Jogos de Tabuleiro

Generated by Doxygen 1.11.0

Chapter 1

Projeto Final - Jogos de tabuleiro

Este projeto implementa um conjunto de jogos de tabuleiro clássicos utilizando C++, permitindo que dois jogadores se enfrentem em uma partida. Os jogos disponíveis incluem:

- Tic Tac Toe (Jogo da Velha)
- **Reversi** (p. 28)
- Connect Four (Lig4)

Cada jogo foi estruturado fazendo uso de herança e polimorfismo para facilitar a reutilização de código e garantir uma fácil extensão para outros jogos futuros. Para mais, ainda foi desenvolvido um sistema de cadastro e gerenciamento de jogadores.

```
Clique aqui para ter acesso à documentação
```

1.1 Como iniciar o projeto?

Para iniciar o programa no seu computador, utilize os seguintes comandos:

1. Instale o software *make* caso não tenha instalado na sua máquina;
2. Em seguida, clone o repositório e o acesse pelo terminal;
3. Compile o código digitando **make** no terminal;
4. Para executar o código compilado, digite **bin/./start_game**;
5. Para executar os testes, digite **make test**;
6. Para limpar os arquivos compilados, digite **make clean**.

1.2 Estrutura do Projeto

A estrutura do projeto é modular, com classes específicas para cada jogo derivadas da classe base **Game** (p. 16). A classe **GameBoard** (p. 18) gerencia a interface com os jogadores e controla o fluxo geral do programa. Principais Arquivos:

- **Player.hpp (**p. **38)/cpp:** Armazena e gerencia quaisquer dados relacionados aos jogadores.
- **Game.hpp (**p. **37)/cpp:** Define a classe abstrata base para os jogos, incluindo a lógica comum para todos os jogos, como a criação e a impressão do tabuleiro e verificação de preenchimento do tabuleiro.
- **GameBoard.hpp (**p. **38)/cpp:** Controla o registro dos jogadores, a leitura/escrita de dados dos jogadores em arquivos, e o início das partidas entre jogadores.
- **TicTacToe.hpp (**p. **39)/cpp:** Implementa o jogo Tic Tac Toe, incluindo a lógica para verificar vitórias e processar jogadas.
- **Reversi.hpp (**p. **39)/cpp:** Implementa o jogo **Reversi** (p. 28), com métodos específicos para gerenciar as regras de captura de peças e contagem de peças no tabuleiro.
- **ConnectFour.hpp (**p. **37)/cpp:** Implementa o jogo Connect Four, com métodos específicos para gerenciar as condições de vitoria e jogada válidas

Ademais, o projeto também é composto pelos arquivos de teste que são responsáveis por testar cada classe e suas funcionalidades.

1.3 Fluxo do software

Nessa seção iremos apresentar como o software funciona para o usuário final. É importante destacar que não foi utilizada uma interface gráfica para a sua execução, sendo feita apenas o uso do terminal do sistema operacional.

1. O software inicia sua execução lendo os jogadores que foram cadastrados previamente no arquivo players.↩ txt. Caso esse arquivo ainda não exista, uma mensagem será emitida avisando sobre isso e continuará a execução do programa. Em seguida, o menu de opções será apresentado.

   Erro na leitura do arquivo: players.txt Iniciando o sistema com nenhum jogador cadastrado.

   Escolha uma opção para interagir com o tabuleiro: CJ - Cadastrar jogador RJ - Remover jogador LJ - Listar jogadores EP - Entrar na partida FS - Sair do tabuleiro

1. **CJ - Cadastrar jogador:** Essa opção é utilizada para cadastrar um jogador no sistema solicitando o apelido e o nome ao usuário. Caso esse apelido já tenha sido cadastrado, ocorrerá um erro e o programa voltará para o menu de opções.

   //Caso de sucesso CJ Informe o apelido e o nome do jogador, respectivamente: Joao Carlos Jogador Joao cadastrado com sucesso!

   //Caso de falha CJ
   Informe o apelido e o nome do jogador, respectivamente: Luiz Felipe ERRO: Jogador existente!

1. **RJ - Remover jogador:** Esta opção irá remover o jogador que possui o apelido informado. Caso este jogador não exista, aparecerá uma mensagem de erro.

//Caso de sucesso RJ Informe o apelido do jogador a ser removido: Gabriel Jogador Gabriel removido com sucesso!

//Caso de falha RJ
Informe o apelido do jogador a ser removido: Fulano ERRO: Jogador não encontrado!

1. **LJ - Listar jogadores:** Esta função irá listar os jogadores cadastrados considerando a ordem informada. É apresentado o apelido, o nome e informações de vitórias/derrotas de cada jogo da aplicação.

   LJ Informe um método de ordenação: [Apelido|Nome] TESTE ERRO: Informe um método de ordenação válido: [Apelido|Nome] nome

   Joao Carlos REVERSI - V: 0 D: 0 LIG4 - V: 0 D: 0 TICTACTOE - V: 0 D: 0

   Luiz Felipe REVERSI - V: 0 D: 0 LIG4 - V: 0 D: 0 TICTACTOE - V: 0 D: 0

1. **EP - Entrar na partida:** Esta função é utilizada para iniciar um dos três jogos. O usuário deve escolher qual jogo será executado e em seguida informar os dois jogadores que irão participar da partida.

   EP Informe o jogo: [Reversi|Lig4|TicTacToe] Minecraft ERRO:Informe um jogo válido: [Reversi|Lig4|↩ TicTacToe] Lig4 Informe o apelido do primeiro jogador: Luiz Informe o apelido do segundo jogador: Joao

1. **FS - Sair do tabuleiro:** Ao pressionar ctrl+D ou digitar FS no menu de opções, o programa será finalizado, salvando todos os jogadores no arquivo players.txt. Lembrando que se ele for finalizado de forma abrupta, com ctrl+D, uma mensagem de erro irá aparecer.

   FS Finalizando o tabuleiro...

1. **Reversi (**p. **28):** O jogo **Reversi** (p. 28) irá apresentar o seguinte tabuleiro a cada jogada realizada (linha, coluna) alternando os jogadores, mostrando o contador de peças logo abaixo. Caso seja informada uma coordenada que não obedeça as regras do jogo ou que ultrapasse o tamanho do tabuleiro, uma mensagem de erro será impressa e o jogador terá uma nova chance para jogar. Além desses casos, pode ocorrer a situação onde um jogador não tenha posições possíveis para jogar, então passará a vez. O fim da partida ocorre quando não há mais espaços a serem preenchidos no tabuleiro ou algum jogador perca todas as peças, sendo assim, o jogador com mais peças ganha. O número -1 pode ser impresso para desistência.

