Documentação - Projeto Final

Generated by Doxygen 1.8.17

Chapter 1

Projeto Final - Programação e Desenvolvimento de Software 2

1.1 Visão Geral

Este projeto final do curso de Programação e Desenvolvimento de Software 2 foi desenvolvido com o objetivo de aplicar conceitos fundamentais de orientação a objetos, boas práticas de programação e programação defensiva. O sistema implementado abrange a criação e gerenciamento de jogos clássicos, como Reversi e Lig4, incluindo funcionalidades como jogadas automáticas controladas por inteligência artificial (IA) e a possibilidade de configurar o tabuleiro de acordo com as preferências do usuário.

1.2 Desenvolvimento do Projeto

Durante o desenvolvimento, enfrentamos desafios significativos relacionados ao planejamento e à implementação das funcionalidades dos jogos. Decidir como as classes seriam estruturadas e quais métodos seriam necessários para garantir a flexibilidade e a extensibilidade do código foi um dos maiores obstáculos.

Além disso, integramos o Doxygen ao projeto para gerar uma documentação mais profissional, o que nos ajudou a entender e a manter o código mais organizado. A realização de testes foi crucial para assegurar que o sistema funcionasse corretamente em diversos cenários, contribuindo para a robustez e a confiabilidade do software.

Outro aspecto implementado foi a criação de uma interface amigável e intuitiva para o usuário, de forma a garantir uma boa experiência ao utilizador final.

1.3 Funcionalidades do Sistema

1.3.1 Jogos Implementados

- **Reversi**: Um clássico jogo de tabuleiro onde o objetivo é capturar o maior número de peças do oponente.
- **Lig4**: Um jogo estratégico em que dois jogadores alternam turnos para tentar alinhar quatro peças consecutivas.

1.3.2 IA e Jogabilidade

Nos jogos desenvolvidos, foi implementada uma inteligência artificial simples que permite que um jogador humano enfrente a máquina. A IA realiza jogadas válidas de maneira automatizada, alternando corretamente os turnos com o jogador.

1.3.3 Extensões Adicionais

- **Configuração de Tabuleiro**: No jogo Lig4, o usuário pode escolher as dimensões do tabuleiro antes de iniciar uma partida, oferecendo maior personalização e desafio.
- **Agente Inteligente (IA)**: Uma IA foi desenvolvida para substituir um dos jogadores em ambos os jogos, proporcionando uma experiência competitiva mesmo quando o jogador não tem um oponente humano disponível.

1.4 Documentação e Testes

A documentação do projeto foi gerada utilizando o Doxygen, que fornece uma visão clara das classes, métodos e interações no sistema. Além disso, foi criado um conjunto abrangente de testes para validar todas as funcionalidades do sistema, garantindo a integridade e a correção do código em diferentes cenários.

1.5 Conclusão

Este projeto representou uma oportunidade valiosa para aplicar os conhecimentos adquiridos ao longo do curso em um contexto prático. Através da implementação dos jogos e da integração das funcionalidades adicionais, adquirimos uma compreensão mais profunda das técnicas de desenvolvimento de software, desde a concepção até a entrega final.

O sucesso do projeto é evidenciado não apenas pela funcionalidade do sistema, mas também pelo desenvolvimento pessoal e técnico de cada membro da equipe. As dificuldades encontradas ao longo do processo serviram como importantes lições, preparando-nos para enfrentar futuros desafios na carreira de desenvolvimento de software.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Class Documentation

5.1 CadastroJogadores Class Reference

Classe responsável pelo gerenciamento de jogadores.

`#include <CadastroJogadores.hpp>`

Public Member Functions

- bool cadastrarJogador (const std::string &apelido, const std::string &nome)

  Cadastra um novo jogador.
- bool removerJogador (const std::string &apelido)

  Remove um jogador do cadastro.
- Jogador * buscarJogador (const std::string &apelido)

  Busca um jogador pelo apelido.
- void listarJogadores (char criterio) const

  Lista todos os jogadores cadastrados.

5.1.1 Detailed Description

Classe responsável pelo gerenciamento de jogadores.

A classe CadastroJogadores permite o cadastro, remoção, busca e listagem de jogadores.

5.1.2 Member Function Documentation

5.1.2.1 buscarJogador()

```
Jogador * CadastroJogadores::buscarJogador (
            const std::string & apelido )
```

Busca um jogador pelo apelido.

Parameters

apelido	O apelido do jogador a ser buscado.

Returns

Jogador* Ponteiro para o jogador encontrado ou nullptr se não for encontrado.

Procura um jogador com o apelido fornecido e retorna um ponteiro para ele.

Parameters

apelido	O apelido do jogador a ser buscado.

Returns

Jogador* Ponteiro para o jogador encontrado ou nullptr se não for encontrado.

5.1.2.2 cadastrarJogador()

```
bool CadastroJogadores::cadastrarJogador (
            const std::string & apelido,
            const std::string & nome )
```

Cadastra um novo jogador.

Parameters

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

Returns

true Se o jogador foi cadastrado com sucesso.

false Se o jogador já existe ou se houve algum problema no cadastro.

Verifica se o jogador com o apelido fornecido já existe no cadastro. Se não existir, adiciona o jogador.

Parameters

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

Returns

true Se o jogador foi cadastrado com sucesso.

false Se o jogador já existe no cadastro.

5.1.2.3 listarJogadores()

```
void CadastroJogadores::listarJogadores (
            char criterio ) const
```

Lista todos os jogadores cadastrados.

Parameters

criterio	Critério de ordenação ('a' para apelido, 'n' para nome).

Ordena os jogadores com base no critério fornecido e imprime as estatísticas de cada jogador.

Parameters

criterio	Critério de ordenação ('A' para ordenar por apelido, 'N' para ordenar por nome).

5.1.2.4 removerJogador()

```
bool CadastroJogadores::removerJogador (
            const std::string & apelido )
```

Remove um jogador do cadastro.

Parameters

apelido	O apelido do jogador a ser removido.

Returns

true Se o jogador foi removido com sucesso.

false Se o jogador não foi encontrado no cadastro.

Procura um jogador com o apelido fornecido e o remove do cadastro.

Parameters

apelido	O apelido do jogador a ser removido.

Returns

true Se o jogador foi removido com sucesso.

false Se o jogador não foi encontrado no cadastro.

The documentation for this class was generated from the following files:

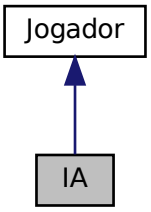
- include/CadastroJogadores.hpp
- src/CadastroJogadores.cpp

5.2 IA Class Reference

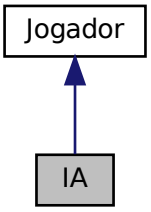
Classe responsável por representar um jogador controlado por inteligência artificial.

`#include <IA.hpp>`

Inheritance diagram for IA:



Collaboration diagram for IA:



Public Member Functions

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

  Construtor da classe IA.
- std::pair< int, int > calcularJogada (Jogo *jogo)

  Calcula a próxima jogada com base no estado atual do jogo.

5.2.1 Detailed Description

Classe responsável por representar um jogador controlado por inteligência artificial.

A classe IA pode substituir um jogador humano e fazer jogadas baseadas em estratégias definidas.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 IA()

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

Construtor da classe IA.

Parameters

nome	O nome da IA.
apelido	O apelido da IA.

5.2.3 Member Function Documentation

5.2.3.1 calcularJogada()

```
std::pair< int, int > IA::calcularJogada (
            Jogo * jogo )
```

Calcula a próxima jogada com base no estado atual do jogo.

Parameters

jogo	Ponteiro para o jogo atual.

Returns

Um par de inteiros representando a linha e a coluna da jogada escolhida.

The documentation for this class was generated from the following files:

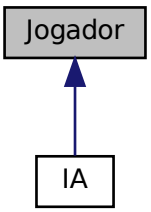
- include/IA.hpp
- src/IA.cpp

5.3 Jogador Class Reference

Armazenar informações do jogador.

