um vencedor no jogo.

Parâmetros

jogadas	Vetor de jogadas realizadas
vencedor	Referência para o jogador vencedor
perdedor	Referência para o jogador perdedor

Retorna

true Se houver um vencedor

false Se não houver um vencedor

Implementa Jogos.

4.10.2.7 lerJogada()

```
std::pair< int, int > Lig4::lerJogada ()  [override], [protected], [virtual]
```

Lê a jogada do jogador atual.

Solicita ao jogador que insira a coluna onde deseja inserir a peça e valida a entrada.

Retorna

std::pair<int, int> Coordenadas (linha, coluna) da jogada válida

Reimplementa Jogos.

4.10.3 Documentação dos símbolos amigos e relacionados

4.10.3.1 Lig4Testes

```
friend class Lig4Testes  [friend]
```

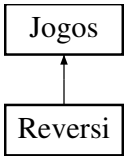
A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- include/Lig4.hpp
- src/Lig4.cpp

4.11 Referência da Classe Reversi

```
#include <Reversi.hpp>
```

Diagrama de hierarquia da classe Reversi:



Membros Públicos

- Reversi ()

  Construtor padrão da classe Reversi. Inicializa o tabuleiro com o tamanho padrão 8x8 e configura as peças iniciais.
- Reversi (int tamanhoTabuleiro)

  Construtor da classe Reversi com tamanho personalizado do tabuleiro.
- void Jogar (Jogador &Jogador1, Jogador &Jogador2) override

  Método principal para executar o jogo.

Membros Públicos herdados de Jogos

- virtual void mostrarTabuleiro ()

  Exibe o tabuleiro atual.

Membros protegidos

- void anunciarInicioPartida (Jogador &Jogador1, Jogador &Jogador2, bool &turno) override

  Anuncia o início da partida e mostra o tabuleiro inicial.
- std::pair< int, int > lerJogada (bool turno)

  Lê a jogada do jogador atual.
- void marcarTabuleiro (std::pair< int, int > &jogada, bool &turno) override

  Marca a jogada no tabuleiro e atualiza as peças, flipando as peças capturadas, caso o movimento seja válido.
- bool movimentoValido (std::pair< int, int > &jogada, char jogador, std::vector< std::pair< int, int > > &flips)

  Verifica se uma jogada é válida. É um método extremamente importante, determinando a boa execução da partida.
- bool haMovimentosDisponiveis (char Jogador)

  Verifica se há movimentos disponíveis para um jogador, ou seja, se há movimentos que capturam ao menos uma peça do adversário.
- void limparTabuleiro () override

  Limpa o tabuleiro e recoloca as peças iniciais.
- bool checarVencedor (std::vector< std::pair< int, int > > &jogadas, Jogador &vencedor, Jogador &perdedor)
- bool checarVencedor ()

  Verifica o vencedor da partida e atualiza os registros dos jogadores envolvidos na partida.
- bool checarEmpate (int numeroJogadas, Jogador &jogador_01, Jogador &jogador_02) override

Membros protegidos herdados de Jogos

- void anunciarTurnoJogador (Jogador &Jogador)

  Anuncia o turno do jogador atual.
- virtual bool sorteioTurno ()

  Sorteia qual jogador começa a partida.
- virtual bool checarJogadaExistente (std::vector< std::pair< int, int > > &jogadas, int linha, int coluna)

  Verifica se uma jogada já foi realizada.
- virtual bool checarPosicaoValida (int linha, int coluna)

  Verifica se uma posição no tabuleiro é válida.
- std::string gerarDivisoriaTabuleiro ()

  Gera uma divisória para separar as linhas do tabuleiro.
- virtual std::pair< int, int > lerJogada ()

Atributos Protegidos

- Jogador * JogadorX = nullptr
- Jogador * JogadorO = nullptr
- int ContadorTurnos = 0

Atributos Protegidos herdados de Jogos

- std::vector< std::vector< char > > tabuleiro

4.11.1 Construtores e Destrutores

4.11.1.1 Reversi() [1/2]

```
Reversi::Reversi ()
```

Construtor padrão da classe Reversi. Inicializa o tabuleiro com o tamanho padrão 8x8 e configura as peças iniciais.

4.11.1.2 Reversi() [2/2]

```
Reversi::Reversi (
            int tamanhoTabuleiro)
```

Construtor da classe Reversi com tamanho personalizado do tabuleiro.

Parâmetros

tamanhoTabuleiro	Tamanho do tabuleiro (deve ser par)

4.11.2 Documentação das funções

4.11.2.1 anunciarInicioPartida()

```
void Reversi::anunciarInicioPartida (
            Jogador & Jogador1,
            Jogador & Jogador2,
            bool & turno)  [override], [protected], [virtual]
```

Anuncia o início da partida e mostra o tabuleiro inicial.

