PDS II Final Project

Gerado por Doxygen 1.9.8

Capítulo 1

Índice Hierárquico

1.1 Hierarquia de Classes

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

Capítulo 2

Índice dos Componentes

2.1 Lista de Classes

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

Capítulo 3

Índice dos Arquivos

3.1 Lista de Arquivos

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

Capítulo 4

Classes

4.1 Referência da Classe Board

Gerencia o tabuleiro do jogo.

`#include <Board.hpp>`

Membros Públicos

- Board (int num_rows_received, int num_columns_received)
  Constrói o tabuleiro com o número de linhas e colunas especificado.
- void set_space (int row, int column, char value)
  Define um valor em uma posição específica do tabuleiro.
- char get_space (int row, int column) const
  Retorna o valor de uma posição específica do tabuleiro.
- void print_game_board () const
  Imprime o estado atual do tabuleiro.
- bool is_move_inside_board (int x, int y) const
  Verifica se uma posição está dentro dos limites do tabuleiro.
- bool is_space_free (int x, int y) const
  Verifica se uma posição no tabuleiro está vazia.

Atributos Privados

- int num_rows
- int num_columns
- std::unique_ptr< std::unique_ptr< char[ ]>[ ]> game_board = nullptr

4.1.1 Descrição detalhada

Gerencia o tabuleiro do jogo.

Oferece métodos para configurar e acessar espaços no tabuleiro, além de verificar a validade de movimentos.

Definição na linha 20 do arquivo Board.hpp.

4.1.2 Construtores e Destrutores

4.1.2.1 Board()

```
Board::Board (
            int num_rows_received,
            int num_columns_received )
```

Constrói o tabuleiro com o número de linhas e colunas especificado.

Construtor que inicializa o tabuleiro com as dimensões recebidas e espaços vazios.

Parâmetros

num_rows_received	Número de linhas.
num_columns_received	Número de colunas.

Definição na linha 18 do arquivo Board.cpp.

4.1.3 Documentação das funções

4.1.3.1 get_space()

```
char Board::get_space (
            int row,
            int column ) const
```

Retorna o valor de uma posição específica do tabuleiro.

Parâmetros

row	Linha do tabuleiro.
column	Coluna do tabuleiro.

Retorna

O valor presente na posição.

Definição na linha 10 do arquivo Board.cpp.

4.1.3.2 is_move_inside_board()

```
bool Board::is_move_inside_board (
            int x,
            int y ) const
```

Verifica se uma posição está dentro dos limites do tabuleiro.

Parâmetros

x	Coordenada da linha.
y	Coordenada da coluna.

Retorna

`true` se a posição está dentro dos limites, `false` caso contrário.

Definição na linha 44 do arquivo Board.cpp.

4.1.3.3 is_space_free()

```
bool Board::is_space_free (
            int x,
            int y ) const
```

Verifica se uma posição no tabuleiro está vazia.

Parâmetros

x	Coordenada da linha.
y	Coordenada da coluna.

Retorna

`true` se a posição está vazia, `false` caso contrário.

Definição na linha 53 do arquivo Board.cpp.

4.1.3.4 print_game_board()

```
void Board::print_game_board ( ) const
```

Imprime o estado atual do tabuleiro.

Definição na linha 32 do arquivo Board.cpp.

4.1.3.5 set_space()

```
void Board::set_space (
            int row,
            int column,
            char value )
```

Define um valor em uma posição específica do tabuleiro.

Parâmetros

row	Linha do tabuleiro.
column	Coluna do tabuleiro.
value	Valor a ser colocado na posição.

Definição na linha 4 do arquivo Board.cpp.

4.1.4 Atributos

4.1.4.1 game_board

```
std::unique_ptr<std::unique_ptr<char[]>[]> Board::game_board = nullptr  [private]
```

Estrutura que armazena o estado do tabuleiro.

Definição na linha 24 do arquivo Board.hpp.

4.1.4.2 num_columns

```
int Board::num_columns  [private]
```

Número de colunas do tabuleiro.

Definição na linha 23 do arquivo Board.hpp.

4.1.4.3 num_rows

```
int Board::num_rows  [private]
```

Número de linhas do tabuleiro.

Definição na linha 22 do arquivo Board.hpp.

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

- include/Board.hpp
- src/Board.cpp

4.2 Referência da Classe Connect4

Gerencia as regras e funcionalidades do jogo Connect4.

```
#include <Connect4.hpp>
```

Diagrama de hierarquia da classe Connect4:

Diagrama de colaboração para Connect4:

Membros Públicos

- Connect4 ()

  Construtor padrão do Connect4, inicializa o jogo com 6 linhas, 7 colunas e o jogador 'X' como inicial.
- bool is_valid_move () const override

  Função declarada somente para fins de sobregarga.
- void make_move () override

  Função declarada somente para fins de sobregarga.
- bool check_win () override

  Verifica em todas direções válidas se o jogador atual venceu o jogo.
- bool is_valid_move (int column)

  Verifica se a jogada é válida para uma coluna específica.
- void make_move (int column)

  Realiza a jogada na coluna especificada.
- char get_current_player ()

  Retorna o jogador atual.
- bool is_board_full () const

  Verifica se o tabuleiro está completamente cheio.
- void print_game_board () const

  Imprime o estado atual do tabuleiro.
- void set_current_player (char player)

  Define o jogador atual.
- ~Connect4 ()

  Destrutor do jogo Connect4.

Membros Públicos herdados de Game

- Game (int num_rows_received, int num_columns_received)

  Constrói um jogo com um tabuleiro de tamanho especificado.
- char switch_players (char current_player)

  Alterna entre os jogadores.
- ~Game ()

  Destrutor da classe base Game.

Atributos Privados

- char current_player

Outros membros herdados

Atributos Protegidos herdados de Game

- Board game_board

4.2.1 Descrição detalhada

Gerencia as regras e funcionalidades do jogo Connect4.

Herda de Game e adiciona métodos específicos para o funcionamento do jogo, como verificar jogadas válidas, realizar jogadas e checar condições de vitória.

Definição na linha 22 do arquivo Connect4.hpp.

4.2.2 Construtores e Destrutores

4.2.2.1 Connect4()

```
Connect4::Connect4 ( )
```

Construtor padrão do Connect4, inicializa o jogo com 6 linhas, 7 colunas e o jogador 'X' como inicial.

Definição na linha 5 do arquivo Connect4.cpp.

4.2.2.2 ~Connect4()

```
Connect4::~Connect4 ( )
```

Destrutor do jogo Connect4.

Definição na linha 123 do arquivo Connect4.cpp.

4.2.3 Documentação das funções

4.2.3.1 check_win()

```
bool Connect4::check_win ( )  [override], [virtual]
```

Verifica em todas direções válidas se o jogador atual venceu o jogo.

Implementa Game.

Definição na linha 36 do arquivo Connect4.cpp.

4.2.3.2 get_current_player()

```
char Connect4::get_current_player ( )
```

Retorna o jogador atual.

Retorna

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

Definição na linha 96 do arquivo Connect4.cpp.

4.2.3.3 is_board_full()

```
bool Connect4::is_board_full ( ) const
```

Verifica se o tabuleiro está completamente cheio.

Retorna

`true` se o tabuleiro estiver cheio, `false` caso contrário.

Definição na linha 101 do arquivo Connect4.cpp.

4.2.3.4 is_valid_move() [1/2]

```
bool Connect4::is_valid_move ( ) const  [override], [virtual]
```

Função declarada somente para fins de sobregarga.

Implementa Game.

Definição na linha 8 do arquivo Connect4.cpp.

4.2.3.5 is_valid_move() [2/2]

```
bool Connect4::is_valid_move (
            int column )
```

Verifica se a jogada é válida para uma coluna específica.

Parâmetros

column	Coluna onde o jogador deseja jogar.

Retorna

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

Definição na linha 13 do arquivo Connect4.cpp.

4.2.3.6 make_move() [1/2]

```
void Connect4::make_move ( )  [override], [virtual]
```

Função declarada somente para fins de sobregarga.

Implementa Game.

Definição na linha 10 do arquivo Connect4.cpp.

4.2.3.7 make_move() [2/2]

```
void Connect4::make_move (
            int column )
```

Realiza a jogada na coluna especificada.

Parâmetros

column	Coluna onde o jogador deseja jogar.

Definição na linha 21 do arquivo Connect4.cpp.

4.2.3.8 print_game_board()

```
void Connect4::print_game_board ( ) const
```

Imprime o estado atual do tabuleiro.

Definição na linha 113 do arquivo Connect4.cpp.

4.2.3.9 set_current_player()

```
void Connect4::set_current_player (
            char player )
```

Define o jogador atual.

Parâmetros

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

Definição na linha 118 do arquivo Connect4.cpp.

4.2.4 Atributos

4.2.4.1 current_player

```
char Connect4::current_player  [private]
```

Representa o jogador atual ('X' ou 'O').

Definição na linha 24 do arquivo Connect4.hpp.

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

- include/Connect4.hpp
- src/Connect4.cpp

4.3 Referência da Classe Game

Classe base para jogos com tabuleiro.

```
#include <Game.hpp>
```

Diagrama de hierarquia da classe Game:

Diagrama de colaboração para Game:

Membros Públicos

- Game (int num_rows_received, int num_columns_received)

  Constrói um jogo com um tabuleiro de tamanho especificado.
- virtual bool is_valid_move () const =0

  Verifica se uma jogada é válida.
- virtual void make_move ()=0

  Realiza uma jogada.
- virtual bool check_win ()=0

  Verifica se há um vencedor no jogo.
- char switch_players (char current_player)

  Alterna entre os jogadores.
- ~Game ()

  Destrutor da classe base Game.

Atributos Protegidos

- Board game_board

4.3.1 Descrição detalhada

Classe base para jogos com tabuleiro.