   1 2 3 4 5 6 7 8
   1 | | | | | | | | 2 | | | | | | | | 3 | | | | | | | | 4 | | | | X | O | | | 5 | | | | O | X | | | 6 | | | | | | | | 7 | | | | | | | |

1.3.1 8 | | | | | | | |

1.3.2 | X = 2 | O = 2 |

Luiz (X), faça sua jogada (linha e coluna):

1 2 ERRO: Não tem peça oposta adjacente. Vez de Luiz jogar(X):

1 9 ERRO: Posição fora dos limites do tabuleiro! Vez de Luiz jogar(X):

-1 Luiz desistiu, vitória de Joao.

//Vitória sem desistência Parabéns Joao! Você venceu!

//Empate O jogo terminou em empate!

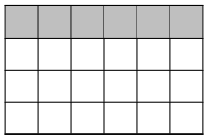
//Um jogador não possui coordenadas para jogar Nao há jogadas possíveis para o(a) jogador(a) Luiz

1. **Lig4:** O jogo será iniciado com uma lista de possíveis tamanhos para o tabuleiro na qual uma deve ser selecionada. Logo em seguida, será apresentado o tabuleiro que receberá o número de uma coluna para efetuar a jogada dos jogadores de forma alternada. Caso seja informada uma coluna que não obedeça as regras do jogo ou que ultrapasse o tamanho do tabuleiro, uma mensagem de erro será impressa e o jogador terá uma nova chance para jogar. Ganha quem formar um quarteto primeiro (vertical, horizontal ou diagonal), caso isso não aconteça, a partida terminará em empate. Para mais, a cada interação, será mostrado acima do tabuleiro a quantidade de formas que o jogador corrente pode ganhar.

   Escolha o tamanho do Tabuleiro:

   1. 5x6
   2. 6x7
   3. 7x8 Digite o número da opção desejada: 1

   Possíveis formas de ganhar para X: 0 1 2 3 4 5 6



Turno de Luiz (X): 8 ERRO:Posição fora dos limites do tabuleiro!
//Erro de coluna Coluna cheia. Tente novamente!
//Vitória Possíveis formas de ganhar para X: 3 1 2 3 4 5 6
| | | | | | | X | | | | | | | X | O | | | | | X | O | | | | |

1.3.3 | X | O | | | | |

Parabéns Luiz! Você venceu!

//Empate O jogo terminou em empate!

1. **Tic Tac Toe:** O jogo Tic Tac Toe mostrará o seguinte tabuleiro a cada jogada realizada (linha, coluna) alternando os jogadores. Caso seja informada uma coordenada que não obedeça as regras do jogo ou que ultrapasse o tamanho do tabuleiro, uma mensagem de erro será impressa e o jogador terá uma nova chance para jogar. Ganha quem fizer uma sequência de três primeiro símbolos primeiro, caso isso não aconteça, a partida terminará em empate.

   1 2 3
   1 | | | | 2 | | | |

1.3.4 3 | | | |

Luiz (X), faça sua jogada (linha e coluna):

//Erro Movimento inválido! Tente novamente.

//Vitória 1 2 3
1 | X | | X | 2 | O | O | X |

1.3.5 3 | O | | X |

Parabéns Joao! Você venceu!

//Empate O jogo terminou em empate!

1.4 Principais Desafios

Alguns dos desafios enfrentados no desenvolvimento deste projeto incluem:

- **Gerenciamento da Herança:** Garantir que a classe base **Game** (p. 16) fornecesse uma estrutura suficientemente flexível para suportar a lógica variada dos diferentes jogos.
- **Implementação da Lógica de Jogo:** Cada jogo possui regras e condições de vitória específicas, o que exigiu um cuidado especial para implementar métodos como checkWin (Tic Tac Toe)/(**ConnectFour** (p. 13)) e thereIsConnection (**Reversi** (p. 28)).
- **Persistência de Dados:** A leitura e escrita das informações dos jogadores em arquivos foi um ponto que exigiu atenção para garantir que os dados fossem armazenados corretamente entre as execuções do programa.
- **Implementação dos Testes:** Pensar em todas as possíveis formas de teste do programa foi uma tarefa árdua e que requereu uma grande quantidade de tempo.
- **Implementação das Funções Extras:** Foi particularmente difícil encontrar logicas para a contador de possíveis vitorias no **ConnectFour** (p. 13), por exemplo.
- **Vazamento de memória:** Foi necessário a utilização da ferramenta Valgrind para encontrar e resolver problemas relacionados ao acesso de memória de maneira indevida.

1.5 Funcionalidades Extras

Além das funcionalidades básicas, algumas características extras foram implementadas:

1. **Contador de Possíveis Formas de Ganhar:** No jogo **ConnectFour** (p. 13) foi criado um contador de "possíveis formas de ganhar", ele conta todas as possíveis linhas em que um jogador consegue conectar 4 peças, com seu valor sendo alterado conforme mais peças são jogadas.
2. **Escolha de Tamanho de Tabuleiro:** No jogo **ConnectFour** (p. 13) o jogador é dada a escolha de 3 diferentes tamanhos de tabuleiro para jogar.
3. **Contador de Peças:** No jogo **Reversi** (p. 28) foi criado um contador para as peças de ambos os jogadores, com o intuito de facilitar o entendimento de quem esta ganhando a partida.
4. **Tic Tac Toe:** Também foi criado um terceiro jogo extra, o clássico Tic Tac Toe ou como é conhecido no Brasil, Jogo da Velha.

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 documented files with brief descriptions:

Chapter 5

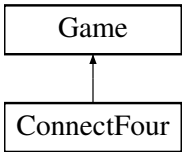
Class Documentation

5.1 ConnectFour Class Reference

Classe que representa o jogo Conecta 4.

`#include <ConnectFour.hpp>`

Inheritance diagram for ConnectFour:



Public Member Functions

- **ConnectFour** (int rows, int cols)

  *Construtor da classe **ConnectFour** (*p. *13).*
- void **countPossibleWins** (std::string currentPiece) const

  Conta o número de possíveis formas de ganhar para o jogador atual.
- bool **checkWin** (std::string currentPiece) const

  Verifica se o jogador atual ganhou.
- bool **makeMove** (int col, std::string currentPiece)

  Faz uma jogada, colocando uma peça na coluna especificada.
- void **match** (**Player** ∗player1, **Player** ∗player2) override

  Executa uma partida entre dois jogadores.

Public Member Functions inherited from Game

- **Game** (int rows, int cols)

  *Construtor da classe **Game** (*p. *16).*
- virtual ∼**Game** ()=default

  Destrutor virtual padrão.
- virtual bool **isBoardFull** () const

  Verifica se o tabuleiro está cheio.

Additional Inherited Members

Protected Attributes inherited from Game

- std::vector< std::vector< std::string > > **_board**
- int **_defaultRows**
- int **_defaultCols**

5.1.1 Detailed Description

Classe que representa o jogo Conecta 4.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 ConnectFour()

```
ConnectFour::ConnectFour (
            int rows,
            int cols)
```

Construtor da classe **ConnectFour** (p. 13).

Parameters

rows	Número de linhas do tabuleiro.
cols	Número de colunas do tabuleiro.