Parâmetros

Jogador1	Referência para o primeiro jogador declarado.
Jogador2	Referência para o segundo jogador declarado.
turno	Indica qual jogador começará a partida e irá ser o 'X'. Caso turno seja "true", será o Jogador1, caso contrário, será o Jogador2.

Implementa Jogos.

4.11.2.2 checarEmpate()

```
bool Reversi::checarEmpate (
            int numeroJogadas,
            Jogador & jogador_01,
            Jogador & jogador_02)  [inline], [override], [protected], [virtual]
```

Implementa Jogos.

4.11.2.3 checarVencedor() [1/2]

```
bool Reversi::checarVencedor ()  [protected]
```

Verifica o vencedor da partida e atualiza os registros dos jogadores envolvidos na partida.

Parâmetros

movimentos	Vetor de pares de inteiros representando as jogadas do jogador
jogador_1	Referência para o Jogador1
jogador_2	Referência para o Jogador2
PrimeiroJogador	Indica qual jogador começou a partida

Retorna

true se houver um vencedor ou caso ocorra empat

false Se a partida ainda não acabou

4.11.2.4 checarVencedor() [2/2]

```
bool Reversi::checarVencedor (
            std::vector< std::pair< int, int > > & jogadas,
            Jogador & vencedor,
            Jogador & perdedor)  [inline], [protected], [virtual]
```

Implementa Jogos.

4.11.2.5 haMovimentosDisponiveis()

```
bool Reversi::haMovimentosDisponiveis (
            char jogador)  [protected]
```

Verifica se há movimentos disponíveis para um jogador, ou seja, se há movimentos que capturam ao menos uma peça do adversário.

Parâmetros

jogador	Caractere representando o jogador ('X' ou 'O')

Retorna

true Se houver movimentos disponíveis

false Se não houver movimentos disponíveis

4.11.2.6 Jogar()

```
void Reversi::Jogar (
            Jogador & Jogador1,
            Jogador & Jogador2)  [override], [virtual]
```

Método principal para executar o jogo.

Parâmetros

Jogador1	Referência para o Jogador1 (primeiro a ser declarado para jogar).
Jogador2	Referência para o Jogador2 (segundo a ser declarado para jogar).

Reimplementa Jogos.

4.11.2.7 lerJogada()

```
std::pair< int, int > Reversi::lerJogada (
            bool turno)  [protected]
```

Lê a jogada do jogador atual.

Parâmetros

turno	Herança da classe Jogos, não é utilizado no Reversi.

Retorna

Um par de inteiros representando a posição da jogada, a qual será avaliado pelos métodos: "checarPosicao↩ Valida" e "movimentoValido"

4.11.2.8 limparTabuleiro()

```
void Reversi::limparTabuleiro ()  [override], [protected], [virtual]
```

Limpa o tabuleiro e recoloca as peças iniciais.

Reimplementa Jogos.

4.11.2.9 marcarTabuleiro()

```
void Reversi::marcarTabuleiro (
            std::pair< int, int > & jogada,
            bool & turno)  [override], [protected], [virtual]
```

Marca a jogada no tabuleiro e atualiza as peças, flipando as peças capturadas, caso o movimento seja válido.

Parâmetros

jogada	Par de inteiros representando a posição da jogada
turno	Herança da classe Jogos, não é utilizado no Reversi.

Reimplementa Jogos.

4.11.2.10 movimentoValido()

```
bool Reversi::movimentoValido (
            std::pair< int, int > & jogada,
            char jogador,
            std::vector< std::pair< int, int > > & flips)  [protected]
```

Verifica se uma jogada é válida. É um método extremamente importante, determinando a boa execução da partida.

Parâmetros

jogada	Par de inteiros representando a posição da jogada
jogador	Caractere representando o jogador ('X' ou 'O')
flips	Vetor de pares de inteiros para armazenar as peças a serem viradas, ou seja, é o vetor que contém as peças capturadas

Retorna

true Se a jogada for válida

false Se a jogada for inválida

4.11.3 Atributos

4.11.3.1 ContadorTurnos

```
int Reversi::ContadorTurnos = 0  [protected]
```

4.11.3.2 JogadorO

```
Jogador* Reversi::JogadorO = nullptr  [protected]
```

4.11.3.3 JogadorX

```
Jogador* Reversi::JogadorX = nullptr  [protected]
```

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

- include/Reversi.hpp
- src/Reversi.cpp

Capítulo 5

Arquivos

5.1 Referência do Arquivo include/BatalhaNaval.hpp

```
#include "Jogos.hpp"
```

Componentes

- class BatalhaNaval

  Classe que implementa o jogo de batalha naval.
- class ExcecaoTipodeBarcoInvalido

  Exceção lançada quando um tipo de barco inválido é fornecido.
- class ExcecaoPosicionamentodeBarco

  Exceção lançada quando o posicionamento do barco é inválido.