Serve como base para implementar diferentes tipos de jogos, fornecendo métodos abstratos para validação de jogadas, execução de jogadas e verificação de vitória.

Definição na linha 20 do arquivo Game.hpp.

4.3.2 Construtores e Destrutores

4.3.2.1 Game()

```
Game::Game (
            int num_rows_received,
            int num_columns_received )
```

Constrói um jogo com um tabuleiro de tamanho especificado.

Construtor que inicializa o tabuleiro com o tamanho recebido.

Parâmetros

num_rows_received	Número de linhas do tabuleiro.
num_columns_received	Número de colunas do tabuleiro.

Definição na linha 7 do arquivo Game.cpp.

4.3.2.2 ~Game()

```
Game::~Game ( )  [inline]
```

Destrutor da classe base Game.

Não realiza nenhuma operação específica.

Definição na linha 72 do arquivo Game.hpp.

4.3.3 Documentação das funções

4.3.3.1 check_win()

```
virtual bool Game::check_win ( )  [pure virtual]
```

Verifica se há um vencedor no jogo.

Método abstrato que deve ser implementado pelas classes derivadas.

Retorna

`true` se houver um vencedor, `false` caso contrário.

Implementado por Connect4, Reversi e Tic_tac_toe.

4.3.3.2 is_valid_move()

```
virtual bool Game::is_valid_move ( ) const  [pure virtual]
```

Verifica se uma jogada é válida.

Método abstrato que deve ser implementado pelas classes derivadas.

Retorna

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

Implementado por Connect4, Reversi e Tic_tac_toe.

4.3.3.3 make_move()

```
virtual void Game::make_move ( )  [pure virtual]
```

Realiza uma jogada.

Método abstrato que deve ser implementado pelas classes derivadas.

Implementado por Connect4, Reversi e Tic_tac_toe.

4.3.3.4 switch_players()

```
char Game::switch_players (
            char current_player )
```

Alterna entre os jogadores.

Parâmetros

current_player	Jogador atual ('X' ou 'O').

Retorna

O caractere do próximo jogador.

Definição na linha 11 do arquivo Game.cpp.

4.3.4 Atributos

4.3.4.1 game_board

```
Board Game::game_board  [protected]
```

Representa o tabuleiro utilizado no jogo.

Definição na linha 22 do arquivo Game.hpp.

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

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

4.4 Referência da Classe Player

```
#include <Player.hpp>
```

Membros Públicos

- Player ()
  Construtor padrão que inicializa o jogador com valores vazios e estatísticas zeradas.
- Player (std::string name_received, std::string username_received)
  Construtor que inicializa o jogador com nome e username.
- Player (std::string name_received, std::string username_received, std::map< std::string, int > num_win_↩ received, std::map< std::string, int > num_loss_received)
  Construtor que inicializa o jogador com nome, username e estatísticas.
- void set_name (std::string name_received)
  Define o nome do jogador.
- void set_username (std::string username_received)
  Define o nome de usuário do jogador.
- void set_num_win (std::string key, int value)
  Atualiza o número de vitórias de um jogo.
- void set_num_loss (std::string key, int value)
  Atualiza o número de derrotas de um jogo.
- std::string get_username ()

Retorna o nome de usuário do jogador.

- std::string get_name ()

  Retorna o nome do jogador.
- std::map< std::string, int > get_num_win ()

  Retorna o mapa de vitórias por jogo.
- std::map< std::string, int > get_num_loss ()

  Retorna o mapa de derrotas por jogo.
- void print_player ()

  Imprime as informações do jogador no console.
- void add_win (std::string key)

  Incrementa o número de vitórias em um jogo.
- void add_loss (std::string key)

  Incrementa o número de derrotas em um jogo.

Membros públicos estáticos

- static bool register_player (Player player_received, std::list< Player > &player_list)

  Registra um jogador em uma lista de jogadores.
- static bool remove_player (std::string username_received, std::list< Player > &player_list)

  Remove um jogador da lista de jogadores.
- static bool compare_username (Player &player1, Player &player2)

  Compara os usernames de dois jogadores.
- static bool compare_name (Player &player1, Player &player2)

  Compara os nomes de dois jogadores.

Atributos Privados

- std::string name
- std::string username
- std::map< std::string, int > num_win
- std::map< std::string, int > num_loss

4.4.1 Descrição detalhada

Definição na linha 17 do arquivo Player.hpp.

4.4.2 Construtores e Destrutores

4.4.2.1 Player() [1/3]

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

Construtor padrão que inicializa o jogador com valores vazios e estatísticas zeradas.

Definição na linha 6 do arquivo Player.cpp.

4.4.2.2 Player() [2/3]

```
Player::Player (
            std::string name_received,
            std::string username_received )
```

Construtor que inicializa o jogador com nome e username.

Parâmetros

name_received	Nome do jogador.
username_received	Nome de usuário.

Definição na linha 9 do arquivo Player.cpp.

4.4.2.3 Player() [3/3]

```
Player::Player (
            std::string name_received,
            std::string username_received,
            std::map< std::string, int > num_win_received,
            std::map< std::string, int > num_loss_received )
```

Construtor que inicializa o jogador com nome, username e estatísticas.

Parâmetros

name_received	Nome do jogador.
username_received	Nome de usuário.
num_win_received	Mapa de vitórias por jogo.
num_loss_received	Mapa de derrotas por jogo.

Definição na linha 12 do arquivo Player.cpp.

4.4.3 Documentação das funções

4.4.3.1 add_loss()

```
void Player::add_loss (
            std::string key )
```

Incrementa o número de derrotas em um jogo.

Parâmetros

key	Nome do jogo.

Definição na linha 61 do arquivo Player.cpp.

4.4.3.2 add_win()

```
void Player::add_win (
            std::string key )
```

Incrementa o número de vitórias em um jogo.

Parâmetros

key	Nome do jogo.

Definição na linha 55 do arquivo Player.cpp.

4.4.3.3 compare_name()

```
bool Player::compare_name (
            Player & player1,
            Player & player2 )  [static]
```

Compara os nomes de dois jogadores.

Parâmetros

player1	Primeiro jogador.
player2	Segundo jogador.

Retorna

`true` se o nome do primeiro jogador for menor, `false` caso contrário.

Definição na linha 106 do arquivo Player.cpp.

4.4.3.4 compare_username()

```
bool Player::compare_username (
            Player & player1,
            Player & player2 )  [static]
```

Compara os usernames de dois jogadores.

Parâmetros

player1	Primeiro jogador.
player2	Segundo jogador.

Retorna

`true` se o username do primeiro jogador for menor, `false` caso contrário.

Definição na linha 96 do arquivo Player.cpp.

4.4.3.5 get_name()

```
std::string Player::get_name ( )
```

Retorna o nome do jogador.

Retorna

Nome do jogador.

Definição na linha 43 do arquivo Player.cpp.

4.4.3.6 get_num_loss()

```
std::map< std::string, int > Player::get_num_loss ( )
```

Retorna o mapa de derrotas por jogo.

Retorna

Mapa com o número de derrotas.

Definição na linha 51 do arquivo Player.cpp.

4.4.3.7 get_num_win()

```
std::map< std::string, int > Player::get_num_win ( )
```

Retorna o mapa de vitórias por jogo.

Retorna

Mapa com o número de vitórias.

Definição na linha 47 do arquivo Player.cpp.

4.4.3.8 get_username()

```
std::string Player::get_username ( )
```

Retorna o nome de usuário do jogador.

Retorna

Nome de usuário.

Definição na linha 39 do arquivo Player.cpp.

4.4.3.9 print_player()

```
void Player::print_player ( )
```

Imprime as informações do jogador no console.

Definição na linha 67 do arquivo Player.cpp.

4.4.3.10 register_player()

```
bool Player::register_player (
            Player player_received,
            std::list< Player > & player_list )  [static]
```

Registra um jogador em uma lista de jogadores.

Parâmetros

player_received	Jogador a ser registrado.
player_list	Lista onde o jogador será adicionado.

Retorna

`true` se o registro for bem-sucedido, `false` caso contrário.

Definição na linha 74 do arquivo Player.cpp.

4.4.3.11 remove_player()

```
bool Player::remove_player (
            std::string username_received,
            std::list< Player > & player_list )  [static]
```

Remove um jogador da lista de jogadores.

Parâmetros

username_received	Nome de usuário do jogador a ser removido.
player_list	Lista de onde o jogador será removido.

Retorna

`true` se a remoção for bem-sucedida, `false` caso contrário.

Definição na linha 85 do arquivo Player.cpp.

4.4.3.12 set_name()

```
void Player::set_name (
            std::string name_received )
```

Define o nome do jogador.

Parâmetros

name_received	Nome do jogador.

Definição na linha 15 do arquivo Player.cpp.

4.4.3.13 set_num_loss()

```
void Player::set_num_loss (
            std::string key,
            int value )
```

Atualiza o número de derrotas de um jogo.

Parâmetros

key	Nome do jogo.
value	Número de derrotas.

Definição na linha 31 do arquivo Player.cpp.

4.4.3.14 set_num_win()

```
void Player::set_num_win (
            std::string key,
            int value )
```

Atualiza o número de vitórias de um jogo.

Parâmetros

key	Nome do jogo.
value	Número de vitórias.

Definição na linha 23 do arquivo Player.cpp.

4.4.3.15 set_username()

```
void Player::set_username (
            std::string username_received )
```

Define o nome de usuário do jogador.

Parâmetros

username_received	Nome de usuário.

Definição na linha 19 do arquivo Player.cpp.

4.4.4 Atributos

4.4.4.1 name

```
std::string Player::name  [private]
```

Nome do jogador.

Definição na linha 19 do arquivo Player.hpp.

4.4.4.2 num_loss

```
std::map<std::string, int> Player::num_loss  [private]
```

Número de derrotas por jogo.