`#include <Jogador.hpp>`

Inheritance diagram for Jogador:



Public Member Functions

- Jogador (const std::string &apelido, const std::string &nome)
  Cria um jogador.
- std::string getApelido () const
  Obtém o apelido do jogador.
- std::string getNome () const
  Obtém o nome do jogador.
- void adicionarVitoria (const std::string &jogo)
  Adiciona uma vitoria ao jogo jogado.
- void adicionarDerrota (const std::string &jogo)
  Adiciona uma derrota ao jogo jogado.
- void imprimirEstatisticas () const
  Imprime todas as estatisticas de vitoria e derrota.

5.3.1 Detailed Description

Armazenar informações do jogador.

A classe Jogador armazena o Apelido, Nome, numero de Vitorias e Derrotas do jogador em cada jogo.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Jogador()

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

Cria um jogador.

Parameters

apelido	O apelido atribuido ao jogador criado.
nome	O nome atribuido ao jogador criado.

Cria um jogador e inicia parametros de vitória e derrota com valor zero.

Parameters

apelido	O apelido atribuido ao jogador criado.
nome	O nome atribuido ao jogador criado.

5.3.3 Member Function Documentation

5.3.3.1 adicionarDerrota()

```
void Jogador::adicionarDerrota (
            const std::string & jogo )
```

Adiciona uma derrota ao jogo jogado.

Adiciona uma derrota ao jogador no jogo jogado.

Parameters

jogo	Define qual jogo foi jogado.

Analisa o valor do parametro "jogo" para adicionar a derrota ao tipo de jogo jogado.

Parameters

jogo	Defini qual tipo de jogo foi a derrota.

5.3.3.2 adicionarVitoria()

```
void Jogador::adicionarVitoria (
            const std::string & jogo )
```

Adiciona uma vitoria ao jogo jogado.

Parameters

jogo	Define qual jogo foi jogado.

Analisa o valor do parametro "jogo" para adicionar a vitoria ao tipo de jogo jogado.

Parameters

jogo	Defini qual tipo de jogo foi a vitória.

5.3.3.3 getApelido()

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

Obtém o apelido do jogador.

Returns

Retorna o apelido do jogador.

5.3.3.4 getNome()

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

Obtém o nome do jogador.

Returns

Retorna o nome do jogador.

5.3.3.5 imprimirEstatisticas()

```
void Jogador::imprimirEstatisticas ( ) const
```

Imprime todas as estatisticas de vitoria e derrota.

Serpara as estatísticas de vitoria e derrota de acordo com o tipo de jogo.

The documentation for this class was generated from the following files:

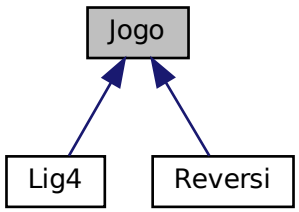
- include/Jogador.hpp
- src/Jogador.cpp

5.4 Jogo Class Reference

Classe base abstrata para jogos.

`#include <Jogo.hpp>`

Inheritance diagram for Jogo:



Public Member Functions

- virtual ~Jogo ()=default
  Destrutor virtual.
- virtual int getAltura () const =0
  Obtém a altura do tabuleiro.
- virtual int getLargura () const =0
  Obtém a largura do tabuleiro.
- virtual void iniciar ()=0
  Inicializa o jogo.
- virtual void imprimirTabuleiro ()=0
  Imprime o tabuleiro do jogo.
- virtual bool validarJogada (int linha, int coluna)=0
  Valida uma jogada.
- virtual bool verificarVitoria ()=0
  Verifica se há um vencedor.
- virtual void realizarJogada (int linha, int coluna)=0
  Realiza uma jogada.
- virtual void realizarJogadaIA ()=0
  Realiza uma jogada automática pela IA.
- virtual bool empatePartida ()=0

5.4.1 Detailed Description

Classe base abstrata para jogos.

Esta classe define a interface para os jogos, fornecendo métodos que devem ser implementados por qualquer classe derivada que represente um jogo específico. Inclui métodos para iniciar o jogo, validar e realizar jogadas, imprimir o tabuleiro e verificar a vitória.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 ~Jogo()

```
virtual Jogo::~Jogo ( )  [virtual], [default]
```

Destrutor virtual.

O destrutor é virtual para garantir que o destruidor da classe derivada seja chamado corretamente quando um objeto da classe derivada for destruído através de um ponteiro para a classe base.

5.4.3 Member Function Documentation

5.4.3.1 empatePartida()

```
virtual bool Jogo::empatePartida ( )  [pure virtual]
```

Implemented in Lig4, and Reversi.

5.4.3.2 getAltura()

```
virtual int Jogo::getAltura ( ) const  [pure virtual]
```

Obtém a altura do tabuleiro.

Returns

A altura do tabuleiro.

Implemented in Lig4, and Reversi.

5.4.3.3 getLargura()

```
virtual int Jogo::getLargura ( ) const  [pure virtual]
```

Obtém a largura do tabuleiro.

Returns

A largura do tabuleiro.

Implemented in Lig4, and Reversi.

5.4.3.4 imprimirTabuleiro()

```
virtual void Jogo::imprimirTabuleiro ( )  [pure virtual]
```

Imprime o tabuleiro do jogo.

Este método deve exibir o estado atual do tabuleiro na saída padrão, permitindo que os jogadores visualizem a disposição dos elementos no tabuleiro.

Implemented in Lig4, and Reversi.

5.4.3.5 iniciar()

```
virtual void Jogo::iniciar ( )  [pure virtual]
```

Inicializa o jogo.

Este método deve preparar o jogo para o início, configurando o estado inicial do tabuleiro e quaisquer outras variáveis necessárias.

Implemented in Lig4, and Reversi.

5.4.3.6 realizarJogada()

```
virtual void Jogo::realizarJogada (
            int linha,
            int coluna )  [pure virtual]
```

Realiza uma jogada.

Parameters

linha	Linha onde a jogada será realizada.
coluna	Coluna onde a jogada será realizada.

Este método deve atualizar o estado do tabuleiro com a jogada realizada na posição especificada.

Implemented in Lig4, and Reversi.

5.4.3.7 realizarJogadaIA()

```
virtual void Jogo::realizarJogadaIA ( )  [pure virtual]
```

Realiza uma jogada automática pela IA.

Este método deve ser implementado para permitir que a IA faça uma jogada automática com base na configuração atual do tabuleiro.

Implemented in Lig4, and Reversi.

5.4.3.8 validarJogada()

```
virtual bool Jogo::validarJogada (
            int linha,
            int coluna )  [pure virtual]
```

Valida uma jogada.

Parameters

linha	Linha onde a jogada será realizada.
coluna	Coluna onde a jogada será realizada.

Returns

Verdadeiro se a jogada for válida, falso caso contrário.

Implemented in Lig4, and Reversi.

5.4.3.9 verificarVitoria()

```
virtual bool Jogo::verificarVitoria ( )  [pure virtual]
```

Verifica se há um vencedor.

Returns

Verdadeiro se houver um vencedor, falso caso contrário.

Implemented in Lig4, and Reversi.

The documentation for this class was generated from the following file:

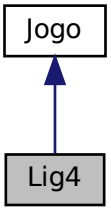
- include/Jogo.hpp

5.5 Lig4 Class Reference

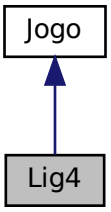
Classe responsável pelo gerenciamento do jogo Lig4, derivada da classe Jogo.

`#include <Lig4.hpp>`

Inheritance diagram for Lig4:



Collaboration diagram for Lig4:



Public Member Functions

- void iniciar () override

  Inicia um novo jogo de Lig4.
- void imprimirTabuleiro () override

  Imprime o estado atual do tabuleiro.
- bool validarJogada (int linha, int coluna) override

  Valida se a posicao da jogada é possivel.
- bool verificarVitoria () override

  Verica se houve vitória.
- void realizarJogada (int linha, int coluna) override

  Realiza a jogada no tabuleiro.
- void realizarJogadaIA () override

Realiza a jogada da IA.