5.2 BatalhaNaval.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef BATALHA_NAVAL
00002 #define BATALHA_NAVAL
00003
00004 #include "Jogos.hpp"
00005
00020 class BatalhaNaval : public Jogos
00021 {
00022   public:
00023     BatalhaNaval();
00024     void Jogar(Jogador &Jogador1, Jogador &Jogador2) override;
00025
00026   protected:
00027     void anunciarInicioPartida(Jogador &Jogador1, Jogador &Jogador2, bool &turno) override;
00028
00029     bool checarEmpate(int numeroJogadas, Jogador &jogador_01, Jogador &jogador_02) override { return
      false; };
00030     bool checarVencedor(std::vector<std::pair<int, int» &jogadasAtacante,
00031                         std::vector<std::pair<int, int» &barcosOponente, Jogador &vencedor, Jogador
      &perdedor);
00032     bool checarVencedor(std::vector<std::pair<int, int» &jogadas, Jogador &vencedor, Jogador
      &perdedor) override
00033     {
00034         return false;
00035     };
00036
00037     std::pair<int, int> lerJogada(std::vector<std::vector<char» &tabuleiroJogador);
```

```
00038     bool verificarEntrada(char tipo, int linhaInicial, int colunaInicial, int linhaFinal, int
      colunaFinal);
00039     void checarPosicaoValida(std::vector<std::vector<char> &tabuleiro);
00040     bool verificarSobreposicao(const std::vector<std::pair<int, int> &barcosJogador, char tipo,
00041                                int linhaInicial, int colunaInicial, int linhaFinal, int
      colunaFinal);
00042
00043     int getTamanhoBarco(char tipo);
00044     void lerBarcos(std::vector<std::pair<int, int> &barcosJogador, Jogador &Jogador);
00045     bool quantidadeBarcosDisponiveis(std::map<char, int> &countBarcos, char tipo);
00046     bool verificarTamanhodoBarco(char tipo, int linhaInicial, int colunaInicial, int linhaFinal, int
      colunaFinal);
00047     void inserirBarcos(std::vector<std::pair<int, int> &barcosJogador, char tipo, int linhaInicial,
      int colunaInicial,
00048                        int linhaFinal, int colunaFinal);
00049
00050     void mostrarTabuleiro(const std::vector<std::vector<char> &tabuleiroJogador);
00051     void marcarTabuleiro(std::pair<int, int> &jogada, bool &turno, std::vector<std::vector<char>
      &tabuleiroJogador,
00052                          std::vector<std::pair<int, int> &barcosJogador);
00053 };
00054
00062 class ExcecaoTipodeBarcoInvalido : public std::exception
00063 {
00064   public:
00065     const char *what() const throw() override { return "ERRO! Tipo de barco invalido."; }
00066 };
00067
00076 class ExcecaoPosicionamentodeBarco : public std::exception
00077 {
00078   public:
00079     const char *what() const throw() override
00080     {
00081         return "ERRO! Barcos devem ser colocados horizontalmente ou verticalmente.";
00082     }
00083 };
00084
00085 #endif
```

5.3 Referência do Arquivo include/CentralDeJogos.hpp

Definição da classe CentralDeJogos.

```
#include "BatalhaNaval.hpp"
#include "JogoDaVelha.hpp"
#include "JogoDaVelhaAi.hpp"
#include "Jogos.hpp"
#include "Lig4.hpp"
#include "Reversi.hpp"
```

Componentes

- class CentralDeJogos

5.3.1 Descrição detalhada

Definição da classe CentralDeJogos.

A classe CentralDeJogos é o núcleo do sistema de gerenciamento de jogos e jogadores, atuando como uma centralizadora de operações, conectando os jogadores aos jogos e garantindo que as estatísticas sejam atualizadas corretamente.

- Gerenciar uma lista de jogadores cadastrados, incluindo suas estatísticas em diferentes jogos.
- Controlar as instâncias dos jogos disponíveis: Jogo da Velha, Lig4, Reversi e Batalha Naval.
- Facilitar a execução de partidas, permitindo que os jogadores escolham um jogo e compitam entre si ou contra uma inteligência artificial (no caso do Jogo da Velha).
- Fornecer funcionalidades para cadastrar, remover, ordenar e listar jogadores.

5.3.1.1 Exemplo de Uso:

1. Inicialize uma instância de `CentralDeJogos`.
2. Cadastre jogadores usando o método `cadastrarJogador`.
3. Execute partidas usando o método `executarPartida`, escolhendo o jogo e os jogadores participantes.
4. Liste os jogadores e suas estatísticas usando o método `listarJogadores`.
5. Ao finalizar, os dados dos jogadores são automaticamente salvos no arquivo de texto.

Observação

A classe depende de um arquivo de texto (`data/DadosJogadoresCadastrados.txt`) para carregar e salvar os dados dos jogadores. Certifique-se de que o arquivo esteja no formato correto.

Veja também

Jogador, JogoDaVelha, Lig4, Reversi, BatalhaNaval, JogoDaVelhaAi

5.4 CentralDeJogos.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef CENTRALDEJOGOS_HPP
00002 #define CENTRALDEJOGOS_HPP
00003
00004 #include "BatalhaNaval.hpp"
00005 #include "JogoDaVelha.hpp"
00006 #include "JogoDaVelhaAi.hpp"
00007 #include "Jogos.hpp"
00008 #include "Lig4.hpp"
00009 #include "Reversi.hpp"
00010
00038 class CentralDeJogos
00039 {
00040   private:
00041     std::vector<Jogador> jogadoresCadastrados;
00042
00043     JogoDaVelhaAi Ai;
00044     JogoDaVelha velha;
00045     Lig4 lig4;
00046     Reversi reversi;
00047     BatalhaNaval batalha;
00048
00049   public:
00050     CentralDeJogos();
00051     ~CentralDeJogos();
00052
00053     std::string validarEntrada();
00054
00055     bool buscarJogador(std::string &apelido);
00056
00057     void cadastrarJogador(std::string &apelido, std::string &nome);
00058     void removerJogador(std::string &apelido);
00059     void ordenarJogadores();
00060     void listarJogadores();
00061
00062     void executarPartida();
00063 };
00064
00065 #endif
```