5.1.3 Member Function Documentation

5.1.3.1 checkWin()

```
bool ConnectFour::checkWin (
            std::string currentPiece) const
```

Verifica se o jogador atual ganhou.

Parameters

currentPiece	Representa a peça atual.

Returns

Verdadeiro se o jogador ganhou, falso caso contrário.

5.1.3.2 countPossibleWins()

```
void ConnectFour::countPossibleWins (
            std::string currentPiece) const
```

Conta o número de possíveis formas de ganhar para o jogador atual.

Parameters

currentPiece	Representa a peça atual.

Returns

Número de possíveis formas de ganhar.

5.1.3.3 makeMove()

```
bool ConnectFour::makeMove (
            int col,
            std::string currentPiece)
```

Faz uma jogada, colocando uma peça na coluna especificada.

Parameters

col	A coluna onde a peça será colocada.
currentPiece	Representa a peça atual.

Returns

Verdadeiro se a jogada for bem-sucedida, falso caso contrário.

5.1.3.4 match()

```
void ConnectFour::match (
             Player * player1,
             Player * player2)  [override], [virtual]
```

Executa uma partida entre dois jogadores.

Parameters

player1	Ponteiro para o primeiro jogador.
player2	Ponteiro para o segundo jogador.

Implements **Game** (p. 17).

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

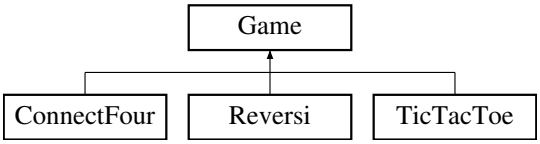
- ProjetoFinal-main/include/ConnectFour.hpp
- ProjetoFinal-main/src/ConnectFour.cpp

5.2 Game Class Reference

Classe base abstrata que representa um jogo genérico.

`#include <Game.hpp>`

Inheritance diagram for Game:



Public Member Functions

- **Game** (int rows, int cols)
  *Construtor da classe **Game** (p. 16).*
- virtual ∼**Game** ()=default
  Destrutor virtual padrão.
- virtual void **printBoard** () const
  Imprime o estado atual do tabuleiro.
- virtual bool **isBoardFull** () const
  Verifica se o tabuleiro está cheio.
- virtual void **match** (**Player** ∗player1, **Player** ∗player2)=0
  Executa uma partida entre dois jogadores.

Protected Attributes

- std::vector< std::vector< std::string > > **_board**
- int **_defaultRows**
- int **_defaultCols**

5.2.1 Detailed Description

Classe base abstrata que representa um jogo genérico.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Game()

```
Game::Game (
            int rows,
            int cols)
```

Construtor da classe **Game** (p. 16).

Parameters

rows	Número de linhas do tabuleiro.
cols	Número de colunas do tabuleiro.

Inicializa o tabuleiro com o número de linhas e colunas especificado, preenchendo-o com espaços vazios.

Parameters

rows	Número de linhas do tabuleiro.
cols	Número de colunas do tabuleiro.

5.2.3 Member Function Documentation

5.2.3.1 isBoardFull()

```
bool Game::isBoardFull () const  [virtual]
```

Verifica se o tabuleiro está cheio.

Returns

Verdadeiro se o tabuleiro estiver cheio, falso caso contrário.

Este método verifica se todas as células do tabuleiro foram preenchidas, o que indicaria que o jogo terminou em um empate, caso nenhuma condição de vitória tenha sido atendida.

Returns

Verdadeiro se o tabuleiro estiver cheio, falso caso contrário.

Reimplemented in **Reversi** (p. 31).

5.2.3.2 match()

```
virtual void Game::match (
              Player * player1,
              Player * player2)  [pure virtual]
```

Executa uma partida entre dois jogadores.

Parameters

player1	Ponteiro para o primeiro jogador.
player2	Ponteiro para o segundo jogador.

Implemented in **ConnectFour** (p. 15), **Reversi** (p. 31), and **TicTacToe** (p. 35).

5.2.3.3 printBoard()

```
void Game::printBoard () const  [virtual]
```

Imprime o estado atual do tabuleiro.

Imprime o tabuleiro no console.

Este método exibe o estado atual do tabuleiro de jogo, mostrando as linhas e colunas com os valores correspondentes em cada célula.

5.2.4 Member Data Documentation

5.2.4.1 _board

```
std::vector<std::vector<std::string> > Game::_board  [protected]
```

Matriz que representa o tabuleiro do jogo.

5.2.4.2 _defaultCols

```
int Game::_defaultCols  [protected]
```

Número padrão de colunas do tabuleiro.

5.2.4.3 _defaultRows

```
int Game::_defaultRows  [protected]
```

Número padrão de linhas do tabuleiro.

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

- ProjetoFinal-main/include/Game.hpp
- ProjetoFinal-main/src/Game.cpp

5.3 GameBoard Class Reference

Classe que gerencia o tabuleiro e os jogadores.

```
#include <GameBoard.hpp>
```

Public Member Functions

- **GameBoard** ()

  *Construtor da classe **GameBoard** (p. 18).*
- ∼**GameBoard** ()

  *Destrutor da classe **GameBoard** (p. 18).*
- bool **searchPlayer** (std::string nickName) const

  Busca um jogador pelo seu apelido.
- void **listStatistics** (std::string orderType) const

  Lista as estatísticas dos jogadores.
- size_t **getNumberOfPlayers** () const

  Retorna o número de jogadores registrados.
- void **removePlayer** (std::string nickName)

  Remove um jogador do sistema.
- void **startGame** (std::string game, std::string nickNamePlayer1, std::string nickNamePlayer2)

  Inicia um jogo entre dois jogadores.
- void **readPlayersFromFile** ()

  Lê os jogadores do arquivo e os carrega no sistema.
- void **writePlayersToFile** ()

  Escreve os jogadores registrados no arquivo.
- std::string **getPlayerNickName** ()

  Obtém o apelido de um jogador.
- void **registerPlayer** (std::string nickName, std::string name)

  Registra um novo jogador no sistema.
- void **clearPlayers** ()

  Remove todos os jogadores do sistema.

Static Public Member Functions

- static std::string **transformToLowerCase** (std::string word)

  Transforma uma string para letras minúsculas.

Static Public Attributes

- static const std::string **FILENAME** = "players.txt"

  Nome do arquivo onde as informações dos jogadores são armazenadas.

5.3.1 Detailed Description

Classe que gerencia o tabuleiro e os jogadores.

A classe **GameBoard** (p. 18) é responsável por gerenciar os jogadores, registrar e remover jogadores, iniciar jogos e manter as estatísticas. Ela também gerencia a leitura e escrita dos dados dos jogadores em arquivos.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 GameBoard()

```
GameBoard::GameBoard ()
```

Construtor da classe **GameBoard** (p. 18).

Inicializa a classe **GameBoard** (p. 18) e pode realizar operações de configuração inicial.