- int getAltura () const override

  Retorna o número de linhas (altura) do tabuleiro.
- int getLargura () const override

  Retorna o número de colunas (largura) do tabuleiro.
- bool empatePartida () override

  Verifica seu acabou o numero de jogadas possiveis.
- void alternarJogador ()

  Alterna o jogador que vai jogar.
- void realizarJogadaNaColuna (int coluna)

  Realiza uma jogada na coluna especificada.

5.5.1 Detailed Description

Classe responsável pelo gerenciamento do jogo Lig4, derivada da classe Jogo.

A classe Lig4 gerencia o estado e as regras do jogo Lig4. Ela herda todos os métodos da classe Jogo e implementa funcionalidades específicas para o jogo Lig4, como a alternância de jogadores, validação de jogadas, realização de jogadas e verificação de vitória.

5.5.2 Member Function Documentation

5.5.2.1 alternarJogador()

```
void Lig4::alternarJogador ( )
```

Alterna o jogador que vai jogar.

Alterna o jogador atual.

Este método troca o jogador atual, alternando entre 'X' e 'O'. Se o jogador atual for 'X', ele será alterado para 'O'. Caso contrário, será alterado para 'X'.

5.5.2.2 empatePartida()

```
bool Lig4::empatePartida ( )  [override], [virtual]
```

Verifica seu acabou o numero de jogadas possiveis.

Parameters

_numJogadas	É o numero de posições ja ocupadas no tabuleiro
_numJogadas	É o numero de posições ja ocupadas no tabuleiro

Returns

'true' se numero de jogadas igual ao maximo possivel 'false' caso contrário

Note

"_numJogadas" é dividida por 2 pois e chamda 2x na por jogada

Implements Jogo.

5.5.2.3 getAltura()

```
int Lig4::getAltura ( ) const  [override], [virtual]
```

Retorna o número de linhas (altura) do tabuleiro.

Returns

Número de linhas do tabuleiro.

Implements Jogo.

5.5.2.4 getLargura()

```
int Lig4::getLargura ( ) const  [override], [virtual]
```

Retorna o número de colunas (largura) do tabuleiro.

Returns

Número de colunas do tabuleiro.

Implements Jogo.

5.5.2.5 imprimirTabuleiro()

```
void Lig4::imprimirTabuleiro ( )  [override], [virtual]
```

Imprime o estado atual do tabuleiro.

Este método percorre o tabuleiro e imprime cada célula, formatando o tabuleiro para que cada linha seja representada por uma linha de caracteres no console. Cada célula é delimitada por um pipe ('|'). No final, as colunas do tabuleiro são numeradas de 0 a 6.

Implements Jogo.

5.5.2.6 iniciar()

```
void Lig4::iniciar ( )  [override], [virtual]
```

Inicia um novo jogo de Lig4.

Esta metodo chama a função criarTabuleiro para inicializa-lo. Ela também define o jogador atual como 'X', dando uma breve descrição de como jogar e exibe o estado inicial do tabuleiro.

Note

Esta função deve ser chamada no início de cada nova partida.

Implements Jogo.

5.5.2.7 realizarJogada()

```
void Lig4::realizarJogada (
            int linha,
            int coluna )  [override], [virtual]
```

Realiza a jogada no tabuleiro.

Parameters

linha	A linha onde a jogada será realizada.
coluna	A coluna onde a jogada será realizada.

Este método verifica se a jogada na coluna especificada é válida. Se for válida, a jogada é realizada naquela coluna, o jogador atual é alternado, e o estado atualizado do tabuleiro é impresso. Caso a jogada seja inválida, uma mensagem de erro é exibida.

Parameters

linha	A linha onde a jogada seria realizada (não utilizada diretamente, pois a jogada ocorre na coluna mais baixa disponível).
coluna	A coluna onde a jogada será realizada.

Implements Jogo.

5.5.2.8 realizarJogadaIA()

```
void Lig4::realizarJogadaIA ( )  [override], [virtual]
```

Realiza a jogada da IA.

Realiza uma jogada automática pela IA.

Este método é responsável por decidir e realizar uma jogada automática pela IA.

A IA escolhe uma jogada com base na configuração atual do tabuleiro e realiza a jogada.

Implements Jogo.

5.5.2.9 realizarJogadaNaColuna()

```
void Lig4::realizarJogadaNaColuna (
            int coluna )
```

Realiza uma jogada na coluna especificada.

Parameters

coluna	A coluna onde a jogada será realizada.

Este método insere a peça do jogador atual na primeira posição disponível (de baixo para cima) na coluna especificada. A jogada é realizada preenchendo a primeira célula vazia encontrada na coluna.

Parameters

coluna	A coluna onde a jogada será realizada.

5.5.2.10 validarJogada()

```
bool Lig4::validarJogada (
            int linha,
            int coluna )  [override], [virtual]
```

Valida se a posicao da jogada é possivel.

Parameters

linha	A linha onde a jogada será realizada.
coluna	A coluna onde a jogada será realizada.

Returns

`true` se a jogada for válida, `false` caso contrário.

A funcao jodaValida retorna 'true' e 'false'

Parameters

linha	A linha onde a jogada será realizada.
coluna	A coluna onde a jogada será realizada.

Returns

`true` se a jogada for válida, `false` caso contrário.

Implements Jogo.

5.5.2.11 verificarVitoria()

```
bool Lig4::verificarVitoria ( )  [override], [virtual]
```

Verica se houve vitória.

Verifica se houve vitória no jogo.

Returns

'true' se houve vitória,'false' caso contrário.

Este método percorre todo o tabuleiro de Lig4. Para cada célula que não está vazia, ele verifica se há uma sequência de quatro peças consecutivas em qualquer uma das direções possíveis (horizontal, vertical, diagonal para cima, ou diagonal para baixo). Se for encontrada uma sequência vencedora, a função retorna 'true'.

Returns

`true` se houve vitória, `false` caso contrário.

Implements Jogo.

The documentation for this class was generated from the following files:

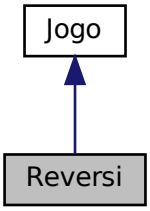
- include/Lig4.hpp
- src/Lig4.cpp

5.6 Reversi Class Reference

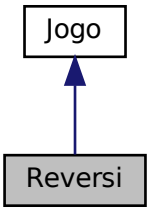
Classe responsável pelo gerenciamento do jogo Reverse, derivada da classe Jogo.

`#include <Reversi.hpp>`

Inheritance diagram for Reversi:



Collaboration diagram for Reversi:



Public Member Functions

- void iniciar () override

  Inicia um novo jogo de Reversi.
- void imprimirTabuleiro () override

  Imprime o estado atual do tabuleiro.
- bool validarJogada (int linha, int coluna) override

  Valida se a posicao da jogada é possivel.
- bool verificarVitoria () override

  Verifica se o jogo terminou por falta de jogadas válidas.
- void realizarJogada (int linha, int coluna) override

  Realiza a jogada no tabuleiro.
- void realizarJogadaIA () override

Realiza a jogada da IA.

- int getAltura () const override

  Obtém a altura do tabuleiro do jogo Reversi.

- int getLargura () const override

  Obtém a largura do tabuleiro do jogo Reversi.

- bool empatePartida () override

Public Attributes

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

  Representação do tabuleiro do jogo lig4, com 6 linhas e 7 colunas.

- char jogadorAtual

  Armazena o jogador atual.

5.6.1 Detailed Description

Classe responsável pelo gerenciamento do jogo Reverse, derivada da classe Jogo.

A classe Reversi gerencia o estado e as regras do jogo Reversi. Ela herda todos os métodos da classe Jogo e implementa funcionalidades específicas para o jogo Reversi, como a alternância de jogadores, validação de jogadas, realização de jogadas e verificação de vitória.

5.6.2 Member Function Documentation

5.6.2.1 empatePartida()

```
bool Reversi::empatePartida ( )  [override], [virtual]
```

Implements Jogo.

5.6.2.2 getAltura()

```
int Reversi::getAltura ( ) const  [override], [virtual]
```

Obtém a altura do tabuleiro do jogo Reversi.