5.5 Referência do Arquivo include/Estatisticas.hpp

Definição da classe Estatisticas.

```
#include <vector>
```

Componentes

- class Estatisticas

5.5.1 Descrição detalhada

Definição da classe Estatisticas.

Esta classe é responsável por armazenar e gerenciar as estatísticas de um jogador em um jogo específico. Ela registra o número de vitórias, derrotas e empates, além de fornecer métodos para acessar e exibir essas informações.

A classe é utilizada em conjunto com a classe `Jogador` para armazenar as estatísticas de cada jogo (Jogo da Velha, Lig4, Reversi, Batalha Naval) individualmente.

5.5.1.1 Atributos:

- `vitorias`: Número de vitórias do jogador no jogo.
- `derrotas`: Número de derrotas do jogador no jogo.
- `empates`: Número de empates do jogador no jogo.

5.5.1.2 Funcionalidades Principais:

- Registrar vitórias, derrotas e empates.
- Acessar o número de vitórias, derrotas e empates.
- Exibir as estatísticas de forma organizada.

Veja também

Jogador

5.6 Estatisticas.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef ESTATISTICAS_HPP
00002 #define ESTATISTICAS_HPP
00003
00004 #include <vector>
00005
00029 class Estatisticas
00030 {
00031   private:
00032     int vitorias;
00033     int derrotas;
00034     int empates;
00035
00036   public:
00037     Estatisticas();
00038     Estatisticas(int vitorias, int derrotas, int empates);
00039
00040     void registrarVitoria();
00041     void registrarDerrota();
00042     void registrarEmpate();
00043
00044     int getVitorias() const;
00045     int getDerrotas() const;
00046     int getEmpates() const;
00047     std::vector<char> getHistorico() const;
00048     void mostrarEstatisticas() const;
00049 };
00050
00051 #endif
```

5.7 Referência do Arquivo include/Jogador.hpp

Definição da classe Jogador.

```
#include <map>
#include <string>
#include <unordered_map>
#include "Estatisticas.hpp"
```

Componentes

- class Jogador

5.7.1 Descrição detalhada

Definição da classe Jogador.

Esta classe representa um jogador no sistema, armazenando informações como nome, apelido e estatísticas em diferentes jogos. As estatísticas são gerenciadas por meio de um mapa que associa o nome do jogo a um objeto da classe `Estatisticas`, permitindo o registro e consulta de vitórias, derrotas e empates para cada jogo individualmente.

5.7.1.1 Atributos:

- `apelido`: Apelido único do jogador, usado para identificação.
- `nome`: Nome completo do jogador.
- `estatisticasPorJogo`: Mapa que armazena as estatísticas do jogador para cada jogo. A chave é o nome do jogo (por exemplo, "VELHA", "LIG4", "REVERSI", "BATALHANAVAL"), e o valor é um objeto da classe `Estatisticas`.

5.7.1.2 Funcionalidades Principais:

- Registro de vitórias, derrotas e empates para um jogo específico.
- Consulta das estatísticas (vitórias, derrotas e empates) para um jogo específico.
- Exibição das estatísticas de um jogo específico.

5.7.1.3 Integração com Outras Classes:

- A classe `Jogador` utiliza a classe `Estatisticas` para armazenar e gerenciar as estatísticas de cada jogo.
- É usada pela classe `CentralDeJogos` para gerenciar os jogadores cadastrados no sistema.

Veja também

Estatisticas, CentralDeJogos

5.8 Jogador.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef JOGADOR_HPP
00002 #define JOGADOR_HPP
00003
00004 #include <map>
00005 #include <string>
00006 #include <unordered_map>
00007
00008 #include "Estatisticas.hpp"
00009
00036 class Jogador
00037 {
00038   private:
00039     std::string apelido;
00040     std::string nome;
00041     std::unordered_map<std::string, Estatisticas> estatisticasPorJogo;
00042
00043   public:
00044     Jogador(const std::string &apelido, const std::string &nome);
00045     Jogador(const std::string &apelido, const std::string &nome, int vitoriasJogoDaVelha, int
      derrotasJogoDaVelha,
00046             int empatesJogoDavelha, int vitoriasLig4, int derrotasLig4, int empatesLig4, int
      vitoriasReversi,
00047             int derrotasReversi, int empatesReversi, int vitoriasBatalhaNaval, int
      derrotasBatalhaNaval,
00048             int empatesBatalhaNaval);
00049
00050     void registrarVitoria(const std::string &nomeJogo);
00051     void registrarDerrota(const std::string &nomeJogo);
00052     void registrarEmpate(const std::string &nomeJogo);
00053
00054     std::string getApelido() const;
00055     std::string getNome() const;
00056     int getVitorias(std::string jogo);
00057     int getDerrotas(std::string jogo);
00058     int getEmpates(std::string jogo);
00059
00060     void mostrarEstatisticas(const std::string &nomeJogo) const;
00061 };
00062
00063 #endif
```