Definição na linha 22 do arquivo Player.hpp.

4.4.4.3 num_win

```
std::map<std::string, int> Player::num_win  [private]
```

Número de vitórias por jogo.

Definição na linha 21 do arquivo Player.hpp.

4.4.4.4 username

```
std::string Player::username  [private]
```

Nome de usuário do jogador.

Definição na linha 20 do arquivo Player.hpp.

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

- include/Player.hpp
- src/Player.cpp

4.5 Referência da Classe Reversi

Gerencia as regras e funcionalidades do jogo Reversi.

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

Diagrama de hierarquia da classe Reversi:

Diagrama de colaboração para Reversi:

Membros Públicos

- Reversi ()

  Construtor padrão do jogo Reversi. Inicializa o tabuleiro padrão do Reversi, além de iniciar cada jogador com 2 de suas respectivas peças.
- int get_num_pieces_player_X ()

  Retorna o número de peças do jogador X.
- int get_num_pieces_player_O ()

  Retorna o número de peças do jogador O.
- void start_reversi_board ()

  Inicializa o tabuleiro com as peças centrais do Reversi.
- void print_reversi_board () const

  Imprime o estado atual do tabuleiro.
- bool is_there_player_piece_at_the_direction (const char player_piece, const std::array< int, 2 > &direction, std::array< int, 2 > adjacent_square) const

  Verifica se há peças do jogador na direção especificada.
- bool is_space_free_reversi (int x, int y) const

  Verifica se uma posição no tabuleiro está livre.
- bool is_there_direction_that_captures_opponent (const std::array< int, 2 > &move_coordinates, char player_piece_type)

  Verifica se há alguma direção que captura peças do oponente.
- bool is_valid_move () const override

  Função declarada somente para fins de sobregarga.
- bool is_valid_move (std::array< int, 2 > &move_coordinates, char player_piece_type)

  Verifica se uma jogada específica é válida.
- void indicate_all_direction_to_make_move ()

  Indica todas as direções possíveis para realizar um movimento.
- void find_all_directions_to_make_move (std::array< int, 2 > &move_coordinates, char player_piece, std::list< std::array< int, 2 > > &directions_to_capture_opponents)

  Encontra todas as direções válidas para capturar peças do oponente.
- void flip_pieces (std::array< int, 2 > directions, std::array< int, 2 > move_coordinates, char player_piece)

  Captura peças do oponente em uma direção específica.
- void control_num_pieces_players (int num_pieces_flipped, char player_piece)

  Controla o número de peças dos jogadores após uma jogada.
- void make_move () override

  Função declarada somente para fins de sobregarga.
- void make_move (std::array< int, 2 > move_coordinates, char player_piece, std::list< std::array< int, 2 > > &directions_to_capture_opponents)

  Realiza a jogada do jogador atual.
- bool is_there_valid_move_for_player (char player_piece)

  Verifica se há uma jogada válida para o jogador.
- bool process_move (std::array< int, 2 > move_coordinates, char player_piece_type)

  Processa uma jogada e verifica sua validade.
- bool check_win () override

  Função declarada somente para fins de sobregarga.
- bool check_win (bool is_there_move_for_player, char opponent_piece)

  Verifica se o jogo tem vencedor considerando as jogadas restantes.
- void register_win_and_loss (Player *player1, Player *player2)

  Registra vitória e derrota dos jogadores.
- ~Reversi ()

  Destrutor do jogo Reversi.

Membros Públicos herdados de Game

- Game (int num_rows_received, int num_columns_received)

  Constrói um jogo com um tabuleiro de tamanho especificado.
- char switch_players (char current_player)

  Alterna entre os jogadores.
- ~Game ()

  Destrutor da classe base Game.

Atributos Privados

- int num_pieces_player_X
- int num_pieces_player_O

Outros membros herdados

Atributos Protegidos herdados de Game

- Board game_board

4.5.1 Descrição detalhada

Gerencia as regras e funcionalidades do jogo Reversi.

Herda de Game e adiciona métodos específicos para o funcionamento do jogo Reversi, como validação de jogadas, captura de peças e verificação de vitória.

Definição na linha 23 do arquivo Reversi.hpp.

4.5.2 Construtores e Destrutores

4.5.2.1 Reversi()

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

Construtor padrão do jogo Reversi. Inicializa o tabuleiro padrão do Reversi, além de iniciar cada jogador com 2 de suas respectivas peças.

Definição na linha 35 do arquivo Reversi.cpp.

4.5.2.2 ~Reversi()

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

Destrutor do jogo Reversi.

Definição na linha 273 do arquivo Reversi.cpp.

4.5.3 Documentação das funções

4.5.3.1 check_win() [1/2]

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

Função declarada somente para fins de sobregarga.

Implementa Game.

Definição na linha 281 do arquivo Reversi.cpp.

4.5.3.2 check_win() [2/2]

```
bool Reversi::check_win (
            bool is_there_move_for_player,
            char opponent_piece )
```

Verifica se o jogo tem vencedor considerando as jogadas restantes.

Parâmetros

is_there_move_for_player	Indica se há jogadas válidas para o jogador.
opponent_piece	Tipo de peça do oponente.

Retorna

`true` se houver vencedor, `false` caso contrário.

Definição na linha 252 do arquivo Reversi.cpp.

4.5.3.3 control_num_pieces_players()

```
void Reversi::control_num_pieces_players (
            int num_pieces_flipped,
            char player_piece )
```

Controla o número de peças dos jogadores após uma jogada.

Parâmetros

num_pieces_flipped	Número de peças capturadas.
player_piece	Tipo de peça do jogador.

Definição na linha 173 do arquivo Reversi.cpp.

4.5.3.4 find_all_directions_to_make_move()

```
void Reversi::find_all_directions_to_make_move (
```

```
            std::array< int, 2 > & move_coordinates,
            char player_piece,
            std::list< std::array< int, 2 > > & directions_to_capture_opponents )
```

Encontra todas as direções válidas para capturar peças do oponente.

Parâmetros

move_coordinates	Coordenadas do movimento.
player_piece	Tipo de peça do jogador.
directions_to_capture_opponents	Lista de direções válidas.

Definição na linha 123 do arquivo Reversi.cpp.

4.5.3.5 flip_pieces()

```
void Reversi::flip_pieces (
            std::array< int, 2 > directions,
            std::array< int, 2 > move_coordinates,
            char player_piece )
```

Captura peças do oponente em uma direção específica.

Parâmetros

directions	Direção onde as peças serão capturadas.
move_coordinates	Coordenadas do movimento.
player_piece	Tipo de peça do jogador.

Definição na linha 154 do arquivo Reversi.cpp.

4.5.3.6 get_num_pieces_player_O()

```
int Reversi::get_num_pieces_player_O ( )
```

Retorna o número de peças do jogador O.

Retorna

Número de peças do jogador O.

Definição na linha 23 do arquivo Reversi.cpp.

4.5.3.7 get_num_pieces_player_X()

```
int Reversi::get_num_pieces_player_X ( )
```

Retorna o número de peças do jogador X.

Retorna

Número de peças do jogador X.

Definição na linha 17 do arquivo Reversi.cpp.

4.5.3.8 indicate_all_direction_to_make_move()

```
void Reversi::indicate_all_direction_to_make_move ( )
```

Indica todas as direções possíveis para realizar um movimento.

4.5.3.9 is_space_free_reversi()

```
bool Reversi::is_space_free_reversi (
            int x,
            int y ) const
```

Verifica se uma posição no tabuleiro está livre.

Parâmetros

x	Linha da posição.
y	Coluna da posição.

Retorna

`true` se a posição estiver livre, `false` caso contrário.

Definição na linha 43 do arquivo Reversi.cpp.

4.5.3.10 is_there_direction_that_captures_opponent()

```
bool Reversi::is_there_direction_that_captures_opponent (
            const std::array< int, 2 > & move_coordinates,
            char player_piece_type )
```

Verifica se há alguma direção que captura peças do oponente.

Parâmetros

move_coordinates	Coordenadas do movimento.
player_piece_type	Tipo de peça do jogador.

Retorna

`true` se houver direção válida, `false` caso contrário.

Definição na linha 72 do arquivo Reversi.cpp.

4.5.3.11 is_there_player_piece_at_the_direction()

```
bool Reversi::is_there_player_piece_at_the_direction (
            const char player_piece,
```

```
            const std::array< int, 2 > & direction,
            std::array< int, 2 > adjacent_square ) const
```

Verifica se há peças do jogador na direção especificada.

Parâmetros

player_piece	Tipo de peça do jogador.
direction	Direção a ser verificada.
adjacent_square	Posição adjacente inicial.

Retorna

`true` se houver peças na direção, `false` caso contrário.

Definição na linha 51 do arquivo Reversi.cpp.

4.5.3.12 is_there_valid_move_for_player()

```
bool Reversi::is_there_valid_move_for_player (
            char player_piece )
```

Verifica se há uma jogada válida para o jogador.

Parâmetros

player_piece	Tipo de peça do jogador.

Retorna

`true` se houver jogada válida, `false` caso contrário.

Definição na linha 224 do arquivo Reversi.cpp.

4.5.3.13 is_valid_move() [1/2]

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

Função declarada somente para fins de sobregarga.

Implementa Game.

Definição na linha 277 do arquivo Reversi.cpp.

4.5.3.14 is_valid_move() [2/2]

```
bool Reversi::is_valid_move (
            std::array< int, 2 > & move_coordinates,
            char player_piece_type )
```

Verifica se uma jogada específica é válida.

Parâmetros

move_coordinates	Coordenadas do movimento.
player_piece_type	Tipo de peça do jogador.

Retorna

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

Definição na linha 106 do arquivo Reversi.cpp.

4.5.3.15 make_move() [1/2]

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

Função declarada somente para fins de sobregarga.

Implementa Game.