Returns

A altura do tabuleiro.

Implements Jogo.

5.6.2.3 getLargura()

```
int Reversi::getLargura ( ) const  [override], [virtual]
```

Obtém a largura do tabuleiro do jogo Reversi.

Returns

A largura do tabuleiro.

Implements Jogo.

5.6.2.4 imprimirTabuleiro()

```
void Reversi::imprimirTabuleiro ( )  [override], [virtual]
```

Imprime o estado atual do tabuleiro.

Este método percorre o tabuleiro e imprime cada célula, formatando o tabuleiro para que cada linha seja representada por uma linha de caracteres no console. Cada célula é delimitada por um pipe ('|'). No final, as colunas e linhas do tabuleiro são numeradas de 0 a 7.

Implements Jogo.

5.6.2.5 iniciar()

```
void Reversi::iniciar ( )  [override], [virtual]
```

Inicia um novo jogo de Reversi.

Esta função inicializa o tabuleiro de Reversi com 8 linhas e 8 colunas, preenchendo 4 espaços no meio (com X e O) e o resto com espaços em branco (' '). Ela também define o jogador atual como 'X', dando uma breve descrição de como jogar e exibe o estado inicial do tabuleiro.

Note

Esta função deve ser chamada no início de cada nova partida.

Implements Jogo.

5.6.2.6 realizarJogada()

```
void Reversi::realizarJogada (
            int linha,
            int coluna )  [override], [virtual]
```

Realiza a jogada no tabuleiro.

Realiza a jogada do jogador atual na posição especificada.

Parameters

linha	A linha onde a jogada será realizada.
coluna	A coluna onde a jogada será realizada.

Este método verifica se a jogada na posição especificada (linha e coluna) é válida para o jogador atual. Se a jogada for válida, o método atualiza o tabuleiro, vira as peças do oponente que foram capturadas, alterna para o próximo jogador e imprime o estado atualizado do tabuleiro.

O método funciona da seguinte maneira:

- Verifica se a jogada é válida chamando o método 'jogadaValida' para a posição especificada.
- Se a jogada for válida:
  - A peça do jogador atual é colocada na posição indicada no tabuleiro.
  - O método 'virarPecas' é chamado para capturar as peças do oponente.
  - O jogador atual é alternado utilizando o método 'alternarJogador'.
  - O tabuleiro atualizado é impresso chamando o método 'imprimirTabuleiro'.
- Se a jogada não for válida, uma mensagem de erro é exibida no console.

Parameters

linha	A linha onde a jogada será realizada.
coluna	A coluna onde a jogada será realizada.

Implements Jogo.

5.6.2.7 realizarJogadaIA()

```
void Reversi::realizarJogadaIA ( )  [override], [virtual]
```

Realiza a jogada da IA.

Realiza uma jogada automática pela IA.

Este método é responsável por decidir e realizar uma jogada automática pela IA.

A IA escolhe uma jogada com base na configuração atual do tabuleiro e realiza a jogada.

Implements Jogo.

5.6.2.8 validarJogada()

```
bool Reversi::validarJogada (
            int linha,
            int coluna )  [override], [virtual]
```

Valida se a posicao da jogada é possivel.

Parameters

linha	A linha onde a jogada será realizada.
coluna	A coluna onde a jogada será realizada.

Returns

`true` se a jogada for válida, `false` caso contrário.

A funcao jodaValida retorna 'true' e 'false'

Parameters

linha	A linha onde a jogada será realizada.
coluna	A coluna onde a jogada será realizada.

Returns

`true` se a jogada for válida, `false` caso contrário.

Implements Jogo.

5.6.2.9 verificarVitoria()

```
bool Reversi::verificarVitoria ( )  [override], [virtual]
```

Verifica se o jogo terminou por falta de jogadas válidas.

Returns

'true' se houve vitória,'false' caso contrário.

Este método percorre todo o tabuleiro e verifica se ainda há jogadas válidas disponíveis para qualquer um dos jogadores ('X' ou 'O'). O jogo é considerado terminado se não houver jogadas válidas restantes para nenhum dos jogadores.

O método funciona da seguinte forma:

- Para cada posição do tabuleiro 8x8, verifica se a posição permite uma jogada válida para 'X' ou 'O'.
- Se for encontrada pelo menos uma jogada válida, o método retorna `false`, indicando que o jogo pode continuar.
- Se nenhuma jogada válida for encontrada em todo o tabuleiro, o método retorna `true`, indicando que o jogo terminou.

Returns

`true` se o jogo terminou (nenhuma jogada válida restante para qualquer jogador), `false` caso contrário.

Implements Jogo.

5.6.3 Member Data Documentation

5.6.3.1 jogadorAtual

```
char Reversi::jogadorAtual
```

Armazena o jogador atual.

O jogador atual pode ser ou X ou O.

5.6.3.2 tabuleiro

```
std::vector<std::vector<char> > Reversi::tabuleiro
```

Representação do tabuleiro do jogo lig4, com 6 linhas e 7 colunas.

The documentation for this class was generated from the following files:

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

5.7 Sistema Class Reference

Gerencia o sistema de jogos e cadastro de jogadores.

```
#include <Sistema.hpp>
```

Public Member Functions

- void executar ()

  Executa o loop principal do sistema.
- bool cadastrarJogador (const std::string &apelido, const std::string &nome)

  Cadastra um novo jogador.
- bool removerJogador (const std::string &apelido)

  Remove um jogador cadastrado.
- void listarJogadores (char criterio)

  Lista os jogadores cadastrados.
- bool executarPartida (const std::string &jogo, const std::string &apelido1, const std::string &apelido2)

  Executa uma partida entre dois jogadores.
- void finalizarSistema ()

  Imprime uma mensagem de finalição da partida.

5.7.1 Detailed Description

Gerencia o sistema de jogos e cadastro de jogadores.

A classe Sistema é responsável por gerenciar todas as operações do sistema, incluindo o gerenciamento de jogos (iniciar e executar partidas), bem como o cadastro, remoção e listagem de jogadores.

5.7.2 Member Function Documentation

5.7.2.1 cadastrarJogador()

```
bool Sistema::cadastrarJogador (
            const std::string & apelido,
            const std::string & nome )
```

Cadastra um novo jogador.

Parameters

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

Returns

'true' se o jogador foi cadastrado com sucesso, 'false' se o jogador já existe ou se houve algum problema no cadastro.

Este método chama a função `cadastrarJogador` da classe `CadastroJogadores` para registrar um novo jogador no sistema.

Parameters

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

Returns

'true' se o jogador foi cadastrado com sucesso, 'false' se o jogador já existe ou se houve algum problema no cadastro.

5.7.2.2 executar()

```
void Sistema::executar ( )
```

Executa o loop principal do sistema.

A implementação do loop principal está localizada no main.cpp.

5.7.2.3 executarPartida()

```
bool Sistema::executarPartida (
            const std::string & jogo,
            const std::string & apelido1,
            const std::string & apelido2 )
```

Executa uma partida entre dois jogadores.

Parameters

jogo	Qual o tipo de jogo será jogado
apelido1	Apelido do jogador 1.
apelido2	Apelido do jogador 2. Se definido como "IA", o jogador 1 jogará contra o computador.

Returns

'true' quando a partida foi executada com sucesso, 'false' se houve algum problema na execução.

Este método faz o seguinte:

- Verifica se ambos os jogadores estão cadastrados.
- Cria o jogo especificado (Reversi ou Lig4).
- Gerencia os turnos dos jogadores, validando e executando jogadas.
- Permite que os jogadores saiam no meio do jogo digitando "SAIR".
- Detecta se houve uma saida de jogador e determina o vencedor.
- Adiciona vitria e derrota nas estatísticas dos jogares.

Parameters

jogo	O tipo de jogo a ser jogado ("R" para Reversi, "L" para Lig4).
apelido1	O apelido do jogador 1.
apelido2	O apelido do jogador 2. Caso seja definido como "IA", o jogador 1 joga contra o computador.