5.9 Referência do Arquivo include/JogoDaVelha.hpp

Definição da classe JogoDaVelha.

```
#include "Jogos.hpp"
```

Componentes

- class JogoDaVelha

5.9.1 Descrição detalhada

Definição da classe JogoDaVelha.

Herda da classe base `Jogos` e implementa as funcionalidades específicas do jogo da velha, como leitura de jogadas, verificação de vitória e empate, e anúncio do início da partida.

5.9.1.1 Funcionalidades Principais:

- Gerenciamento de partidas de jogo da velha.
- Verificação de vitória (linhas, colunas ou diagonais completas).
- Verificação de empate (tabuleiro cheio sem vencedor).
- Anúncio do início da partida e alternância de turnos entre os jogadores.

5.9.1.2 Integração com Outras Classes:

- Herda da classe `Jogos`, que define a interface comum para todos os jogos.
- Utiliza a classe `Jogador` para representar os jogadores participantes.
- É amiga da classe `JogoDaVelhaAi`, permitindo acesso a métodos protegidos para implementação de inteligência artificial.

Veja também

Jogos, Jogador, JogoDaVelhaAi

5.10 JogoDaVelha.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef JOGODAVELHA_HPP
00002 #define JOGODAVELHA_HPP
00003
00004 #include "Jogos.hpp"
00026 class JogoDaVelha : public Jogos
00027 {
00028   public:
00029     JogoDaVelha();
00030     JogoDaVelha(int tamanhoTabuleiro);
00031
00032   protected:
00033     void anunciarInicioPartida(Jogador &Jogador1, Jogador &Jogador2, bool &turno) override;
00034
00035     std::pair<int, int> lerJogada() override;
00036
00037     bool checarDiagonal(std::vector<std::pair<int, int» &jogadas);
00038     bool checarColunas(std::vector<std::pair<int, int» &jogadas);
00039     bool checarLinhas(std::vector<std::pair<int, int» &jogadas);
00040     bool checarVencedor(std::vector<std::pair<int, int» &jogadas, Jogador &vencedor, Jogador
      &perdedor) override;
00041     bool checarEmpate(int numeroJogadas, Jogador &jogador_01, Jogador &jogador_02) override;
00042
00043     friend class JogoDaVelhaAi;
00044     friend class JogoDaVelhaTests;
00045 };
00046
00047 #endif
```

5.11 Referência do Arquivo include/JogoDaVelhaAi.hpp

```
#include "JogoDaVelha.hpp"
#include "Jogos.hpp"
#include <iostream>
#include <vector>
```

Componentes

- class JogoDaVelhaAi

Variáveis

- const int TABULEIRO_SIZE = 9
- const char VAZIO = ' '
- const char JOGADOR_X = 'X'
- const char JOGADOR_O = 'O'

5.11.1 Variáveis

5.11.1.1 JOGADOR_O