Inicializa a instância de **GameBoard** (p. 18) e lê os jogadores de um arquivo.

5.3.2.2 ∼GameBoard()

```
GameBoard::∼GameBoard ()
```

Destrutor da classe **GameBoard** (p. 18).

Realiza a limpeza de memória e outras operações de finalização necessárias.

Escreve os jogadores no arquivo antes de destruir a instância.

5.3.3 Member Function Documentation

5.3.3.1 clearPlayers()

```
void GameBoard::clearPlayers ()
```

Remove todos os jogadores do sistema.

Remove todos os jogadores do sistema e limpa o arquivo de jogadores.

Limpa o vetor de jogadores, removendo todas as instâncias de jogadores.

5.3.3.2 getNumberOfPlayers()

```
size_t GameBoard::getNumberOfPlayers () const
```

Retorna o número de jogadores registrados.

Returns

O número de jogadores atualmente registrados.

O número de jogadores.

5.3.3.3 getPlayerNickName()

```
std::string GameBoard::getPlayerNickName ()
```

Obtém o apelido de um jogador.

Obtém o apelido de um jogador existente a partir da entrada do usuário.

Returns

O apelido de um jogador.

O apelido de um jogador válido.

5.3.3.4 listStatistics()

```
void GameBoard::listStatistics (
            std::string orderType) const
```

Lista as estatísticas dos jogadores.

Parameters

orderType	O tipo de ordenação das estatísticas (pode ser alfabética, por número de vitórias, etc.).

As estatísticas podem ser ordenadas por apelido ou nome.

Parameters

orderType	O tipo de ordenação desejada ("apelido" ou "nome").

5.3.3.5 readPlayersFromFile()

```
void GameBoard::readPlayersFromFile ()
```

Lê os jogadores do arquivo e os carrega no sistema.

Lê os jogadores de um arquivo.

Se o arquivo não puder ser lido, o sistema será iniciado sem jogadores.

5.3.3.6 registerPlayer()

```
void GameBoard::registerPlayer (
            std::string nickName,
            std::string name)
```

Registra um novo jogador no sistema.

Parameters

nickName	O apelido do jogador.
name	O nome do jogador.
nickName	O apelido do jogador.
name	O nome real do jogador.

5.3.3.7 removePlayer()

```
void GameBoard::removePlayer (
            std::string nickName)
```

Remove um jogador do sistema.

Parameters

nickName	O apelido do jogador a ser removido.

5.3.3.8 searchPlayer()

```
bool GameBoard::searchPlayer (
            std::string nickName) const
```

Busca um jogador pelo seu apelido.

Verifica se um jogador com o apelido fornecido já está registrado.

Parameters

nickName	O apelido do jogador a ser buscado.

Returns

Verdadeiro se o jogador for encontrado, falso caso contrário.

Parameters

nickName	O apelido do jogador.

Returns

Verdadeiro se o jogador for encontrado, falso caso contrário.

5.3.3.9 startGame()

```
void GameBoard::startGame (
            std::string game,
            std::string nickNamePlayer1,
            std::string nickNamePlayer2)
```

Inicia um jogo entre dois jogadores.

Parameters

game	O nome do jogo a ser iniciado (ex.: "Reversi", "TicTacToe").
nickNamePlayer1	O apelido do primeiro jogador.
nickNamePlayer2	O apelido do segundo jogador.

O jogo pode ser "reversi", "tictactoe", ou "connect four".

Parameters

game	O nome do jogo.
nickNamePlayer1	O apelido do primeiro jogador.
nickNamePlayer2	O apelido do segundo jogador.

5.3.3.10 transformToLowerCase()

```
std::string GameBoard::transformToLowerCase (
            std::string word)  [static]
```

Transforma uma string para letras minúsculas.

Transforma uma string em letras minúsculas.

Parameters

word	A string que será transformada.

Returns

A string transformada em letras minúsculas.

Parameters

word	A palavra a ser transformada.

Returns

A palavra em letras minúsculas.

5.3.3.11 writePlayersToFile()

```
void GameBoard::writePlayersToFile ()
```

Escreve os jogadores registrados no arquivo.

Escreve os jogadores em um arquivo.

Se o arquivo não puder ser criado, um erro será exibido.

5.3.4 Member Data Documentation

5.3.4.1 FILENAME

```
const std::string GameBoard::FILENAME = "players.txt"  [static]
```

Nome do arquivo onde as informações dos jogadores são armazenadas.

Nome do arquivo para testes com jogadores.

Nome do arquivo que armazena os dados dos jogadores.

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

- ProjetoFinal-main/include/GameBoard.hpp
- ProjetoFinal-main/src/GameBoard.cpp
- ProjetoFinal-main/src/ **main.cpp**
- ProjetoFinal-main/tests/ **testGameBoardClass.cpp**

5.4 Player Class Reference

Classe que representa um jogador.

`#include <Player.hpp>`

Public Member Functions

- **Player** ()=default
  *Construtor padrão da classe **Player** (p. 24).*
- **Player** (std::string nickName, std::string name)
  Construtor que inicializa um jogador com nome e apelido.
- ~**Player** ()
  *Destrutor da classe **Player** (p. 24).*
- void **showStatistics** () const
  Exibe as estatísticas do jogador.
- void **readPlayers** (std::ifstream &in)
  Lê as informações do jogador de um arquivo.
- void **writePlayers** (std::ofstream &out) const
  Escreve as informações do jogador em um arquivo.
- std::string **getName** () const
  Obtém o nome real do jogador.
- std::string **getNickName** () const
  Obtém o apelido do jogador.
- void **setVictories** (std::string gameName)
  Incrementa o número de vitórias do jogador em um determinado jogo.
- void **setDefeats** (std::string gameName)
  Incrementa o número de derrotas do jogador em um determinado jogo.
- int **getVictories** (const std::string &gameName) const
  Obtém o número de vitórias do jogador em um determinado jogo.
- int **getDefeats** (const std::string &gameName) const
  Obtém o número de derrotas do jogador em um determinado jogo.

5.4.1 Detailed Description

Classe que representa um jogador.

A classe **Player** (p. 24) armazena as informações de um jogador, incluindo seu nome, apelido, e estatísticas de vitórias e derrotas em diferentes jogos.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Player() [1/2]

`Player::Player () [default]`

Construtor padrão da classe **Player** (p. 24).

Inicializa um jogador com valores padrão.

5.4.2.2 Player() [2/2]

```
Player::Player (
            std::string nickName,
            std::string name)
```

Construtor que inicializa um jogador com nome e apelido.

Construtor da classe **Player** (p. 24).

Parameters

nickName	O apelido do jogador.
name	O nome real do jogador.

Inicializa um jogador com seu apelido e nome, e define as estatísticas de vitórias e derrotas para cada jogo (**Reversi** (p. 28), Connect Four, **TicTacToe** (p. 33)) como zero.

Parameters

nickName	Apelido do jogador.
name	Nome do jogador.