Returns

`true` se a partida foi concluída com sucesso, `false` se houve algum problema na execução, como jogadores não encontrados ou jogo inválido.

5.7.2.4 finalizarSistema()

```
void Sistema::finalizarSistema ( )
```

Imprime uma mensagem de finalição da partida.

5.7.2.5 listarJogadores()

```
void Sistema::listarJogadores (
            char criterio )
```

Lista os jogadores cadastrados.

Parameters

criterio	O critério de ordenação para a listagem de jogadores e suas informações.

Este método chama a função `listarJogadores` da classe `CadastroJogadores` para listar os jogadores cadastrados, de acordo com o critério especificado.

Parameters

criterio	O critério de ordenação para a listagem de jogadores.

5.7.2.6 removerJogador()

```
bool Sistema::removerJogador (
            const std::string & apelido )
```

Remove um jogador cadastrado.

Parameters

apelido	O apelido do jogador a ser removido.

Returns

`true` se o jogador foi removido com sucesso, `false` se o jogador não existe ou se houve algum problema na remoção.

Este método chama a função `removerJogador` da classe `CadastroJogadores` para remover um jogador existente do sistema.

Parameters

apelido	O apelido do jogador a ser removido.

Returns

`true` se o jogador foi removido com sucesso, `false` se o jogador não existe ou se houve algum problema na remoção.

The documentation for this class was generated from the following files:

- include/Sistema.hpp
- src/Sistema.cpp

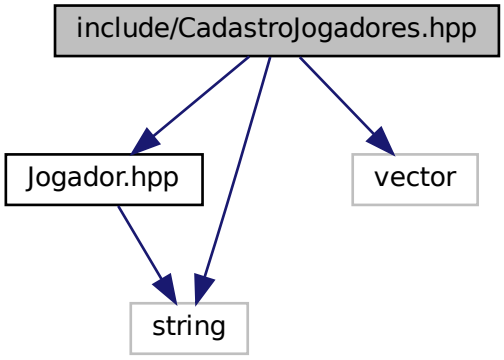
Chapter 6

File Documentation

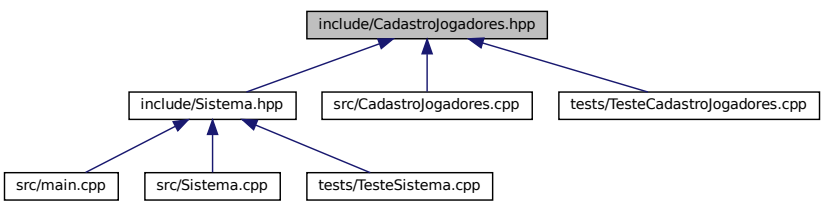
6.1 include/CadastroJogadores.hpp File Reference

```
#include "Jogador.hpp"
#include <vector>
#include <string>
```

Include dependency graph for CadastroJogadores.hpp:



This graph shows which files directly or indirectly include this file:



Classes

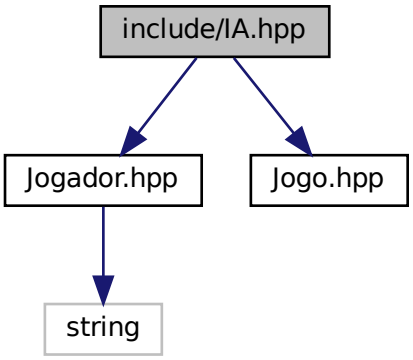
- class CadastroJogadores

  Classe responsável pelo gerenciamento de jogadores.

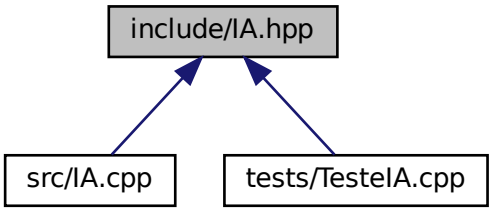
6.2 include/IA.hpp File Reference

```
#include "Jogador.hpp"
#include "Jogo.hpp"
```

Include dependency graph for IA.hpp:



This graph shows which files directly or indirectly include this file:



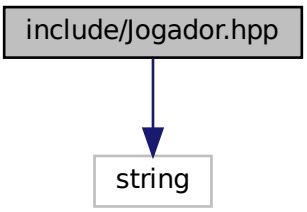
Classes

- class IA

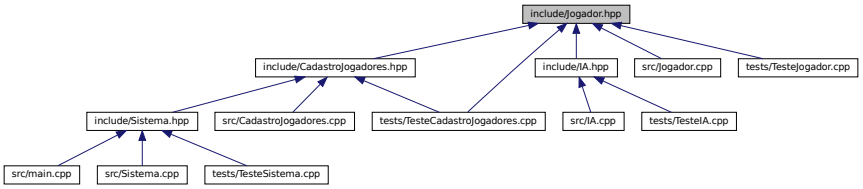
  Classe responsável por representar um jogador controlado por inteligência artificial.

6.3 include/Jogador.hpp File Reference

`#include <string>`
Include dependency graph for Jogador.hpp:



This graph shows which files directly or indirectly include this file:



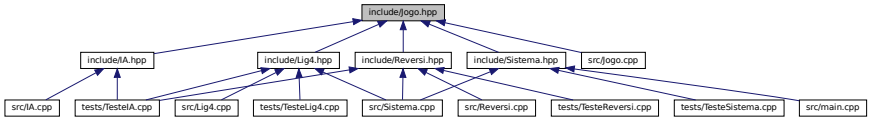
Classes

- class Jogador

  Armazenar informações do jogador.

6.4 include/Jogo.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

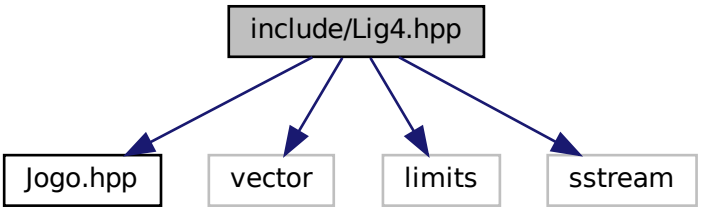
- class Jogo

  Classe base abstrata para jogos.

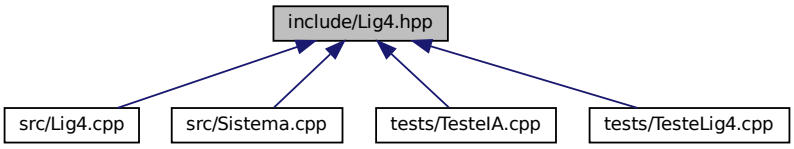
6.5 include/Lig4.hpp File Reference

```
#include "Jogo.hpp"
#include <vector>
#include <limits>
#include <sstream>
```

Include dependency graph for Lig4.hpp:



This graph shows which files directly or indirectly include this file:



Classes

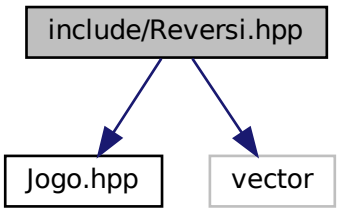
- class Lig4

  Classe responsável pelo gerenciamento do jogo Lig4, derivada da classe Jogo.