```
const char JOGADOR_O = 'O'
```

5.11.1.2 JOGADOR_X

```
const char JOGADOR_X = 'X'
```

5.11.1.3 TABULEIRO_SIZE

```
const int TABULEIRO_SIZE = 9
```

5.11.1.4 VAZIO

```
const char VAZIO = ' '
```

5.12 JogoDaVelhaAi.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef JOGODAVELHAAI_HPP
00002 #define JOGODAVELHAAI_HPP
00003
00004 #include "JogoDaVelha.hpp"
00005 #include "Jogos.hpp"
00006 #include <iostream>
00007 #include <vector>
00008
00009 const int TABULEIRO_SIZE = 9;
00010 const char VAZIO = ' ';
00011 const char JOGADOR_X = 'X';
00012 const char JOGADOR_O = 'O';
00013 class JogoDaVelhaAi
00014 {
00015   public:
00016     JogoDaVelhaAi();
00017     void Jogar(Jogador &Jogador1, Jogador &Jogador2);
00018
00019   private:
00020     int MAX_PROFUNDIDADE;
00021
00022     JogoDaVelha jogo;
00023     std::vector<char> tabuleiro;
00024
00025     bool checarVitoria(char jogador) const;
00026     bool isTabuleiroCheio() const;
00027     int minimax(bool isMaximizador, int profundidade);
00028     int getMelhorMovimento();
00029
00030     std::pair<int, int> jogarHumano(bool turno);
00031     std::pair<int, int> jogarAI(bool turno);
00032 };
00033
00034 #endif
```

5.13 Referência do Arquivo include/Jogos.hpp

Definição da classe Jogos.

```
#include <algorithm>
#include <ctime>
#include <fstream>
#include <iostream>
#include <limits>
#include <random>
#include <string>
#include <utility>
#include <vector>
#include "Jogador.hpp"
```

Componentes

- class Jogos

5.13.1 Descrição detalhada

Definição da classe Jogos.

Esta classe é uma classe base abstrata que define a interface comum para todos os jogos do sistema. Ela fornece métodos e atributos básicos para gerenciar partidas, como manipulação do tabuleiro, verificação de vitória e empate, e controle de turnos. Classes derivadas, como `JogoDaVelha`, `Lig4`, `Reversi` e `BatalhaNaval`, implementam as funcionalidades específicas de cada jogo.

5.13.1.1 Funcionalidades Principais:

- Gerenciamento de tabuleiros genéricos.
- Controle de turnos entre jogadores.
- Verificação de vitória e empate (a serem implementados pelas classes derivadas).
- Exibição do tabuleiro e anúncio de turnos.
- Sorteio de quem começa a partida.

5.13.1.2 Métodos Virtuais Puros:

- `checarVencedor`: Verifica se há um vencedor no jogo (deve ser implementado pelas classes derivadas).
- `checarEmpate`: Verifica se o jogo terminou em empate (deve ser implementado pelas classes derivadas).
- `anunciarInicioPartida`: Anuncia o início da partida e define o jogador que começa (deve ser implementado pelas classes derivadas).

5.13.1.3 Métodos Virtuais:

- `marcarTabuleiro`: Marca uma jogada no tabuleiro.
- `limparTabuleiro`: Limpa o tabuleiro para uma nova partida.
- `lerJogada`: Lê a jogada do jogador (pode ser sobrescrito pelas classes derivadas).

5.13.1.4 Métodos Públicos:

- `mostrarTabuleiro`: Exibe o tabuleiro atual.
- `Jogar`: Inicia e gerencia uma partida entre dois jogadores.

5.13.1.5 Integração com Outras Classes:

- Utiliza a classe `Jogador` para representar os participantes do jogo.
- Serve como base para classes de jogos específicos, como `JogoDaVelha`, `Lig4`, `Reversi` e `BatalhaNaval`.

Veja também

Jogador, JogoDaVelha, Lig4, Reversi, BatalhaNaval

5.14 Jogos.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef JOGOS_HPP
00002 #define JOGOS_HPP
00003
00004 #include <algorithm>
00005 #include <ctime>
00006 #include <fstream>
00007 #include <iostream>
00008 #include <limits>
00009 #include <random>
00010 #include <string>
00011 #include <utility>
00012 #include <vector>
00013
00014 #include "Jogador.hpp"
00015
00052 class Jogos
00053 {
00054   protected:
00055     std::vector<std::vector<char> tabuleiro;
00056
00057     virtual bool checarVencedor(std::vector<std::pair<int, int> &jogadas, Jogador &vencedor, Jogador
      &perdedor) = 0;
00058     virtual bool checarEmpate(int numeroJogadas, Jogador &jogador_01, Jogador &jogador_02) = 0;
00059     virtual void anunciarInicioPartida(Jogador &Jogador1, Jogador &Jogador2, bool &turno) = 0;
00060
00061     virtual void marcarTabuleiro(std::pair<int, int> &jogada, bool &turno);
00062
00063     virtual void limparTabuleiro();
00064
00065     void anunciarTurnoJogador(Jogador &Jogador);
00066
00067     virtual bool sorteioTurno();
00068     virtual bool checarJogadaExistente(std::vector<std::pair<int, int> &jogadas, int linha, int
      coluna);
00069     virtual bool checarPosicaoValida(int linha, int coluna);
00070
00071     std::string gerarDivisoriaTabuleiro();
00072
00073     virtual std::pair<int, int> lerJogada() { return {0, 0}; };
00074
00075   public:
00076     virtual void mostrarTabuleiro();
00077     virtual void Jogar(Jogador &Jogador1, Jogador &Jogador2);
00078 };
00079
00080 #endif
```

5.15 Referência do Arquivo include/Lig4.hpp

Definição da classe Lig4.