5.4.2.3 ~Player()

```
Player::~Player ()
```

Destrutor da classe **Player** (p. 24).

Realiza a limpeza de recursos utilizados pelo jogador, se necessário.

Destrói o objeto **Player** (p. 24).

5.4.3 Member Function Documentation

5.4.3.1 getDefeats()

```
int Player::getDefeats (
            const std::string & gameName) const  [inline]
```

Obtém o número de derrotas do jogador em um determinado jogo.

Parameters

gameName	O nome do jogo.

Returns

O número de derrotas no jogo especificado.

5.4.3.2 getName()

```
std::string Player::getName () const
```

Obtém o nome real do jogador.

Obtém o nome do jogador.

Returns

O nome do jogador.

std::string Retorna o nome do jogador.

5.4.3.3 getNickName()

```
std::string Player::getNickName () const
```

Obtém o apelido do jogador.

Returns

O apelido do jogador.

std::string Retorna o apelido do jogador.

5.4.3.4 getVictories()

```
int Player::getVictories (
            const std::string & gameName) const  [inline]
```

Obtém o número de vitórias do jogador em um determinado jogo.

Parameters

gameName	O nome do jogo.

Returns

O número de vitórias no jogo especificado.

5.4.3.5 readPlayers()

```
void Player::readPlayers (
            std::ifstream & in)
```

Lê as informações do jogador de um arquivo.

Lê os dados dos jogadores de um arquivo.

Parameters

in	O fluxo de entrada de arquivo de onde as informações serão lidas.

Carrega o apelido, nome, vitórias e derrotas do jogador a partir de um arquivo de entrada.

Parameters

in	Stream de entrada do arquivo.

5.4.3.6 setDefeats()

```
void Player::setDefeats (
            std::string gameName)
```

Incrementa o número de derrotas do jogador em um determinado jogo.

Incrementa o número de derrotas para um jogo específico.

Parameters

gameName	O nome do jogo em que a derrota ocorreu.

Aumenta o contador de derrotas do jogador para o jogo especificado.

Parameters

gameName	Nome do jogo (ex: "reversi", "connectFour", "ticTacToe").

5.4.3.7 setVictories()

```
void Player::setVictories (
            std::string gameName)
```

Incrementa o número de vitórias do jogador em um determinado jogo.

Incrementa o número de vitórias para um jogo específico.

Parameters

gameName	O nome do jogo em que a vitória ocorreu.

Aumenta o contador de vitórias do jogador para o jogo especificado.

Parameters

gameName	Nome do jogo (ex: "reversi", "connectFour", "ticTacToe").

5.4.3.8 showStatistics()

```
void Player::showStatistics () const
```

Exibe as estatísticas do jogador.

Mostra o número de vitórias e derrotas do jogador em cada jogo.

Mostra o apelido, nome, número de vitórias e derrotas para cada jogo.

5.4.3.9 writePlayers()

```
void Player::writePlayers (
            std::ofstream & out) const
```

Escreve as informações do jogador em um arquivo.

Escreve os dados dos jogadores em um arquivo.

Parameters

out	O fluxo de saída de arquivo para onde as informações serão escritas.

Salva o apelido, nome, vitórias e derrotas do jogador em um arquivo de saída.

Parameters

out	Stream de saída do arquivo.

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

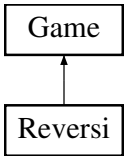
- ProjetoFinal-main/include/Player.hpp
- ProjetoFinal-main/src/ **Player.cpp**

5.5 Reversi Class Reference

Essa classe representa o jogo **Reversi** (p. 28).

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

Inheritance diagram for Reversi:



Public Member Functions

- **Reversi** (int rows, int cols)

  *Constroi um jogo **Reversi** (p. 28) com as dimenções do tabuleiro especificadas.*
- std::string **getBoardContent** (int row, int col) const

  Retorna o conteúdo da posição especificada no tabuleiro.
- bool **thereIsNearby** (int rows, int cols, std::string watching)

  Verifica se tem uma peça oposta em posições adjacentes.
- bool **thereIsConnection** (int rows, int cols, std::string turn, std::string watching)

  Verifica se existe conexão entre as peças.
- void **piecesCounter** ()

  Atualiza o contador de peças X e O.
- bool **isAnyPossiblePlay** (std::string turn, std::string watching)

  Checa se existe alguma jogada possível.
- bool **isBoardFull** () const override

  Verifica se o tabuleiro está cheio.
- void **makeMove** (int rows, int cols, std::string turn, std::string watching)

  Realiza uma jogada no tabuleiro.
- void **match** (**Player** ∗player1, **Player** ∗player2) override

  Começa uma partida entre dois jogadores.

Public Member Functions inherited from Game

- **Game** (int rows, int cols)

  *Construtor da classe **Game** (p. 16).*
- virtual ∼**Game** ()=default

  Destrutor virtual padrão.
- virtual void **printBoard** () const

  Imprime o estado atual do tabuleiro.

Additional Inherited Members

Protected Attributes inherited from Game

- std::vector< std::vector< std::string > > **_board**
- int **_defaultRows**
- int **_defaultCols**

5.5.1 Detailed Description

Essa classe representa o jogo **Reversi** (p. 28).

Essa classe é derivada da classe "Game" e implementa métodos, regras e lógicas específicas do jogo **Reversi** (p. 28)

5.5.2 Constructor & Destructor Documentation

5.5.2.1 Reversi()

```
Reversi::Reversi (
            int rows,
            int cols)
```

Constroi um jogo **Reversi** (p. 28) com as dimenções do tabuleiro específicadas.

Construtor da classe **Reversi** (p. 28).

Parameters

rows	Número de linhas do tabuleiro.
cols	Número de colunas do tabuleiro.

5.5.3 Member Function Documentation

5.5.3.1 getBoardContent()

```
std::string Reversi::getBoardContent (
            int row,
            int col) const
```

Retorna o conteúdo da posição especificada no tabuleiro.

Parameters

row	A linha da posição.
col	A coluna da posição.

Returns

O conteúdo da posição no tabuleiro.

O método é usado nos casos de teste do **Reversi** (p. 28) para verificar jogadas específicas.

Parameters

row	Número de linhas do tabuleiro.
col	Número de colunas do tabuleiro.

Returns

O conteúdo da posição no tabuleiro.

5.5.3.2 isAnyPossiblePlay()

```
bool Reversi::isAnyPossiblePlay (
            std::string turn,
            std::string watching)
```

Checa se existe alguma jogada possível.

Verifica se há alguma jogada possível.

Parameters

turn	Representa a peça do jogador.
watching	Representa a peça do outro jogador.

Returns

true se tem alguma jogada possível, false se não tem.

Parameters

turn	Peça do jogador atual.
watching	Peça do outro jogador.

Returns

true se há jogadas possíveis, false caso contrário.

5.5.3.3 isBoardFull()

```
bool Reversi::isBoardFull () const  [override], [virtual]
```

Verifica se o tabuleiro está cheio.