Definição na linha 279 do arquivo Reversi.cpp.

4.5.3.16 make_move() [2/2]

```
void Reversi::make_move (
            std::array< int, 2 > move_coordinates,
            char player_piece,
            std::list< std::array< int, 2 > > & directions_to_capture_opponents )
```

Realiza a jogada do jogador atual.

Parâmetros

move_coordinates	Coordenadas do movimento.
player_piece	Tipo de peça do jogador.
directions_to_capture_opponents	Direções para capturar peças.

Definição na linha 191 do arquivo Reversi.cpp.

4.5.3.17 print_reversi_board()

```
void Reversi::print_reversi_board ( ) const
```

Imprime o estado atual do tabuleiro.

Definição na linha 29 do arquivo Reversi.cpp.

4.5.3.18 process_move()

```
bool Reversi::process_move (
            std::array< int, 2 > move_coordinates,
            char player_piece_type )
```

Processa uma jogada e verifica sua validade.

Parâmetros

move_coordinates	Coordenadas do movimento.
player_piece_type	Tipo de peça do jogador.

Retorna

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

Definição na linha 208 do arquivo Reversi.cpp.

4.5.3.19 register_win_and_loss()

```
void Reversi::register_win_and_loss (
            Player * player1,
            Player * player2 )
```

Registra vitória e derrota dos jogadores.

Parâmetros

player1	Jogador 1.
player2	Jogador 2.

Definição na linha 260 do arquivo Reversi.cpp.

4.5.3.20 start_reversi_board()

```
void Reversi::start_reversi_board ( )
```

Inicializa o tabuleiro com as peças centrais do Reversi.

Definição na linha 8 do arquivo Reversi.cpp.

4.5.4 Atributos

4.5.4.1 num_pieces_player_O

```
int Reversi::num_pieces_player_O  [private]
```

Número de peças do jogador O.

Definição na linha 26 do arquivo Reversi.hpp.

4.5.4.2 num_pieces_player_X

```
int Reversi::num_pieces_player_X  [private]
```

Número de peças do jogador X.

Definição na linha 25 do arquivo Reversi.hpp.

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

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

4.6 Referência da Classe Tic_tac_toe

Gerencia as regras e funcionalidades do Jogo da Velha.

```
#include <Tic_tac_toe.hpp>
```

Diagrama de hierarquia da classe Tic_tac_toe:

Diagrama de colaboração para Tic_tac_toe:

Membros Públicos

- Tic_tac_toe ()

  Construtor padrão do jogo da velha. Inicializa o tabuleiro como 3x3, define o jogador atual como 'X' e o vencedor como 'F' (nenhum).
- void make_move () override

  Função declarada somente para fins de sobregarga.
- void make_move (int x, int y)

  Realiza uma jogada em uma posição específica.
- bool is_valid_move () const override

  Função declarada somente para fins de sobregarga.
- bool is_valid_move (int &x, int &y) const

  Verifica se uma jogada específica é válida.
- bool check_win () override

  Função declarada somente para fins de sobregarga.
- char check_tic_tac_toe_win () const

  Verifica se há um vencedor no jogo.
- char get_current_player () const

  Retorna o jogador atual.
- bool check_tie () const

  Verifica se o jogo terminou em empate.
- void print_tic_tac_toe_board () const

  Imprime o estado atual do tabuleiro.
- ~Tic_tac_toe ()

  Destrutor do Jogo da Velha.

Membros Públicos herdados de Game

- Game (int num_rows_received, int num_columns_received)

  Constrói um jogo com um tabuleiro de tamanho especificado.
- char switch_players (char current_player)

  Alterna entre os jogadores.
- ~Game ()

  Destrutor da classe base Game.

Atributos Privados

- char current_player
- char winner

Outros membros herdados

Atributos Protegidos herdados de Game

- Board game_board

4.6.1 Descrição detalhada

Gerencia as regras e funcionalidades do Jogo da Velha.

Herda de Game e adiciona métodos específicos para o funcionamento do Jogo da Velha, como validação de jogadas, verificação de vitória e empate.

Definição na linha 20 do arquivo Tic_tac_toe.hpp.

4.6.2 Construtores e Destrutores

4.6.2.1 Tic_tac_toe()

```
Tic_tac_toe::Tic_tac_toe ( )
```

Construtor padrão do jogo da velha. Inicializa o tabuleiro como 3x3, define o jogador atual como 'X' e o vencedor como 'F' (nenhum).

Definição na linha 8 do arquivo Tic_tac_toe.cpp.

4.6.2.2 ~Tic_tac_toe()

```
Tic_tac_toe::~Tic_tac_toe ( )
```

Destrutor do Jogo da Velha.

Definição na linha 109 do arquivo Tic_tac_toe.cpp.

4.6.3 Documentação das funções

4.6.3.1 check_tic_tac_toe_win()

```
char Tic_tac_toe::check_tic_tac_toe_win ( ) const
```

Verifica se há um vencedor no jogo.

Confere todas as linhas, colunas e diagonais para ver se há três peças consecutivas do mesmo jogador.

Retorna

Caractere do jogador vencedor ('X' ou 'O') ou 'F' se não houver vencedor.

Definição na linha 56 do arquivo Tic_tac_toe.cpp.

4.6.3.2 check_tie()

```
bool Tic_tac_toe::check_tie ( ) const
```

Verifica se o jogo terminou em empate.

Confere se todas as posições do tabuleiro estão ocupadas sem haver vitória.

Retorna

`true` se houver empate, `false` caso contrário.

Definição na linha 89 do arquivo Tic_tac_toe.cpp.

4.6.3.3 check_win()

```
bool Tic_tac_toe::check_win ( )  [override], [virtual]
```

Função declarada somente para fins de sobregarga.

Implementa Game.

Definição na linha 117 do arquivo Tic_tac_toe.cpp.

4.6.3.4 get_current_player()

```
char Tic_tac_toe::get_current_player ( ) const
```

Retorna o jogador atual.

Retorna

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

Definição na linha 104 do arquivo Tic_tac_toe.cpp.

4.6.3.5 is_valid_move() [1/2]

```
bool Tic_tac_toe::is_valid_move ( ) const  [override], [virtual]
```

Função declarada somente para fins de sobregarga.

Implementa Game.

Definição na linha 113 do arquivo Tic_tac_toe.cpp.

4.6.3.6 is_valid_move() [2/2]

```
bool Tic_tac_toe::is_valid_move (
            int & x,
            int & y ) const
```

Verifica se uma jogada específica é válida.

Valida se as coordenadas fornecidas estão dentro do tabuleiro e a posição está livre através das funções da classe Board.

Parâmetros

x	Coordenada da linha da jogada.
y	Coordenada da coluna da jogada.

Retorna

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

Definição na linha 11 do arquivo Tic_tac_toe.cpp.

4.6.3.7 make_move() [1/2]

```
void Tic_tac_toe::make_move ( )  [override], [virtual]
```

Função declarada somente para fins de sobregarga.

Implementa Game.

Definição na linha 115 do arquivo Tic_tac_toe.cpp.

4.6.3.8 make_move() [2/2]

```
void Tic_tac_toe::make_move (
            int x,
            int y )
```

Realiza uma jogada em uma posição específica.

Verifica se a jogada é válida, atualiza o tabuleiro e alterna o jogador através de funções da classe Tic_tac_toe e da classe Board.

Parâmetros

x	Coordenada da linha.
y	Coordenada da coluna.

Definição na linha 25 do arquivo Tic_tac_toe.cpp.

4.6.3.9 print_tic_tac_toe_board()

```
void Tic_tac_toe::print_tic_tac_toe_board ( ) const
```

Imprime o estado atual do tabuleiro.

Definição na linha 19 do arquivo Tic_tac_toe.cpp.

4.6.4 Atributos

4.6.4.1 current_player

```
char Tic_tac_toe::current_player  [private]
```

Jogador atual ('X' ou 'O').

Definição na linha 22 do arquivo Tic_tac_toe.hpp.

4.6.4.2 winner

```
char Tic_tac_toe::winner  [private]
```

Armazena o vencedor do jogo, se houver.

Definição na linha 23 do arquivo Tic_tac_toe.hpp.

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

- include/Tic_tac_toe.hpp
- src/Tic_tac_toe.cpp

Capítulo 5

Arquivos

5.1 Referência do Arquivo include/Board.hpp

Representa o tabuleiro de um jogo genérico.

`#include <memory>`
Gráfico de dependência de inclusões para Board.hpp: Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com esse arquivo:

Componentes

- class Board

  Gerencia o tabuleiro do jogo.

5.1.1 Descrição detalhada

Representa o tabuleiro de um jogo genérico.

Define o tabuleiro com funcionalidades para manipular posições e verificar condições dentro do jogo.

Definição no arquivo Board.hpp.