```
#include "Jogos.hpp"
```

Componentes

- class Lig4

5.15.1 Descrição detalhada

Definição da classe Lig4.

Esta classe representa o jogo Lig4 (Connect Four), um jogo de tabuleiro para dois jogadores. Ela herda da classe base `Jogos` e implementa as funcionalidades específicas do Lig4, como leitura de jogadas, verificação de vitória e empate, e anúncio do início da partida.

5.15.1.1 Funcionalidades Principais:

- Gerenciamento de partidas de Lig4.
- Verificação de vitória (linhas, colunas ou diagonais completas).
- Verificação de empate (tabuleiro cheio sem vencedor).
- Anúncio do início da partida e alternância de turnos entre os jogadores.

5.15.1.2 Métodos Sobrescritos:

- `anunciarInicioPartida`: Anuncia o início da partida e define o jogador que começa.
- `checarVencedor`: Verifica se há um vencedor no jogo.
- `checarEmpate`: Verifica se o jogo terminou em empate.
- `lerJogada`: Lê a jogada do jogador atual.

5.15.1.3 Métodos Adicionais:

- `checarDiagonal`: Verifica se há um vencedor nas diagonais do tabuleiro.
- `checarColunas`: Verifica se há um vencedor nas colunas do tabuleiro.
- `checarLinhas`: Verifica se há um vencedor nas linhas do tabuleiro.

5.15.1.4 Integração com Outras Classes:

- Herda da classe `Jogos`, que define a interface comum para todos os jogos.
- Utiliza a classe `Jogador` para representar os jogadores participantes.

Veja também

Jogos, Jogador

5.16 Lig4.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef LIG4_HPP
00002 #define LIG4_HPP
00003
00004 #include "Jogos.hpp"
00005
00037 class Lig4 : public Jogos
00038 {
00039   public:
00040     Lig4(int tamanhoTabuleiro);
00041     Lig4();
00042
00043   protected:
00044     void anunciarInicioPartida(Jogador &Jogador1, Jogador &Jogador2, bool &turno) override;
00045
00046     bool checarDiagonal(std::vector<std::pair<int, int» &jogadas);
00047     bool checarColunas(std::vector<std::pair<int, int» &jogadas);
00048     bool checarLinhas(std::vector<std::pair<int, int» &jogadas);
00049     bool checarVencedor(std::vector<std::pair<int, int» &jogadas, Jogador &vencedor, Jogador
      &perdedor) override;
00050     bool checarEmpate(int numeroJogadas, Jogador &jogador_01, Jogador &jogador_02) override;
00051
00052     std::pair<int, int> lerJogada() override;
00053
00054     friend class Lig4Testes;
00055 };
00056
00057 #endif
```

5.17 Referência do Arquivo include/Reversi.hpp

Definição da classe Reversi e seus métodos para o jogo Reversi.

```
#include "Jogos.hpp"
```

Componentes

- class Reversi

5.17.1 Descrição detalhada

Definição da classe Reversi e seus métodos para o jogo Reversi.

5.18 Reversi.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef REVERSI_HPP
00002 #define REVERSI_HPP
00003
00004 #include "Jogos.hpp"
00005 class Reversi : public Jogos
00006 {
00007   public:
00008     Reversi();
00009     Reversi(int tamanhoTabuleiro);
00010     void Jogar(Jogador &Jogador1, Jogador &Jogador2) override;
00011
00012   protected:
00013     void anunciarInicioPartida(Jogador &Jogador1, Jogador &Jogador2, bool &turno) override;
00014
00015     std::pair<int, int> lerJogada(bool turno);
00016
00017     void marcarTabuleiro(std::pair<int, int> &jogada, bool &turno) override;
00018     bool movimentoValido(std::pair<int, int> &jogada, char jogador, std::vector<std::pair<int, int»
      &flips);
00019     bool haMovimentosDisponiveis(char Jogador);
00020
00021     void limparTabuleiro() override;
00022     bool checarVencedor(std::vector<std::pair<int, int» &jogadas, Jogador &vencedor, Jogador
      &perdedor)
00023     {
00024         return false;
00025     };
00026     bool checarVencedor();
00027     bool checarEmpate(int numeroJogadas, Jogador &jogador_01, Jogador &jogador_02) override { return
      false; };
00028
00029     Jogador *JogadorX = nullptr;
00030     Jogador *JogadorO = nullptr;
00031     int ContadorTurnos = 0;
00032 };
00033
00034 #endif
```

5.19 Referência do Arquivo src/BatalhaNaval.cpp

```
#include "BatalhaNaval.hpp"
```

5.20 Referência do Arquivo src/CentralDeJogos.cpp

```
#include "CentralDeJogos.hpp"
```

5.21 Referência do Arquivo src/Estatisticas.cpp

Implementação dos métodos da classe Estatisticas.