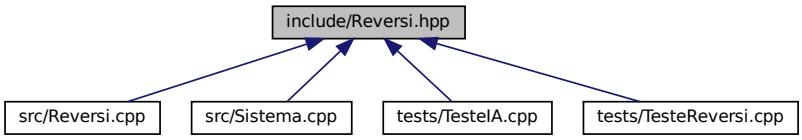
6.6 include/Reversi.hpp File Reference

```
#include "Jogo.hpp"
#include <vector>
```

Include dependency graph for Reversi.hpp:



This graph shows which files directly or indirectly include this file:



Classes

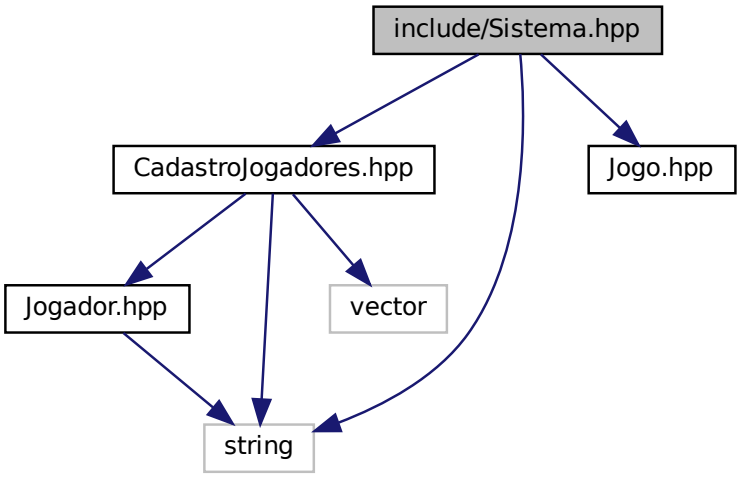
- class Reversi

  Classe responsável pelo gerenciamento do jogo Reverse, derivada da classe Jogo.

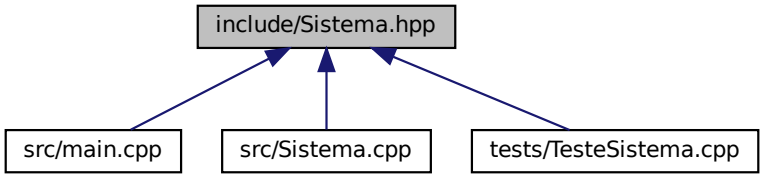
6.7 include/Sistema.hpp File Reference

```
#include "CadastroJogadores.hpp"
#include "Jogo.hpp"
#include <string>
```

Include dependency graph for Sistema.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Sistema

  Gerencia o sistema de jogos e cadastro de jogadores.

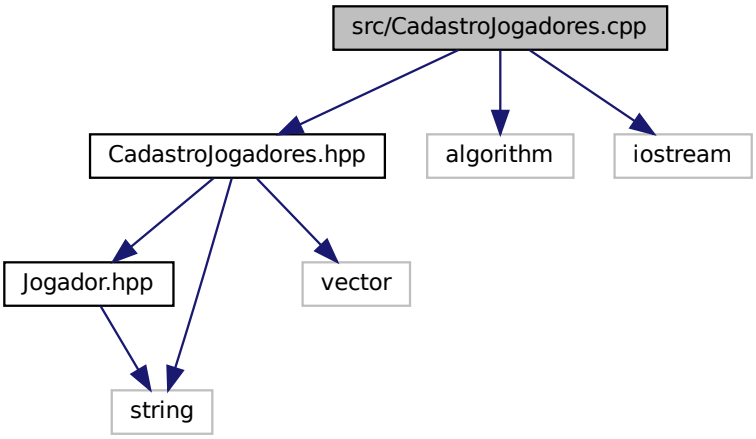
6.8 README.md File Reference

6.9 src/CadastroJogadores.cpp File Reference

```
#include "CadastroJogadores.hpp"
#include <algorithm>
```

```
#include <iostream>
```

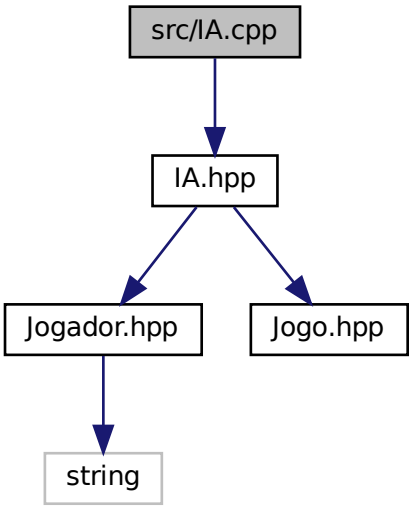
Include dependency graph for CadastroJogadores.cpp:



6.10 src/IA.cpp File Reference

```
#include "IA.hpp"
```

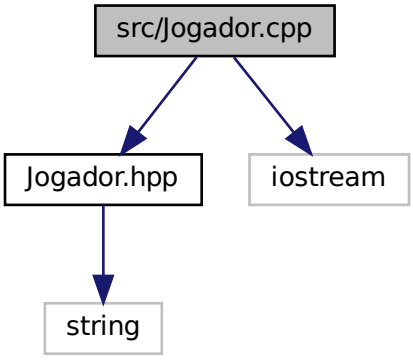
Include dependency graph for IA.cpp:



6.11 src/Jogador.cpp File Reference

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

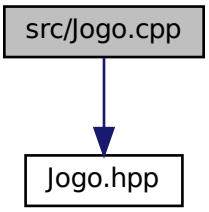
Include dependency graph for Jogador.cpp:



6.12 src/Jogo.cpp File Reference

```
#include "Jogo.hpp"
```

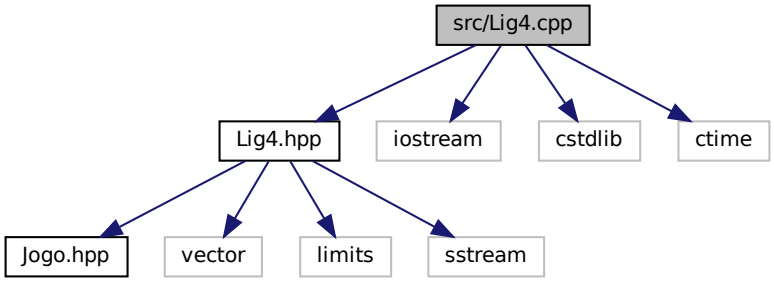
Include dependency graph for Jogo.cpp:



6.13 src/Lig4.cpp File Reference

```
#include "Lig4.hpp"
#include <iostream>
#include <cstdlib>
```

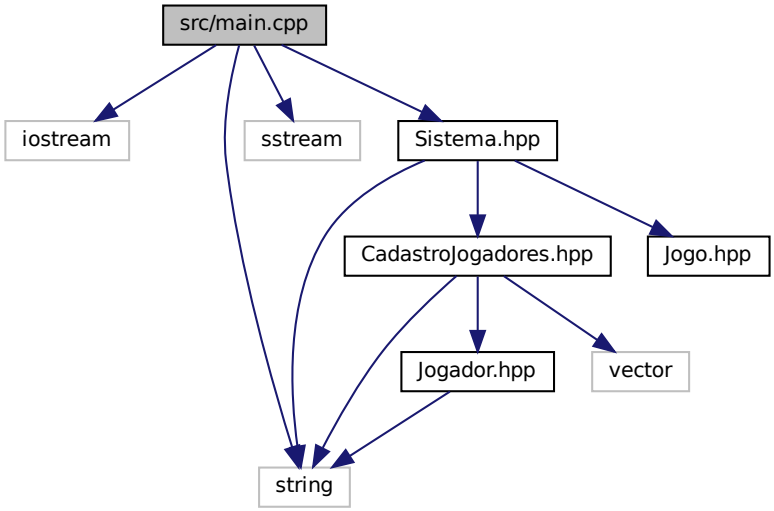
```
#include <ctime>
```
Include dependency graph for Lig4.cpp:



6.14 src/main.cpp File Reference

Ponto de entrada para o sistema de gerenciamento de jogos.

```
#include <iostream>
#include <string>
#include <sstream>
#include "Sistema.hpp"
```
Include dependency graph for main.cpp:



Functions

- int main ()

  Função principal do programa.

6.14.1 Detailed Description

Ponto de entrada para o sistema de gerenciamento de jogos.

Este arquivo contém o ponto de entrada principal para o sistema. Ele permite que o usuário interaja com o sistema por meio de comandos para cadastrar jogadores, remover jogadores, listar jogadores, e iniciar jogos de Reversi ou Lig4. O programa continua executando até que o comando "FS" seja digitado, momento em que o sistema é finalizado.

6.14.2 Function Documentation

6.14.2.1 main()

```
int main ( )
```

Função principal do programa.