Returns

true se o tabuleiro está cheio, false se não estiver.

true se o tabuleiro está cheio, false caso contrário.

Reimplemented from **Game** (p. 17).

5.5.3.4 makeMove()

```
void Reversi::makeMove (
            int rows,
            int cols,
            std::string turn,
            std::string watching)
```

Realiza uma jogada no tabuleiro.

Parameters

rows	A linha de sua jogada.
cols	A coluna de sua jogada.
turn	Representa a peça do jogador.
watching	Representa a peça do outro jogador.
rows	Linha onde a peça será colocada.
cols	Coluna onde a peça será colocada.
turn	Peça do jogador atual.
watching	Peça do outro jogador.

5.5.3.5 match()

```
void Reversi::match (
            Player * player1,
            Player * player2)  [override], [virtual]
```

Começa uma partida entre dois jogadores.

Realiza uma partida de **Reversi** (p. 28) entre dois jogadores.

Parameters

player1	Ponteiro para o primeiro jogador.
player2	Ponteiro para o segundo jogador.
player1	Ponteiro para o jogador 1.
player2	Ponteiro para o jogador 2.

Implements **Game** (p. 17).

5.5.3.6 piecesCounter()

```
void Reversi::piecesCounter ()
```

Atualiza o contador de peças X e O.

Conta o número de peças de cada jogador no tabuleiro e mostra.

5.5.3.7 thereIsConnection()

```
bool Reversi::thereIsConnection (
            int rows,
            int cols,
            std::string turn,
            std::string watching)
```

Verifica se existe conexão entre as peças.

Verifica se há uma conexão entre uma posição e outra peça do mesmo tipo.

Parameters

rows	A linha da peça que precisa ser verificada.
cols	A coluna da peça que precisa ser verificada.
turn	Representa a peça do jogador.
watching	Representa a peça do outro jogador.

Returns

True se tem uma conexão, false se não tem.

Parameters

rows	Linha da posição a ser verificada.
cols	Coluna da posição a ser verificada.
turn	Peça do jogador atual.
watching	Peça do outro jogador.

Returns

true se há conexão, false caso contrário.

5.5.3.8 thereIsNearby()

```
bool Reversi::thereIsNearby (
            int rows,
            int cols,
            std::string watching)
```

Verifica se tem uma peça oposta em posições adjacentes.

Verifica se há uma peça do oponente adjacente a uma posição.

Parameters

rows	A linha onde a peça vai ser jogada após a verificação.
cols	A coluna onde a peça vai ser jogada após a verificação.
watching	Representa a peça do outro jogador.

Returns

true se tem uma peça oposta em posição adjacente, false se não tem.

Parameters

rows	Linha da posição a ser verificada.
cols	Coluna da posição a ser verificada.
watching	Peça do outro jogador.

Returns

true se há uma peça do oponente adjacente, false caso contrário.

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

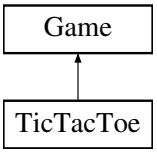
- ProjetoFinal-main/include/Reversi.hpp
- ProjetoFinal-main/src/ **Reversi.cpp**

5.6 TicTacToe Class Reference

Classe que representa o jogo Tic Tac Toe (Jogo da Velha).

`#include <TicTacToe.hpp>`

Inheritance diagram for TicTacToe:



Public Member Functions

- **TicTacToe** (int rows, int cols)

  *Construtor da classe **TicTacToe** (p. 33). Inicializa um tabuleiro 3x3 para o jogo.*
- bool **checkWin** () const

  Verifica se há um vencedor no jogo.
- bool **makeMove** (int row, int col, const std::string &symbol)

  Executa uma jogada no tabuleiro.
- void **match** (**Player** *player1, **Player** *player2) override

  Executa uma partida de Tic Tac Toe entre dois jogadores.

Public Member Functions inherited from Game

- **Game** (int rows, int cols)

  *Construtor da classe **Game** (p. 16).*
- virtual ~**Game** ()=default

  Destrutor virtual padrão.
- virtual void **printBoard** () const

  Imprime o estado atual do tabuleiro.
- virtual bool **isBoardFull** () const

  Verifica se o tabuleiro está cheio.

Additional Inherited Members

Protected Attributes inherited from Game

- std::vector< std::vector< std::string > > **_board**
- int **_defaultRows**
- int **_defaultCols**

5.6.1 Detailed Description

Classe que representa o jogo Tic Tac Toe (Jogo da Velha).

5.6.2 Constructor & Destructor Documentation

5.6.2.1 TicTacToe()

```
TicTacToe::TicTacToe (
            int rows,
            int cols)
```

Construtor da classe **TicTacToe** (p. 33). Inicializa um tabuleiro 3x3 para o jogo.

Construtor da classe **TicTacToe** (p. 33).

Parameters

rows	A linha de sua jogada.
cols	A coluna de sua jogada.

5.6.3 Member Function Documentation

5.6.3.1 checkWin()

```
bool TicTacToe::checkWin () const
```

Verifica se há um vencedor no jogo.

Returns

Verdadeiro se houver um vencedor, falso caso contrário.

5.6.3.2 makeMove()

```
bool TicTacToe::makeMove (
            int row,
            int col,
            const std::string & symbol)
```

Executa uma jogada no tabuleiro.

Parameters

row	Linha da jogada.
col	Coluna da jogada.
symbol	Símbolo do jogador.

Returns

Verdadeiro se a jogada for válida, falso caso contrário.

5.6.3.3 match()

```
void TicTacToe::match (
            Player * player1,
            Player * player2)  [override], [virtual]
```

Executa uma partida de Tic Tac Toe entre dois jogadores.

Parameters

player1	Ponteiro para o primeiro jogador.
player2	Ponteiro para o segundo jogador.

Implements **Game** (p. 17).

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

- ProjetoFinal-main/include/TicTacToe.hpp
- ProjetoFinal-main/src/TicTacToe.cpp

Chapter 6

File Documentation

6.1 ConnectFour.hpp

```
00001 #ifndef CONNECTFOUR_HPP
00002 #define CONNECTFOUR_HPP
00003
00004 #include "Game.hpp"
00005 #include <vector>
00006 #include <string>
00007
00011 class ConnectFour : public Game {
00012   private:
00013
00019     bool checkHorizontal(std::string currentPiece) const;
00020
00026     bool checkVertical(std::string currentPiece) const;
00027
00033     bool checkDiagonal(std::string currentPiece) const;
00034
00040     bool isValidMove(int col) const;
00041
00042
00046     void printBoard() const override;
00047
00048
00049   public:
00055     ConnectFour(int rows, int cols);
00056
00062     void countPossibleWins(std::string currentPiece) const;
00063
00069     bool checkWin(std::string currentPiece) const;
00070
00077     bool makeMove(int col, std::string currentPiece);
00078
00079
00085     void match(Player* player1, Player* player2) override;
00086 };
00087
00088 #endif
```