5.2 Board.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef BOARD_H
00002 #define BOARD_H
00003 #include <memory>
00004
00020 class Board {
00021     private:
00022         int num_rows;
00023         int num_columns;
00024         std::unique_ptr<std::unique_ptr<char[]>[]> game_board = nullptr;
00026     public:
00032         Board(int num_rows_received, int num_columns_received);
00033
00034
00041         void set_space(int row, int column, char value);
```

```
00042
00043
00050         char get_space(int row, int column) const;
00051
00052
00056         void print_game_board() const;
00057
00058
00065         bool is_move_inside_board(int x, int y) const;
00066
00067
00074         bool is_space_free(int x, int y) const;
00075 };
00076
00077 #endif
```

5.3 Referência do Arquivo include/Connect4.hpp

Implementa o jogo Connect4 (Lig4), baseado na classe genérica Game.

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

Gráfico de dependência de inclusões para Connect4.hpp: Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com esse arquivo:

Componentes

- class Connect4

  Gerencia as regras e funcionalidades do jogo Connect4.

5.3.1 Descrição detalhada

Implementa o jogo Connect4 (Lig4), baseado na classe genérica Game.

Contém as regras e ações específicas do jogo Connect4.

Definição no arquivo Connect4.hpp.

5.4 Connect4.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef CONNECT4_H
00002 #define CONNECT4_H
00003
00004 #include "Game.hpp"
00005 #include <iostream>
00006
00022 class Connect4 : public Game {
00023     private:
00024         char current_player;
00026     public:
00031         Connect4();
00032
00033
00037         bool is_valid_move() const override;
00038
00039
00043         void make_move() override;
00044
00045
00049         bool check_win() override;
```

```
00050
00051
00057         bool is_valid_move(int column);
00058
00059
00064         void make_move(int column);
00065
00066
00071         char get_current_player();
00072
00073
00078         bool is_board_full() const;
00079
00080
00084         void print_game_board() const;
00085
00086
00091         void set_current_player(char player);
00092
00093
00097         ~Connect4();
00098 };
00099
00100 #endif
```

5.5 Referência do Arquivo include/Game.hpp

Classe base abstrata para jogos genéricos com tabuleiro.

```
#include "Board.hpp"
#include <array>
```

Gráfico de dependência de inclusões para Game.hpp: Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com esse arquivo:

Componentes

- class Game

  Classe base para jogos com tabuleiro.

5.5.1 Descrição detalhada

Classe base abstrata para jogos genéricos com tabuleiro.

Define a estrutura e os métodos principais para jogos que utilizam um tabuleiro.

Definição no arquivo Game.hpp.

5.6 Game.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef GAME_H
00002 #define GAME_H
00003 #include "Board.hpp"
00004 #include <array>
00005
00020 class Game {
00021     protected:
00022         Board game_board;
00024     public:
00030         Game(int num_rows_received, int num_columns_received);
00031
```

```
00032
00039         virtual bool is_valid_move() const = 0;
00040
00041
00047         virtual void make_move() = 0;
00048
00049
00056         virtual bool check_win() = 0;
00057
00058
00064         char switch_players(char current_player);
00065
00066
00072         ~Game() {}
00073 };
00074
00075 #endif
```

5.7 Referência do Arquivo include/Player.hpp

Gerencia informações e ações relacionadas a jogadores.

```
#include <iostream>
#include <map>
#include <list>
```

Gráfico de dependência de inclusões para Player.hpp: Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com esse arquivo:

Componentes

- class Player

Funções

- void read_register_file (std::list< Player > &player_list, std::ifstream &file_in)

  Lê os dados de registro de jogadores de um arquivo.
- void write_register_file (std::list< Player > &player_list, std::ofstream &file_out)

  Escreve os dados de registro de jogadores em um arquivo.

5.7.1 Descrição detalhada

Gerencia informações e ações relacionadas a jogadores.

Define a estrutura de dados e métodos para representar jogadores, incluindo estatísticas de vitórias e derrotas, e funcionalidades para manipular listas de jogadores.

Definição no arquivo Player.hpp.

5.7.2 Funções

5.7.2.1 read_register_file()

```
void read_register_file (
            std::list< Player > & player_list,
            std::ifstream & file_in )
```

Lê os dados de registro de jogadores de um arquivo.

Parâmetros

player_list	Lista de jogadores a ser preenchida.
file_in	Arquivo de entrada contendo os registros.

Definição na linha 116 do arquivo Player.cpp.

5.7.2.2 write_register_file()

```
void write_register_file (
            std::list< Player > & player_list,
            std::ofstream & file_out )
```

Escreve os dados de registro de jogadores em um arquivo.

Parâmetros

player_list	Lista de jogadores a ser registrada.
file_out	Arquivo de saída onde os registros serão armazenados.

Definição na linha 138 do arquivo Player.cpp.

5.8 Player.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef PLAYER_H
00002 #define PLAYER_H
00003
00004 #include <iostream>
00005 #include <map>
00006 #include <list>
00007
00017 class Player {
00018     private:
00019         std::string name;
00020         std::string username;
00021         std::map<std::string, int> num_win;
00022         std::map<std::string, int> num_loss;
00024     public:
00028         Player();
00029
00035         Player(std::string name_received, std::string username_received);
00036
00044         Player(std::string name_received, std::string username_received, std::map<std::string, int>
      num_win_received, std::map<std::string, int> num_loss_received);
00045
00050         void set_name(std::string name_received);
00051
00056         void set_username(std::string username_received);
00057
00063         void set_num_win(std::string key, int value);
00064
00070         void set_num_loss(std::string key, int value);
00071
00076         std::string get_username();
00077
00082         std::string get_name();
00083
00088         std::map<std::string, int> get_num_win();
00089
00094         std::map<std::string, int> get_num_loss();
00095
00099         void print_player();
```

```
00100
00105         void add_win(std::string key);
00106
00111         void add_loss(std::string key);
00112
00119         static bool register_player(Player player_received, std::list<Player> &player_list);
00120
00127         static bool remove_player(std::string username_received, std::list<Player> &player_list);
00128
00135         static bool compare_username(Player &player1, Player &player2);
00136
00143         static bool compare_name(Player &player1, Player &player2);
00144 };
00145
00151 void read_register_file(std::list<Player> &player_list, std::ifstream &file_in);
00152
00158 void write_register_file(std::list<Player> &player_list, std::ofstream &file_out);
00159
00160 #endif
```

5.9 Referência do Arquivo include/Reversi.hpp

Implementa o jogo Reversi, baseado na classe genérica Game.

```
#include "Game.hpp"
#include "Player.hpp"
#include <array>
#include <list>
```

Gráfico de dependência de inclusões para Reversi.hpp: Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com esse arquivo:

Componentes

- class Reversi

  Gerencia as regras e funcionalidades do jogo Reversi.

5.9.1 Descrição detalhada

Implementa o jogo Reversi, baseado na classe genérica Game.

Contém as regras e ações específicas do jogo Reversi.

Definição no arquivo Reversi.hpp.

5.10 Reversi.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef REVERSI_H
00002 #define REVERSI_H
00003
00004 #include "Game.hpp"
00005 #include "Player.hpp"
00006 #include <array>
00007 #include <list>
00008
00023 class Reversi : public Game {
00024     private:
00025         int num_pieces_player_X;
00026         int num_pieces_player_O;
00028     public:
```

```
00033         Reversi();
00034
00035
00040         int get_num_pieces_player_X();
00041
00042
00047         int get_num_pieces_player_O();
00048
00049
00053         void start_reversi_board();
00054
00055
00059         void print_reversi_board() const;
00060
00061
00069         bool is_there_player_piece_at_the_direction(const char player_piece, const std::array<int, 2>&
      direction,
00070              std::array<int, 2> adjacent_square) const;
00071
00072
00079         bool is_space_free_reversi(int x, int y) const;
00080
00081
00088         bool is_there_direction_that_captures_opponent(const std::array<int, 2> &move_coordinates,
      char player_piece_type);
00089
00090
00094         bool is_valid_move() const override;
00095
00096
00103         bool is_valid_move(std::array<int, 2>& move_coordinates, char player_piece_type);
00104
00105
00109         void indicate_all_direction_to_make_move();
00110
00111
00118         void find_all_directions_to_make_move(std::array<int, 2>& move_coordinates, char player_piece,
00119                                               std::list<std::array<int, 2»&
      directions_to_capture_opponents);
00120
00121
00128         void flip_pieces(std::array<int, 2> directions, std::array<int, 2> move_coordinates, char
      player_piece);
00129
00130
00136         void control_num_pieces_players(int num_pieces_flipped, char player_piece);
00137
00138
00142         void make_move() override;
00143
00144
00151         void make_move(std::array<int, 2> move_coordinates, char player_piece,
00152                        std::list<std::array<int, 2»& directions_to_capture_opponents);
00153
00154
00160         bool is_there_valid_move_for_player(char player_piece);
00161
00162
00169         bool process_move(std::array<int, 2> move_coordinates, char player_piece_type);
00170
00171
00175         bool check_win() override;
00176
00177
00184         bool check_win(bool is_there_move_for_player, char opponent_piece);
00185
00186
00192         void register_win_and_loss(Player *player1, Player *player2);
00193
00194
00198         ~Reversi();
00199 };
00200
00201 #endif
```

5.11 Referência do Arquivo include/Tic_tac_toe.hpp

Implementa o Jogo da Velha (Tic Tac Toe), baseado na classe genérica Game.

`#include "Game.hpp"`

Gráfico de dependência de inclusões para Tic_tac_toe.hpp: Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com esse arquivo:

Componentes

- class Tic_tac_toe

  Gerencia as regras e funcionalidades do Jogo da Velha.

5.11.1 Descrição detalhada

Implementa o Jogo da Velha (Tic Tac Toe), baseado na classe genérica Game.

Contém as regras e ações específicas do Jogo da Velha.

Definição no arquivo Tic_tac_toe.hpp.

5.12 Tic_tac_toe.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef TIC_TAC_TOE_H
00002 #define TIC_TAC_TOE_H
00003
00004 #include "Game.hpp"
00005
00020 class Tic_tac_toe : public Game {
00021     private:
00022         char current_player;
00023         char winner;
00025     public:
00030         Tic_tac_toe();
00031
00032
00036         void make_move() override;
00037
00038
00047         void make_move(int x, int y);
00048
00049
00053         bool is_valid_move() const override;
00054
00055
00065         bool is_valid_move(int& x, int& y) const;
00066
00067
00071         bool check_win() override;
00072
00073
00080         char check_tic_tac_toe_win() const;
00081
00082
00087         char get_current_player() const;
00088
00089
00096         bool check_tie() const;
00097
00098
00102         void print_tic_tac_toe_board() const;
00103
00104
00108         ~Tic_tac_toe();
00109 };
00110
00111 #endif
```