Esta função é o ponto de entrada do programa e fornece uma interface de linha de comando para o usuário interagir com o sistema. Os comandos aceitos são:

- CJ 'apelido' 'nome' : Cadastra um novo jogador.
- RJ 'apelido' : Remove um jogador existente.
- LJ 'criterio' : Lista os jogadores cadastrados com base no critério ('N' para nome, 'A' para apelido).
- EP 'jogo' 'apelido1' 'apelido2' : Inicia um jogo entre dois jogadores (R para Reversi, L para Lig4).
- EP 'jogo' 'apelido1' 'IA' : Inicia um jogo entre um jogador e a IA.
- FS : Finaliza o sistema.

O programa continua em execução até que o comando "FS" seja inserido.

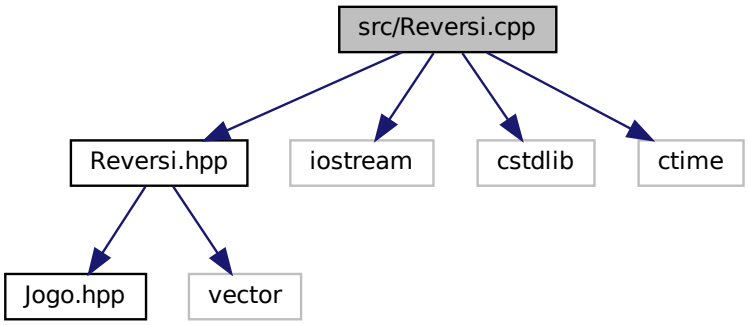
Returns

Retorna 0 após o sistema ser finalizado.

6.15 src/Reversi.cpp File Reference

```
#include "Reversi.hpp"
#include <iostream>
#include <cstdlib>
#include <ctime>
```

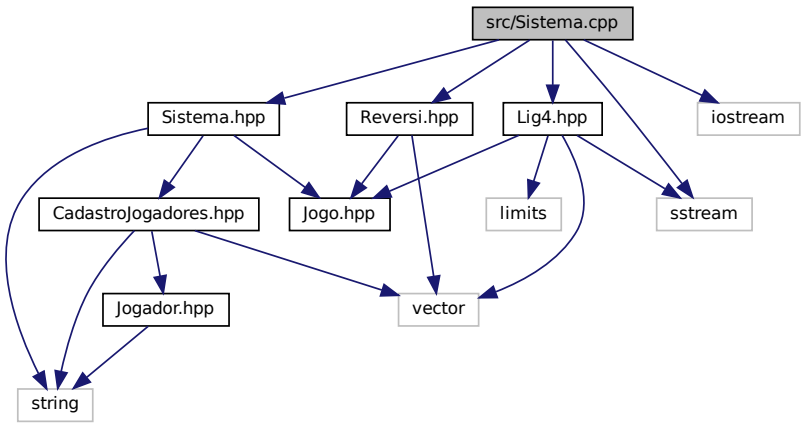
Include dependency graph for Reversi.cpp:



6.16 src/Sistema.cpp File Reference

```
#include "Sistema.hpp"
#include "Reversi.hpp"
#include "Lig4.hpp"
#include <iostream>
#include <sstream>
```

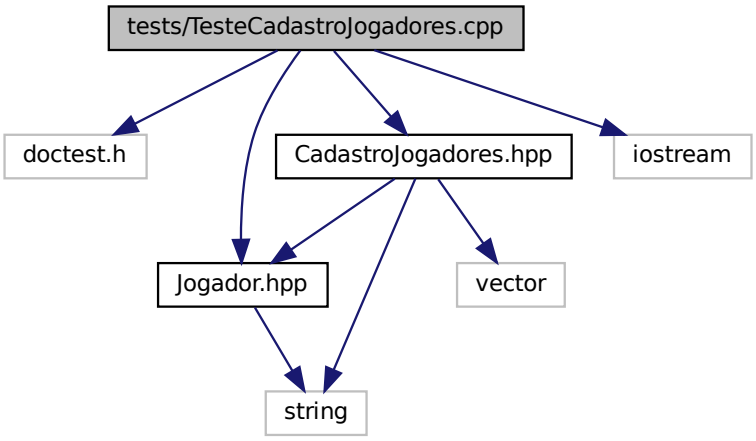
Include dependency graph for Sistema.cpp:



6.17 tests/TesteCadastroJogadores.cpp File Reference

```
#include "doctest.h"
#include "CadastroJogadores.hpp"
#include "Jogador.hpp"
#include <iostream>
```

Include dependency graph for TesteCadastroJogadores.cpp:



Functions

- TEST_CASE ("Teste do método cadastrarJogador")
- TEST_CASE ("Teste do método removerJogador")
- TEST_CASE ("Teste do método buscarJogador")
- TEST_CASE ("Teste do método listarJogadores")

6.17.1 Function Documentation

6.17.1.1 TEST_CASE() [1/4]

```
TEST_CASE (
            "Teste do método buscarJogador"  )
```

6.17.1.2 TEST_CASE() [2/4]

```
TEST_CASE (
            "Teste do método cadastrarJogador"  )
```

6.17.1.3 TEST_CASE() [3/4]

```
TEST_CASE (
            "Teste do método listarJogadores"  )
```

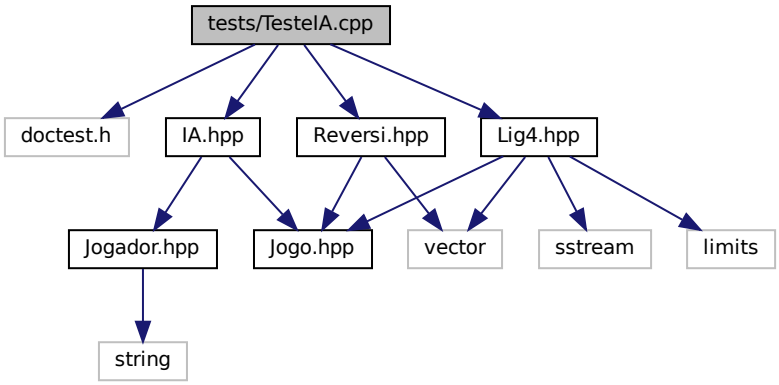
6.17.1.4 TEST_CASE() [4/4]

```
TEST_CASE (
            "Teste do método removerJogador"  )
```

6.18 tests/TesteIA.cpp File Reference

```
#include "doctest.h"
#include "IA.hpp"
#include "Reversi.hpp"
#include "Lig4.hpp"
```

Include dependency graph for TesteIA.cpp:



Macros

- #define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN

Functions

- TEST_CASE ("Teste de criação de IA")
- TEST_CASE ("Teste de cálculo de jogada pela IA")

6.18.1 Macro Definition Documentation

6.18.1.1 DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN

```
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
```

6.18.2 Function Documentation

6.18.2.1 TEST_CASE() [1/2]

```
TEST_CASE (
            "Teste de criação de IA"  )
```

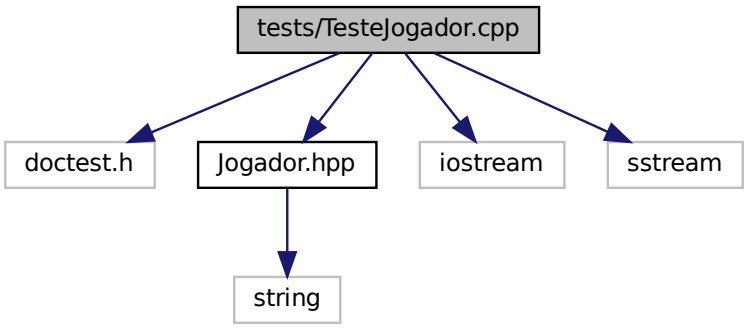
6.18.2.2 TEST_CASE() [2/2]

```
TEST_CASE (
            "Teste de cálculo de jogada pela IA"  )
```

6.19 tests/TesteJogador.cpp File Reference

```
#include "doctest.h"
#include "Jogador.hpp"
#include <iostream>
#include <sstream>
```

Include dependency graph for TesteJogador.cpp:



Functions

- TEST_CASE ("Teste do construtor e métodos de acesso")
- TEST_CASE ("Teste de adicionar vitórias e derrotas")

6.19.1 Function Documentation

6.19.1.1 TEST_CASE() [1/2]

```
TEST_CASE (
            "Teste de adicionar vitórias e derrotas"  )
```

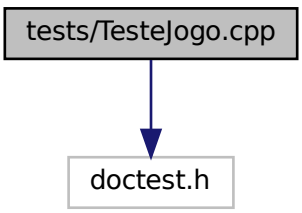
6.19.1.2 TEST_CASE() [2/2]

```
TEST_CASE (
            "Teste do construtor e métodos de acesso"  )
```

6.20 tests/TesteJogo.cpp File Reference

```
#include "doctest.h"
```

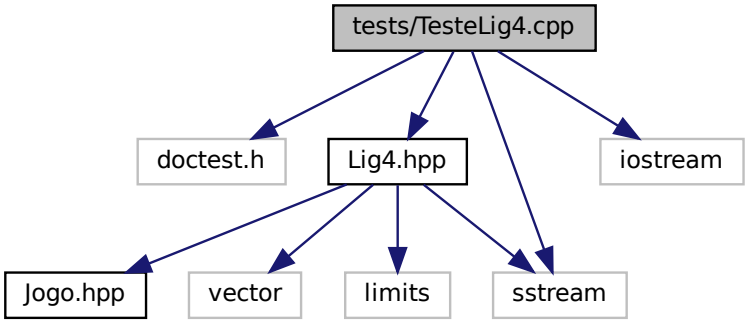
Include dependency graph for TesteJogo.cpp:



6.21 tests/TesteLig4.cpp File Reference

```
#include "doctest.h"
#include "Lig4.hpp"
#include <iostream>
#include <sstream>
```

Include dependency graph for TesteLig4.cpp:



Functions

- TEST_CASE ("Teste de iniciar e imprimir o tabuleiro")
- TEST_CASE ("Teste de jogadas válidas e alternância de jogadores")
- TEST_CASE ("Teste de jogada inválida")
- TEST_CASE ("Teste de verificação de vitória")

6.21.1 Function Documentation

6.21.1.1 TEST_CASE() [1/4]

```
TEST_CASE (
            "Teste de iniciar e imprimir o tabuleiro"  )
```

6.21.1.2 TEST_CASE() [2/4]

```
TEST_CASE (
            "Teste de jogada inválida"  )
```

6.21.1.3 TEST_CASE() [3/4]

```
TEST_CASE (
            "Teste de jogadas válidas e alternância de jogadores"  )
```

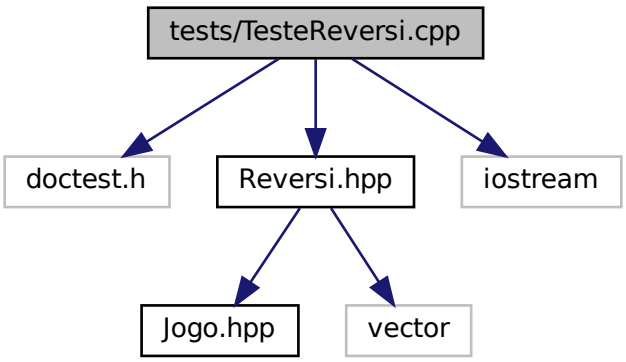
6.21.1.4 TEST_CASE() [4/4]

```
TEST_CASE (
            "Teste de verificação de vitória"  )
```

6.22 tests/TesteReversi.cpp File Reference

```
#include "doctest.h"
#include "Reversi.hpp"
#include <iostream>
```

Include dependency graph for TesteReversi.cpp:



Macros

- #define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN

Functions

- int contarPecas (const Reversi &jogo, char jogador)
- TEST_CASE ("Teste de inicialização do jogo de Reversi")
- TEST_CASE ("Teste de validação de jogadas no Reversi")
- TEST_CASE ("Teste de alternância de jogadores no Reversi")
- TEST_CASE ("Teste de verificação de vitória no Reversi")
- TEST_CASE ("Teste de realização de jogadas e virada de peças no Reversi")

6.22.1 Macro Definition Documentation

6.22.1.1 DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN

```
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
```

6.22.2 Function Documentation

6.22.2.1 contarPecas()

```
int contarPecas (
            const Reversi & jogo,
            char jogador )
```

6.22.2.2 TEST_CASE() [1/5]

```
TEST_CASE (
            "Teste de alternância de jogadores no Reversi"  )
```

6.22.2.3 TEST_CASE() [2/5]

```
TEST_CASE (
            "Teste de inicialização do jogo de Reversi"  )
```

6.22.2.4 TEST_CASE() [3/5]

```
TEST_CASE (
            "Teste de realização de jogadas e virada de peças no Reversi"  )
```

6.22.2.5 TEST_CASE() [4/5]

```
TEST_CASE (
            "Teste de validação de jogadas no Reversi"  )
```

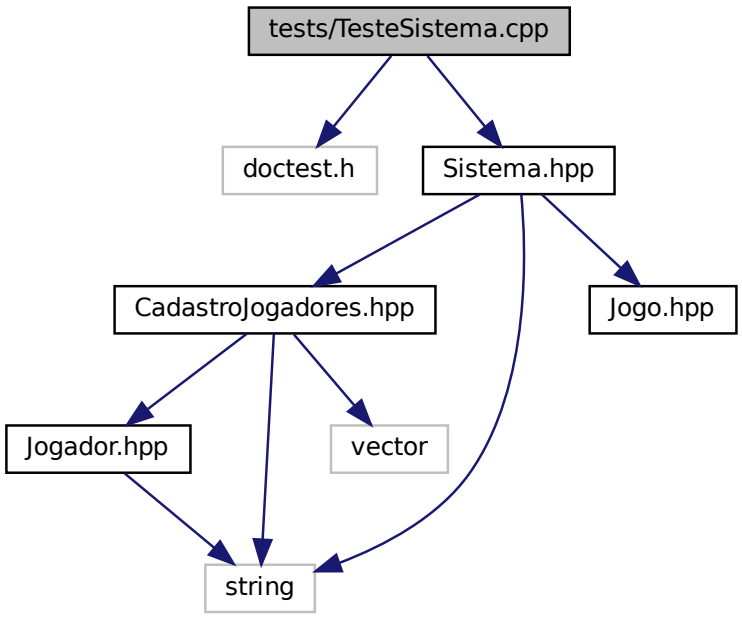
6.22.2.6 TEST_CASE() [5/5]

```
TEST_CASE (
            "Teste de verificação de vitória no Reversi"  )
```

6.23 tests/TesteSistema.cpp File Reference

```
#include "doctest.h"
#include "Sistema.hpp"
```

Include dependency graph for TesteSistema.cpp:



Macros

- #define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN

Functions

- TEST_CASE ("Teste de cadastro de jogador no sistema")
- TEST_CASE ("Teste de remoção de jogador no sistema")
- TEST_CASE ("Teste de listagem de jogadores no sistema")
- TEST_CASE ("Teste de criação de jogos no sistema")
- TEST_CASE ("Teste de execução de partida no sistema")
- int main ()

6.23.1 Macro Definition Documentation

6.23.1.1 DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN

```
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
```

6.23.2 Function Documentation

6.23.2.1 main()

```
int main ( )
```

6.23.2.2 TEST_CASE() [1/5]

```
TEST_CASE (
             "Teste de cadastro de jogador no sistema"  )
```

6.23.2.3 TEST_CASE() [2/5]

```
TEST_CASE (
             "Teste de criação de jogos no sistema"  )
```

6.23.2.4 TEST_CASE() [3/5]

```
TEST_CASE (
             "Teste de execução de partida no sistema"  )
```

6.23.2.5 TEST_CASE() [4/5]

```
TEST_CASE (
             "Teste de listagem de jogadores no sistema"  )
```

6.23.2.6 TEST_CASE() [5/5]

```
TEST_CASE (
             "Teste de remoção de jogador no sistema"  )
```

Index