6.2 Game.hpp

```
00001 #ifndef GAME_HPP
00002 #define GAME_HPP
00003
00004 #include "Player.hpp"
00005 #include <vector>
00006 #include <string>
00007
00011 class Game {
00012   protected:
00013     std::vector<std::vector<std::string» _board;
00014     int _defaultRows;
00015     int _defaultCols;
00018   public:
00024     Game(int rows, int cols);
00025
```

```
00029     virtual ~Game() = default;
00030
00034     virtual void printBoard() const;
00035
00040     virtual bool isBoardFull() const;
00041
00047     virtual void match(Player* player1, Player* player2) = 0;
00048 };
00049
00050 #endif
```

6.3 GameBoard.hpp

```
00001 #ifndef GAMEBOARD_H
00002 #define GAMEBOARD_H
00003
00004 #include "Player.hpp"
00005 #include "ConnectFour.hpp"
00006 #include "Reversi.hpp"
00007 #include "TicTacToe.hpp"
00008 #include <string>
00009 #include <vector>
00010
00018 class GameBoard {
00019   private:
00023     std::vector<Player*> _players;
00024
00025   public:
00031     GameBoard();
00032
00038     ~GameBoard();
00039
00043     static const std::string FILENAME;
00044
00051     static std::string transformToLowerCase(std::string word);
00052
00059     bool searchPlayer(std::string nickName) const;
00060
00066     void listStatistics(std::string orderType) const;
00067
00073     size_t getNumberOfPlayers() const;
00074
00080     void removePlayer(std::string nickName);
00081
00089     void startGame(
00090       std::string game,
00091       std::string nickNamePlayer1,
00092       std::string nickNamePlayer2
00093     );
00094
00098     void readPlayersFromFile();
00099
00103     void writePlayersToFile();
00104
00110     std::string getPlayerNickName();
00111
00118     void registerPlayer(std::string nickName, std::string name);
00119
00125     void clearPlayers();
00126 };
00127
00128 #endif
```

6.4 Player.hpp

```
00001 #ifndef PLAYER_HPP
00002 #define PLAYER_HPP
00003
00004 #include <string>
00005 #include <fstream>
00006 #include <map>
00007
00014 class Player {
00015   private:
00019     std::string _nickName;
00020
00024     std::string _name;
00025
00029     std::map<std::string, int> _victories;
```

```
00030
00034     std::map<std::string, int> _defeats;
00035
00036   public:
00042     Player() = default;
00043
00050     Player(std::string nickName, std::string name);
00051
00057     ~Player();
00058
00064     void showStatistics() const;
00065
00071     void readPlayers(std::ifstream& in);
00072
00078     void writePlayers(std::ofstream& out) const;
00079
00085     std::string getName() const;
00086
00092     std::string getNickName() const;
00093
00099     void setVictories(std::string gameName);
00100
00106     void setDefeats(std::string gameName);
00107
00114     int getVictories(const std::string& gameName) const { return _victories.at(gameName); }
00115
00122     int getDefeats(const std::string& gameName) const { return _defeats.at(gameName); }
00123 };
00124
00125 #endif
```

6.5 Reversi.hpp

```
00001 #ifndef REVERSI_HPP
00002 #define REVERSI_HPP
00003
00004 #include <vector>
00005 #include <string>
00006 #include "Game.hpp"
00007
00014 class Reversi : public Game {
00015   private:
00016     int _xCounter = 2;
00017     int _oCounter = 2;
00018
00019   public:
00026     Reversi(int rows, int cols);
00027
00035     std::string getBoardContent(int row, int col) const;
00036
00045     bool thereIsNearby(int rows, int cols, std::string watching);
00046
00056     bool thereIsConnection(int rows, int cols, std::string turn, std::string watching);
00057
00061     void piecesCounter();
00062
00063
00070     bool isAnyPossiblePlay(std::string turn, std::string watching);
00071
00077     bool isBoardFull() const override;
00078
00079
00088     void makeMove(int rows, int cols, std::string turn, std::string watching);
00089
00090
00097     void match(Player* player1, Player* player2) override;
00098 };
00099
00100 #endif
```

6.6 TicTacToe.hpp

```
00001 #ifndef TICTACTOE_HPP
00002 #define TICTACTOE_HPP
00003
00004 #include "Game.hpp"
00005
00009 class TicTacToe : public Game {
00010   public:
```

```
00017     TicTacToe(int rows, int cols);
00018
00023     bool checkWin() const;
00024
00032     bool makeMove(int row, int col, const std::string& symbol);
00033
00039     void match(Player* player1, Player* player2) override;
00040
00041 };
00042
00043 #endif
```

6.7 ProjetoFinal-main/src/main.cpp File Reference

Arquivo principal para a execução do jogo com o tabuleiro.

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

Functions

- int **main** ()

  Função principal que executa o menu do jogo.

6.7.1 Detailed Description

Arquivo principal para a execução do jogo com o tabuleiro.

6.7.2 Function Documentation

6.7.2.1 main()

```
int main ()
```

Função principal que executa o menu do jogo.

Esta função permite que o usuário interaja com o sistema de tabuleiro, cadastrando jogadores, removendo jogadores, listando estatísticas, e iniciando partidas dos jogos **Reversi** (p. 28), Connect Four (Lig4) e **TicTacToe** (p. 33).

Returns

int Retorna 0 ao final da execução.

6.8 ProjetoFinal-main/src/Player.cpp File Reference

Implementação da classe **Player** (p. 24) que representa um jogador.

```
#include "Player.hpp"
#include <iostream>
#include <string>
```

6.8.1 Detailed Description

Implementação da classe **Player** (p. 24) que representa um jogador.

6.9 ProjetoFinal-main/src/Reversi.cpp File Reference

Implementação da classe **Reversi** (p. 28) para o jogo **Reversi** (p. 28).

```
#include "Game.hpp"
#include "Reversi.hpp"
#include <iostream>
#include <limits>
#include <string>
#include <stdexcept>
```

6.9.1 Detailed Description

Implementação da classe **Reversi** (p. 28) para o jogo **Reversi** (p. 28).

6.10 ProjetoFinal-main/tests/testConnectFourClass.cpp File Reference

Testes para a classe **ConnectFour** (p. 13) utilizando a biblioteca doctest.

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

Functions

- **TEST_CASE** ("Test valid and invalid movements in ConnectFour")

  *Testa movimentos válidos e inválidos no jogo **ConnectFour** (p. 13).*
- **TEST_CASE** ("Test horizontal and vertical win check")

  *Testa a verificação de vitória horizontal e vertical no jogo **ConnectFour** (p. 13).*
- **TEST_CASE** ("Test the diagonal win check")

  *Testa a verificação de vitória diagonal no jogo **ConnectFour** (p. 13).*
- **TEST_CASE** ("Test draw")

  *Testa a detecção de empate no jogo **ConnectFour** (p. 13).*
- **TEST_CASE** ("Test countPossibleWins method")

  *Testa o método countPossibleWins da classe **ConnectFour** (p. 13).*

6.10.1 Detailed Description

Testes para a classe **ConnectFour** (p. 13) utilizando a biblioteca doctest.