5.13 Referência do Arquivo src/Board.cpp

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

Gráfico de dependência de inclusões para Board.cpp:

5.14 Board.cpp

Ir para a documentação desse arquivo.

```
00001 #include "Board.hpp"
00002 #include <iostream>
00003
00004 void Board::set_space(int row, int column, char value)
00005 {
00006     this->game_board[row][column] = value;
00007 }
00008
00009
00010 char Board::get_space(int row, int column) const
00011 {
00012     return this->game_board[row][column];
00013 }
00014
00018 Board::Board(int num_rows_received, int num_columns_received) : num_rows(num_rows_received),
00019 num_columns(num_columns_received)
00020 {
00021     game_board = std::unique_ptr<std::unique_ptr<char[]>[]>(new std::unique_ptr<char[]>[num_rows]);
00022
00023     for (int i = 0; i < num_rows; ++i) {
00024         game_board[i] = std::unique_ptr<char[]>(new char[num_columns]);
00025         for (int j = 0; j < num_columns; j++){
00026             game_board[i][j] = ' ';
00027         }
00028     }
00029 }
00030
00031
00032 void Board::print_game_board() const
00033 {
00034     for(int i = 0; i < num_rows; i++) {
00035         std::cout « "|" « std::ends;
00036         for(int j = 0; j < num_columns; j++) {
00037             std::cout « this->game_board[i][j] « "|" « std::ends;
00038         }
00039         std::cout « std::endl;
00040     }
00041 }
00042
00043
00044 bool Board::is_move_inside_board(int x, int y) const
00045 {
00046     if ((x < 0 || x > this->num_rows - 1) || (y < 0 || y > this->num_columns - 1))
00047         return false;
00048
00049     return true;
00050 }
00051
00052
00053 bool Board::is_space_free(int x, int y) const
00054 {
00055     if ((this->game_board[x][y] == ' '))
00056         return true;
00057
00058     return false;
00059 }
```

5.15 Referência do Arquivo src/Connect4.cpp

```
#include "Connect4.hpp"
#include <iostream>
```

Gráfico de dependência de inclusões para Connect4.cpp:

5.16 Connect4.cpp

Ir para a documentação desse arquivo.

```
00001 #include "Connect4.hpp"
00002 #include <iostream>
00003
```

```
00004
00005 Connect4::Connect4() : Game(6, 7), current_player('X') {}
00006
00007 // Funções declaradas somente para fins de sobrecarga
00008 bool Connect4::is_valid_move() const { return true; }
00009
00010 void Connect4::make_move() {}
00011 //
00012
00013 bool Connect4::is_valid_move(int column)
00014 {
00015     column--;
00016     if (game_board.get_space(0, column) != ' ') return false;
00017     if (column < 0 || column >= 7) return false;
00018     return true;
00019 }
00020
00021 void Connect4::make_move(int column)
00022 {
00023     column--;
00024     // Encontra a linha mais baixa disponível na coluna e coloca a peça do jogador atual
00025     for(int i = 5; i >= 0; i--)
00026     {
00027         if(game_board.get_space(i, column) == ' ')
00028         {
00029             game_board.set_space(i, column, current_player);
00030             return;
00031         }
00032     }
00033 }
00034
00035
00036 bool Connect4::check_win()
00037 {
00038     // Verificação de vitória horizontal
00039     for (int row = 0; row < 6; ++row)
00040     {
00041         for (int col = 0; col <= 3; ++col)
00042         {
00043             if (game_board.get_space(row, col) == current_player &&
00044                 game_board.get_space(row, col + 1) == current_player &&
00045                 game_board.get_space(row, col + 2) == current_player &&
00046                 game_board.get_space(row, col + 3) == current_player) {
00047                     return true;
00048             }
00049         }
00050     }
00051
00052     // Verificação de vitória vertical
00053     for (int row = 0; row <= 2; ++row)
00054     {
00055         for (int col = 0; col < 7; ++col)
00056         {
00057             if (game_board.get_space(row, col) == current_player &&
00058                 game_board.get_space(row + 1, col) == current_player &&
00059                 game_board.get_space(row + 2, col) == current_player &&
00060                 game_board.get_space(row + 3, col) == current_player) {
00061                     return true;
00062             }
00063         }
00064     }
00065
00066     // Verificação de vitória diagonal para a direita
00067     for (int row = 0; row <= 2; ++row)
00068     {
00069         for (int col = 0; col <= 3; ++col)
00070         {
00071             if (game_board.get_space(row, col) == current_player &&
00072                 game_board.get_space(row + 1, col + 1) == current_player &&
00073                 game_board.get_space(row + 2, col + 2) == current_player &&
00074                 game_board.get_space(row + 3, col + 3) == current_player) {
00075                     return true;
00076             }
00077         }
00078     }
00079
00080     // Verificação de vitória diagonal para a esquerda
00081     for (int row = 3; row < 6; ++row)
00082     {
00083         for (int col = 0; col <= 3; ++col)
00084         {
00085             if (game_board.get_space(row, col) == current_player &&
00086                 game_board.get_space(row - 1, col + 1) == current_player &&
00087                 game_board.get_space(row - 2, col + 2) == current_player &&
00088                 game_board.get_space(row - 3, col + 3) == current_player) {
00089                     return true;
00090             }
```

```
00091         }
00092     }
00093     return false;
00094 }
00095
00096 char Connect4::get_current_player()
00097 {
00098     return current_player;
00099 }
00100
00101 bool Connect4::is_board_full() const
00102 {
00103     for(int col = 0; col < 7; ++col)
00104     {
00105         if(game_board.get_space(0, col) == ' ')
00106         {
00107             return false;
00108         }
00109     }
00110     return true;
00111 }
00112
00113 void Connect4::print_game_board() const
00114 {
00115     game_board.print_game_board();
00116 }
00117
00118 void Connect4::set_current_player(char player)
00119 {
00120     current_player = player;
00121 }
00122
00123 Connect4::~Connect4() {}
```

5.17 Referência do Arquivo src/Game.cpp

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

Gráfico de dependência de inclusões para Game.cpp:

5.18 Game.cpp

Ir para a documentação desse arquivo.

```
00001 #include "Game.hpp"
00002 #include <iostream>
00003
00007 Game::Game(int num_rows_received, int num_columns_received)
00008     : game_board(num_rows_received, num_columns_received) {}
00009
00010
00011 char Game::switch_players(char current_player)
00012 {
00013     char opponent_player = (current_player == 'X') ? 'O' : 'X';
00014     return opponent_player;
00015 }
```

5.19 Referência do Arquivo src/main.cpp

```
#include "Player.hpp"
#include "Reversi.hpp"
#include "Tic_tac_toe.hpp"
#include "Connect4.hpp"
#include <limits>
#include <fstream>
#include <algorithm>
#include <bits/stdc++.h>
```

Gráfico de dependência de inclusões para main.cpp:

Funções

- Player * find_player_in_list (std::list< Player > &player_list, const std::string &user_name)

  Procura se há um jogador específico existe na lista.
- int main ()

  Função principal que gerencia os comandos do sistema de jogadores e execução de jogos.

5.19.1 Funções

5.19.1.1 find_player_in_list()

```
Player * find_player_in_list (
            std::list< Player > & player_list,
            const std::string & user_name )
```

Procura se há um jogador específico existe na lista.

Parâmetros

player_list	Lista que registra todos os jogadores.
user_name	Nome do jogador a ser procurado.

Retorna

O endereço de memória do jogador caso seja encontrado, 'nullptr' caso contrário.

Definição na linha 17 do arquivo main.cpp.

5.19.1.2 main()

```
int main ( )
```

Função principal que gerencia os comandos do sistema de jogadores e execução de jogos.

Realiza operações como listar, cadastrar e remover jogadores, além de permitir a execução dos jogos Reversi, Lig4 (Connect4) e Velha (Tic Tac Toe).

Definição na linha 33 do arquivo main.cpp.