```
#include "Estatisticas.hpp"
#include <iostream>
```

5.21.1 Descrição detalhada

Implementação dos métodos da classe Estatisticas.

Este arquivo contém a implementação dos métodos da classe Estatisticas, que gerencia as estatísticas de um jogador em um jogo específico, incluindo vitórias, derrotas e empates.

5.22 Referência do Arquivo src/Jogador.cpp

Implementação dos métodos da classe Jogador.

```
#include "Jogador.hpp"
#include <iostream>
```

5.22.1 Descrição detalhada

Implementação dos métodos da classe Jogador.

Este arquivo contém a implementação dos métodos da classe Jogador, que gerencia as informações e estatísticas de um jogador nos diferentes jogos disponíveis.

5.23 Referência do Arquivo src/JogoDaVelha.cpp

Implementação dos métodos da classe JogoDaVelha.

```
#include "JogoDaVelha.hpp"
```

5.23.1 Descrição detalhada

Implementação dos métodos da classe JogoDaVelha.

Este arquivo contém a implementação dos métodos da classe JogoDaVelha, que gerencia o jogo da velha, incluindo a inicialização do tabuleiro, leitura de jogadas, verificação de vitória e empate, e anúncio do início da partida.

5.24 Referência do Arquivo src/JogoDaVelhaAi.cpp

Implementação da lógica do jogo da velha com IA usando o algoritmo Minimax.

```
#include "JogoDaVelhaAi.hpp"
```

5.24.1 Descrição detalhada

Implementação da lógica do jogo da velha com IA usando o algoritmo Minimax.

Este arquivo contém a implementação das funções da classe JogoDaVelhaAi, incluindo a lógica de verificação da vitória, do empate, e do algoritmo Minimax para a IA.

5.25 Referência do Arquivo src/Jogos.cpp

Implementação dos métodos da classe Jogos.

```
#include "Jogos.hpp"
#include "Jogador.hpp"
```

5.25.1 Descrição detalhada

Implementação dos métodos da classe Jogos.

Este arquivo contém a implementação dos métodos da classe Jogos, que fornece funcionalidades básicas para gerenciar partidas de jogos de tabuleiro, como exibição do tabuleiro, controle de turnos, validação de jogadas e execução de partidas.

5.26 Referência do Arquivo src/Lig4.cpp

Implementação dos métodos da classe Lig4.

```
#include "Lig4.hpp"
```

5.26.1 Descrição detalhada

Implementação dos métodos da classe Lig4.

Este arquivo contém a implementação dos métodos da classe Lig4, que gerencia o jogo Lig4 (Connect Four), incluindo a inicialização do tabuleiro, leitura de jogadas, verificação de vitória e empate, e anúncio do início da partida.

5.27 Referência do Arquivo src/main.cpp

Função principal do sistema de gerenciamento de jogos.

```
#include <CentralDeJogos.hpp>
#include <iostream>
#include <stdexcept>
#include <vector>
```

Funções

- std::string validarEntrada ()
- void exibirMenu ()
- int main ()

5.27.1 Descrição detalhada

Função principal do sistema de gerenciamento de jogos.

Este arquivo contém a função `main`, que é o ponto de entrada do programa. A função gerencia a interação com o usuário, exibindo um menu de opções e executando as funcionalidades correspondentes, como cadastrar jogadores, remover jogadores, listar jogadores e executar partidas.

5.27.1.1 Fluxo do Programa:

1. Exibe um menu de opções para o usuário.
2. Lê a entrada do usuário e valida o comando.
3. Executa a funcionalidade correspondente ao comando:
   - **CJ**: Cadastra um novo jogador.
   - **RJ**: Remove um jogador existente.
   - **LJ**: Lista todos os jogadores cadastrados.
   - **EP**: Executa uma partida em um dos jogos disponíveis.
   - **FS**: Finaliza o sistema.
4. Repete o processo até que o usuário escolha a opção de finalizar o sistema.

5.27.1.2 Dependências:

- Utiliza a classe `CentralDeJogos` para gerenciar jogadores e partidas.
- Utiliza funções auxiliares como `validarEntrada` e `exibirMenu` para interação com o usuário.

Veja também

CentralDeJogos

5.27.2 Funções

5.27.2.1 exibirMenu()

```
void exibirMenu ()
```

5.27.2.2 main()

```
int main ()
```

5.27.2.3 validarEntrada()

```
std::string validarEntrada ()
```

5.28 Referência do Arquivo src/Reversi.cpp

```
#include "Reversi.hpp"
```

Índice Remissivo