6.11 ProjetoFinal-main/tests/testGameBoardClass.cpp File Reference

Testes para a classe **GameBoard** (p. 18) utilizando a biblioteca doctest.

```
#include <doctest.h>
#include "GameBoard.hpp"
#include "Player.hpp"
```

Functions

- **TEST_CASE** ("Testing **GameBoard** class")
  *Testa a classe **GameBoard** (p. 18) e seus métodos principais.*
- **TEST_CASE** ("Test getPlayerNickName method")
  *Testa o método getPlayerNickName da classe **GameBoard** (p. 18).*
- **TEST_CASE** ("Test listStatistics method")
  *Testa o método listStatistics da classe **GameBoard** (p. 18).*
- **TEST_CASE** ("Test writePlayersToFile method")
  *Testa o método writePlayersToFile da classe **GameBoard** (p. 18).*
- **TEST_CASE** ("Test readPlayersFromFile method")
  *Testa o método readPlayersFromFile da classe **GameBoard** (p. 18).*

6.11.1 Detailed Description

Testes para a classe **GameBoard** (p. 18) utilizando a biblioteca doctest.

6.11.2 Function Documentation

6.11.2.1 TEST_CASE() [1/3]

```
TEST_CASE (
            "Test getPlayerNickName method" )
```

Testa o método getPlayerNickName da classe **GameBoard** (p. 18).

Testa a entrada válida de um jogador existente.

Testa entrada inválida e repetição longa.

6.11.2.2 TEST_CASE() [2/3]

```
TEST_CASE (
            "Test listStatistics method" )
```

Testa o método listStatistics da classe **GameBoard** (p. 18).

Testa listStatistics com ordenação por apelido.

Testa listStatistics com ordenação por nome.

6.11.2.3 TEST_CASE() [3/3]

```
TEST_CASE (
            "Testing  GameBoard class" )
```

Testa a classe **GameBoard** (p. 18) e seus métodos principais.

Testa o método transformToLowerCase da classe **GameBoard** (p. 18).

Testa o registro e a pesquisa de jogadores.

Testa o método removePlayer da classe **GameBoard** (p. 18).

Testa o método getNumberOfPlayers da classe **GameBoard** (p. 18).

6.12 ProjetoFinal-main/tests/testGameClass.cpp File Reference

Testes para a classe **Game** (p. 16) utilizando a biblioteca doctest.

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

Functions

- **TEST_CASE** ("Test **Game** constructor")

  *Testa o método construtor da classe **Game** (p. 16).*
- **TEST_CASE** ("Test printBoard method")

  *Testa o método printBoard da classe **Game** (p. 16).*
- **TEST_CASE** ("Test isBoardFull method")

  *Testa o método isBoardFull da classe **Game** (p. 16).*

6.12.1 Detailed Description

Testes para a classe **Game** (p. 16) utilizando a biblioteca doctest.

6.13 ProjetoFinal-main/tests/testPlayerClass.cpp File Reference

Testes para a classe **Player** (p. 24) utilizando a biblioteca doctest.

```
#include "doctest.h"
#include "Player.hpp"
#include "GameBoard.hpp"
#include <iostream>
#include <sstream>
#include <fstream>
```

Functions

- **TEST_CASE** ("Testing **Player** class")

  *Testa a classe **Player** (p. 24) e seus métodos principais.*

6.13.1 Detailed Description

Testes para a classe **Player** (p. 24) utilizando a biblioteca doctest.

6.13.2 Function Documentation

6.13.2.1 TEST_CASE()

```
TEST_CASE (
            "Testing  Player class" )
```

Testa a classe **Player** (p. 24) e seus métodos principais.

Testa o construtor da classe **Player** (p. 24).

Testa as estatísticas iniciais do jogador.

Testa os métodos setVictories e setDefeats.

Testa os métodos readPlayers e writePlayers.

6.14 ProjetoFinal-main/tests/testReversiClass.cpp File Reference

Testes para a classe **Reversi** (p. 28) utilizando a biblioteca doctest.

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

Functions

- **TEST_CASE** ("Test the construction and initial state of the board")

  *Testa o método construtor da classe **Reversi** (p. 28).*
- **TEST_CASE** ("Test thereIsNearby method")

  *Testa o método thereIsNearby da classe **Reversi** (p. 28).*
- **TEST_CASE** ("Test thereIsConnection method")

  *Testa o método thereIsConnection da classe **Reversi** (p. 28).*
- **TEST_CASE** ("Test makeMove method)")

  *Testa o método makeMove da classe **Reversi** (p. 28).*
- **TEST_CASE** ("Test piecesCounter method")

  *Testa o método piecesCounter da classe **Reversi** (p. 28).*
- **TEST_CASE** ("Test isBoardFull method")

  *Testa o método isBoardFull da classe **Reversi** (p. 28).*
- **TEST_CASE** ("Test isAnyPossiblePlay method")

  *Testa o método isAnyPossiblePlay da classe **Reversi** (p. 28).*

6.14.1 Detailed Description

Testes para a classe **Reversi** (p. 28) utilizando a biblioteca doctest.

6.15 ProjetoFinal-main/tests/testTicTacToeClass.cpp File Reference

Testes para a classe **TicTacToe** (p. 33) utilizando a biblioteca doctest.

```
#include <doctest.h>
#include "TicTacToe.hpp"
#include "Game.hpp"
```

Functions

- **TEST_CASE** ("Test valid and invalid moves in TicTacToe")
  *Testa movimentos válidos e inválidos no jogo **TicTacToe** (p. 33).*
- **TEST_CASE** ("TicTacToe tie check test")
  *Testa a verificação de empate no jogo **TicTacToe** (p. 33).*
- **TEST_CASE** ("TicTacToe win check test")
  *Testa a verificação de vitória no jogo **TicTacToe** (p. 33).*

6.15.1 Detailed Description

Testes para a classe **TicTacToe** (p. 33) utilizando a biblioteca doctest.

6.15.2 Function Documentation

6.15.2.1 TEST_CASE() [1/2]

```
TEST_CASE (
            "Test valid and invalid moves in TicTacToe" )
```

Testa movimentos válidos e inválidos no jogo **TicTacToe** (p. 33).

Testa um movimento válido no jogo **TicTacToe** (p. 33).

Testa movimentos inválidos fora do tabuleiro.

Testa movimentos inválidos em uma posição já ocupada.

6.15.2.2 TEST_CASE() [2/2]

```
TEST_CASE (
            "TicTacToe win check test" )
```

Testa a verificação de vitória no jogo **TicTacToe** (p. 33).

Testa a vitória por linha.

Testa a vitória por coluna.

Testa a vitória pela diagonal principal.

Testa a vitória pela diagonal secundária.

Testa a ausência de vitória.

Index