5.20 main.cpp

Ir para a documentação desse arquivo.

```
00001 #include "Player.hpp"
00002 #include "Reversi.hpp"
00003 #include "Tic_tac_toe.hpp"
00004 #include "Connect4.hpp"
00005 #include <limits>
00006 #include <fstream>
00007 #include <algorithm>
00008 #include <bits/stdc++.h>
00009
00017 Player *find_player_in_list(std::list<Player> &player_list, const std::string &user_name)
```

```
00018 {
00019     for (auto &player : player_list)
00020     {
00021         if (player.get_username() == user_name)
00022             return &player;
00023     }
00024     return nullptr;
00025 }
00026 
00027 
00033 int main()
00034 {
00035     std::ifstream file_in;
00036     file_in.open("/home/leonardo/PDS2-Final_Project/teste");
00037     if (!file_in.is_open())
00038     {
00039         std::cout « "Erro ao abrir o arquivo" « std::endl;
00040         return 1;
00041     }
00042 
00043     std::list<Player> player_list;
00044     read_register_file(player_list, file_in);
00045 
00046     file_in.close();
00047 
00048     std::string command;
00049     std::string name_in, username_in;
00050     bool error = false;
00051 
00052     // Loop principal que processa os comandos do usuário
00053     while (std::cin » command)
00054     {
00055         if (command == "LJ")
00056         {
00057             // Listar jogadores ordenados por nome ou username
00058             char sort_command;
00059             std::cin » sort_command;
00060             if (sort_command == 'A')
00061                 player_list.sort(Player::compare_username);
00062             else if (sort_command == 'N')
00063                 player_list.sort(Player::compare_name);
00064             else
00065             {
00066                 std::cout « "ERRO: comando inexistente" « std::endl;
00067                 continue;
00068             }
00069             std::list<Player>::iterator it;
00070             for (it = player_list.begin(); it != player_list.end(); it++)
00071                 it->print_player();
00072 
00073             continue;
00074         }
00075         else if (command == "CJ")
00076         {
00077             // Cadastrar um novo jogador
00078             std::string line_in;
00079             std::getline(std::cin, line_in);
00080             std::stringstream stream_in(line_in);
00081             stream_in » username_in;
00082             stream_in.ignore();
00083             std::getline(stream_in, name_in);
00084             if (name_in == "")
00085             {
00086                 std::cout « "ERRO: dados incorretos" « std::endl;
00087                 continue;
00088             }
00089             Player new_player(name_in, username_in);
00090             if (Player::register_player(new_player, player_list) == true)
00091                 std::cout « "Jogador " « new_player.get_username() « " cadastrado com sucesso" «
      std::endl;
00092             else
00093                 std::cout « "ERRO: jogador repetido" « std::endl;
00094             continue;
00095         }
00096         else if (command == "RJ")
00097         {
00098             // Remover um jogador existente
00099             std::cin » username_in;
00100             if (Player::remove_player(username_in, player_list) == true)
00101                 std::cout « "Jogador " « username_in « " removido com sucesso" « std::endl;
00102             else
00103                 std::cout « "ERRO: jogador inexistente" « std::endl;
00104             continue;
00105         }
00106         else if (command == "EP")
00107         {
00108             // Iniciar um jogo entre dois jogadores
```

```
00109             char game;
00110             std::string username_player1, username_player2;
00111 
00112             try
00113             {
00114                 std::cin » game;
00115                 if (game != 'R' && game != 'V' && game != 'L')
00116                 {
00117                     throw std::invalid_argument("Entrada inválida, jogos disponíveis: R, V e L");
00118                 }
00119 
00120                 std::cin » username_player1 » username_player2;
00121                 Player *player1 = find_player_in_list(player_list, username_player1);
00122                 Player *player2 = find_player_in_list(player_list, username_player2);
00123 
00124                 if (player1 == nullptr)
00125                     throw std::invalid_argument("ERRO: jogador " + username_player1 + " inexistente");
00126 
00127                 else if (player2 == nullptr)
00128                     throw std::invalid_argument("ERRO: jogador " + username_player2 + " inexistente");
00129 
00130                 // Inicialização do jogo com base no tipo selecionado
00131                 if (game == 'R')
00132                 {
00133                     Reversi reversi_game;
00134                     char player_piece = 'X';
00135                     char opponent_piece = 'O';
00136 
00137                     while (true)
00138                     {
00139                         int x, y;
00140                         bool is_there_movement_for_player =
      reversi_game.is_there_valid_move_for_player(player_piece);
00141                         bool someone_won = reversi_game.check_win(is_there_movement_for_player,
      player_piece);
00142                         reversi_game.print_reversi_board();
00143                         std::cout « "X: " « reversi_game.get_num_pieces_player_X() « " " « "O: "
00144                                  « reversi_game.get_num_pieces_player_O() « std::endl;
00145 
00146                         if (someone_won)
00147                         {
00148                             reversi_game.register_win_and_loss(player1, player2);
00149 
00150                             if (reversi_game.get_num_pieces_player_X() >
      reversi_game.get_num_pieces_player_O())
00151                                 std::cout « username_player1 « " ganhou!" « std::endl;
00152 
00153                             else if (reversi_game.get_num_pieces_player_X() <
      reversi_game.get_num_pieces_player_O())
00154                                 std::cout « username_player2 « " ganhou!" « std::endl;
00155 
00156                             else
00157                                 std::cout « "Houve empate!" « std::endl;
00158 
00159                             break;
00160                         }
00161                         else if (is_there_movement_for_player && !someone_won)
00162                         {
00163                             if (player_piece == 'X')
00164                                 std::cout « username_player1 « " " « "[X]" « ": " « std::ends;
00165                             else
00166                                 std::cout « username_player2 « " " « "[O]" « ": " « std::ends;
00167 
00168                             try {
00169 
00170                                 if (!(std::cin » x )) {
00171                                     std::cin.clear();
00172                                     std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
      '\n');
00173                                     throw std::invalid_argument("Entrada inválida. Por favor forneça
      dois números inteiros.");
00174                                 }
00175 
00176                                 if(!(std::cin » y)){
00177                                     std::cin.clear();
00178                                     std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
      '\n');
00179                                     throw std::invalid_argument("Entrada inválida, vez passada para o
      oponente");
00180                                 }
00181 
00182                                 if (!reversi_game.process_move({x, y}, player_piece)) {
00183                                     throw std::invalid_argument("Jogada inválida, vez passada para o
      oponente.");
00184                                 }
00185 
00186                             }
```

```
00187                             catch (const std::invalid_argument &e) {
00188
00189                                 std::cout « "Error: " « e.what() « std::endl;
00190                             }
00191
00192                             player_piece = reversi_game.switch_players(player_piece);
00193                             opponent_piece = reversi_game.switch_players(opponent_piece);
00194
00195                         }
00196                         else if (!is_there_movement_for_player && !someone_won)
00197                         {
00198                             player_piece = reversi_game.switch_players(player_piece);
00199                             opponent_piece = reversi_game.switch_players(opponent_piece);
00200                             std::cout « "Não há jogadas válidas, vez passada para o oponente" «
      std::endl;
00201                         }
00202                     }
00203                 }
00204                 else if (game == 'L')
00205                 {
00206                     Connect4 connect4_game;
00207                     bool game_over = false;
00208
00209                     while (!game_over)
00210                     {
00211                         int column;
00212                         connect4_game.print_game_board();
00213                         char current_player = connect4_game.get_current_player();
00214
00215                         std::cout « "Turno de jogador <" « current_player « ">:" « std::endl;
00216
00217                         try {
00218                             if (!(std::cin » column))
00219                             {
00220                                 std::cin.clear();
00221                                 std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00222                                 throw std::invalid_argument("Entrada inválida. Por favor, insira um
      número inteiro.");
00223                             }
00224
00225                             if (!connect4_game.is_valid_move(column))
00226                             {
00227                                 throw std::out_of_range("Movimento inválido!");
00228                             }
00229
00230                             connect4_game.make_move(column);
00231
00232                             if (connect4_game.check_win())
00233                             {
00234                                 connect4_game.print_game_board();
00235                                 if (current_player == 'X')
00236                                 {
00237                                     player1->add_win("Lig4");
00238                                     player2->add_loss("Lig4");
00239                                     std::cout « "Parabéns, " « username_player1 « "! Você venceu!" «
      std::endl;
00240                                 }
00241                                 else
00242                                 {
00243                                     player2->add_win("Lig4");
00244                                     player1->add_loss("Lig4");
00245                                     std::cout « "Parabéns, " « username_player2 « "! Você venceu!" «
      std::endl;
00246                                 }
00247                                 game_over = true;
00248                             }
00249                             else if (connect4_game.is_board_full())
00250                             {
00251                                 std::cout « "O jogo terminou em empate!" « std::endl;
00252                                 game_over = true;
00253                             }
00254                         }
00255                         catch (const std::out_of_range& e)
00256                         {
00257                             std::cout « "Erro: " « e.what() « std::endl;
00258                         }
00259                         catch (const std::runtime_error& e)
00260                         {
00261                             std::cout « "Erro: " « e.what() « std::endl;
00262                         }
00263                         catch (const std::invalid_argument& e)
00264                         {
00265                             std::cout « "Erro: " « e.what() « std::endl;
00266                         }
00267                         connect4_game.set_current_player((current_player == 'X') ? 'O' : 'X');
00268
00269                     }
```

```
00270                 }
00271                 else if (game == 'V')
00272                 {
00273                     Tic_tac_toe tic_tac_toe_game;
00274                     int x, y;
00275 
00276                     std::cout << username_player1 << " is X and " << username_player2 << " is O" <<
      std::endl;
00277 
00278                     while (true)
00279                     {
00280                         if (tic_tac_toe_game.check_tic_tac_toe_win() != 'F')
00281                         {
00282                             tic_tac_toe_game.print_tic_tac_toe_board();
00283                             std::cout << username_player1 << " won!" << std::endl;
00284                             player1->add_win("Velha");
00285                             player2->add_loss("Velha");
00286                             break;
00287                         }
00288 
00289                         if (tic_tac_toe_game.check_tie())
00290                         {
00291                             std::cout << "Draw! The board is full" << std::endl;
00292                             break;
00293                         }
00294 
00295                         std::cout << "Player " << tic_tac_toe_game.get_current_player() << " turn:" <<
      std::endl;
00296                         tic_tac_toe_game.print_tic_tac_toe_board();
00297 
00298                         try
00299                         {
00300                             if (!(std::cin >> x >> y))
00301                                 throw std::invalid_argument("Invalid input. Please enter two integers
      for your move");
00302                         }
00303 
00304                         catch (const std::invalid_argument &e)
00305                         {
00306                             std::cerr << "Error: " << e.what() << std::endl;
00307                             std::cin.clear();
00308                             std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00309                             continue;
00310                         }
00311 
00312                         tic_tac_toe_game.make_move(x, y);
00313                     }
00314                 }
00315             }
00316             catch (std::invalid_argument &e)
00317             {
00318                 std::cout << e.what() << std::endl;
00319             }
00320         }
00321         else if (command == "FS")
00322             break;
00323 
00324         else
00325         {
00326             if (error == false)
00327             {
00328                 std::cout << "ERRO: comando inexistente" << std::endl;
00329                 error = true;
00330                 continue;
00331             }
00332         }
00333     }
00334 
00335     // Escrita do arquivo de registro atualizado
00336     std::ofstream file_out;
00337     file_out.open("teste");
00338     if (!file_out.is_open())
00339     {
00340         std::cout << "Erro ao abrir o arquivo" << std::endl;
00341         return 1;
00342     }
00343 
00344     write_register_file(player_list, file_out);
00345 
00346     file_out.close();
00347 
00348     return 0;
00349 }
```

5.21 Referência do Arquivo src/Player.cpp

```
#include "Player.hpp"
#include <string.h>
#include <bits/stdc++.h>
```

Gráfico de dependência de inclusões para Player.cpp:

Funções

- void read_register_file (std::list< Player > &player_list, std::ifstream &file_in)

  Lê os dados de registro de jogadores de um arquivo.
- void write_register_file (std::list< Player > &player_list, std::ofstream &file_out)

  Escreve os dados de registro de jogadores em um arquivo.

5.21.1 Funções

5.21.1.1 read_register_file()

```
void read_register_file (
            std::list< Player > & player_list,
            std::ifstream & file_in )
```

Lê os dados de registro de jogadores de um arquivo.

Parâmetros

player_list	Lista de jogadores a ser preenchida.
file_in	Arquivo de entrada contendo os registros.

Definição na linha 116 do arquivo Player.cpp.

5.21.1.2 write_register_file()

```
void write_register_file (
            std::list< Player > & player_list,
            std::ofstream & file_out )
```

Escreve os dados de registro de jogadores em um arquivo.

Parâmetros

player_list	Lista de jogadores a ser registrada.
file_out	Arquivo de saída onde os registros serão armazenados.

Definição na linha 138 do arquivo Player.cpp.

5.22 Player.cpp

Ir para a documentação desse arquivo.

```
00001 #include "Player.hpp"
00002 #include <string.h>
00003 #include <bits/stdc++.h>
00004
00005
00006 Player::Player():
00007     Player("", "", {{"Reversi", 0}, {"Lig4", 0}, {"Velha", 0}} , {{"Reversi", 0}, {"Lig4", 0},
      {"Velha", 0}}) {};
00008
00009 Player::Player(std::string name_received, std::string username_received):
00010     Player(name_received, username_received, {{"Reversi", 0}, {"Lig4", 0}, {"Velha", 0}} ,
      {{"Reversi", 0}, {"Lig4", 0}, {"Velha", 0}}) {};
00011
00012 Player::Player(std::string name_received, std::string username_received, std::map<std::string, int>
      num_win_received, std::map<std::string, int> num_loss_received):
00013    name(name_received), username(username_received), num_win(num_win_received),
      num_loss(num_loss_received) {};
00014
00015 void Player::set_name(std::string name_received){
00016     this->name = name_received;
00017 }
00018
00019 void Player::set_username(std::string username_received){
00020     this->username = username_received;
00021 }
00022
00023 void Player::set_num_win(std::string key, int value){
00024     std::map<std::string, int>::iterator it = this->num_win.find(key);
00025     if (it == this->num_win.end())
00026         this->num_win.insert({key, value});
00027     else
00028         it->second = value;
00029 }
00030
00031 void Player::set_num_loss(std::string key, int value){
00032     std::map<std::string, int>::iterator it = this->num_loss.find(key);
00033     if (it == this->num_loss.end())
00034         this->num_loss.insert({key, value});
00035     else
00036         it->second = value;
00037 }
00038
00039 std::string Player::get_username(){
00040     return this->username;
00041 }
00042
00043 std::string Player::get_name(){
00044     return this->name;
00045 }
00046
00047 std::map<std::string, int> Player::get_num_win(){
00048     return this->num_win;
00049 }
00050
00051 std::map<std::string, int> Player::get_num_loss(){
00052     return this->num_loss;
00053 }
00054
00055 void Player::add_win(std::string key){
00056     std::map<std::string, int>::iterator it = this->num_win.find(key);
00057     if (it != this->num_win.end())
00058         it->second++;
00059 }
00060
00061 void Player::add_loss(std::string key){
00062     std::map<std::string, int>::iterator it = this->num_loss.find(key);
00063     if (it != this->num_loss.end())
00064         it->second++;
00065 }
00066
00067 void Player::print_player(){
00068     std::cout « this->username « " " « this->name « std::endl;
00069     std::cout « "REVERSI" « "\t" « "- V: " « this->num_win.find("Reversi")->second « " D: " «
      this->num_loss.find("Reversi")->second « std::endl;
00070     std::cout « "LIG4" « "\t" « "- V: " « this->num_win.find("Lig4")->second « " D: " «
      this->num_loss.find("Lig4")->second « std::endl;
00071     std::cout « "VELHA" « "\t" « "- V: " « this->num_win.find("Velha")->second « " D: " «
      this->num_loss.find("Velha")->second « std::endl;
00072 }
00073
00074 bool Player::register_player(Player player_received, std::list<Player> &player_list){
00075     std::list<Player>::iterator it;
```

```
00076     for (it = player_list.begin(); it != player_list.end(); it++){
00077         if (it->get_username() == player_received.get_username()){
00078             return false;
00079         }
00080     }
00081     player_list.push_back(player_received);
00082     return true;
00083 }
00084
00085 bool Player::remove_player(std::string username_received, std::list<Player> &player_list){
00086     std::list<Player>::iterator it;
00087     for (it = player_list.begin(); it != player_list.end(); it++){
00088         if (it->get_username() == username_received){
00089             it = player_list.erase(it);
00090             return true;
00091         }
00092     }
00093     return false;
00094 }
00095
00096 bool Player::compare_username(Player &player1, Player &player2){
00097     for (unsigned int i = 0; (i < player1.get_username().size()) && (i <
      player2.get_username().size()); i++){
00098         if (tolower(player1.get_username()[i]) < tolower(player2.get_username()[i]))
00099             return true;
00100         else if (tolower(player1.get_username()[i]) > tolower(player2.get_username()[i]))
00101             return false;
00102     }
00103     return player1.get_username().size() < player2.get_username().size();
00104 }
00105
00106 bool Player::compare_name(Player &player1, Player &player2){
00107     for (unsigned int i = 0; (i < player1.get_name().size()) && (i < player2.get_name().size()); i++){
00108         if (tolower(player1.get_name()[i]) < tolower(player2.get_name()[i]))
00109             return true;
00110         else if (tolower(player1.get_name()[i]) > tolower(player2.get_name()[i]))
00111             return false;
00112     }
00113     return player1.get_name().size() < player2.get_name().size();
00114 }
00115
00116 void read_register_file(std::list<Player> &player_list, std::ifstream &file_in) {
00117     std::string file_line[5];
00118     Player player_in;
00119     int i = 0;
00120     while (getline(file_in, file_line[i])) {
00121         i++;
00122         if (i == 5) {
00123         player_in.set_username(file_line[0]);
00124         player_in.set_name(file_line[1]);
00125         std::string key_in, num_win_in, num_loss_in;
00126         for (int j = 2; j < 5; j++){
00127             std::stringstream file_stream(file_line[j]);
00128             file_stream » key_in » num_win_in » num_loss_in;
00129             player_in.set_num_win(key_in, stoi(num_win_in));
00130             player_in.set_num_loss(key_in, stoi(num_loss_in));
00131         }
00132         player_list.push_back(player_in);
00133         i = 0;
00134         }
00135     }
00136 }
00137
00138 void write_register_file(std::list<Player> &player_list, std::ofstream &file_out){
00139     std::list<Player>::iterator it;
00140     for (it = player_list.begin(); it != player_list.end(); it++){
00141         file_out « it->get_username() « std::endl;
00142         file_out « it->get_name() « std::endl;
00143         file_out « "Reversi" « " " « it->get_num_win().find("Reversi")->second « " " «
      it->get_num_loss().find("Reversi")->second « std::endl;
00144         file_out « "Lig4" « " " « it->get_num_win().find("Lig4")->second « " " «
      it->get_num_loss().find("Lig4")->second « std::endl;
00145         file_out « "Velha" « " " « it->get_num_win().find("Velha")->second « " " «
      it->get_num_loss().find("Velha")->second « std::endl;
00146     }
00147 }
```

5.23 Referência do Arquivo src/Reversi.cpp

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

```
#include "list"
#include "array"
#include "Player.hpp"
```

Gráfico de dependência de inclusões para Reversi.cpp:

Variáveis

- const int num_columns_and_rows_reversi = 8

5.23.1 Variáveis

5.23.1.1 num_columns_and_rows_reversi

```
const int num_columns_and_rows_reversi = 8
```

Definição na linha 6 do arquivo Reversi.cpp.

5.24 Reversi.cpp

Ir para a documentação desse arquivo.

```
00001 #include "Reversi.hpp"
00002 #include <iostream>
00003 #include "list"
00004 #include "array"
00005 #include "Player.hpp"
00006 const int num_columns_and_rows_reversi = 8;
00007
00008 void Reversi::start_reversi_board()
00009 {
00010     this->game_board.set_space(3, 3, 'X');
00011     this->game_board.set_space(4, 4, 'X');
00012     this->game_board.set_space(3, 4, 'O');
00013     this->game_board.set_space(4, 3, 'O');
00014 }
00015
00016
00017 int Reversi::get_num_pieces_player_X()
00018 {
00019     return this->num_pieces_player_X;
00020 }
00021
00022
00023 int Reversi::get_num_pieces_player_O()
00024 {
00025     return this->num_pieces_player_O;
00026 }
00027
00028
00029 void Reversi::print_reversi_board() const
00030 {
00031     game_board.print_game_board();
00032 }
00033
00034
00035 Reversi::Reversi() : Game(num_columns_and_rows_reversi, num_columns_and_rows_reversi)
00036 {
00037     this->start_reversi_board();
00038     this->num_pieces_player_X = 2;
00039     this->num_pieces_player_O = 2;
00040 };
00041
00042
00043 bool Reversi::is_space_free_reversi(int x, int y) const
00044 {
00045     if (this->game_board.is_space_free(x, y) || this->game_board.get_space(x, y) == '*')
00046         return true;
00047     return false;
00048 }
```

```
00049
00050
00051 bool Reversi::is_there_player_piece_at_the_direction(const char player_piece,
00052     const std::array<int, 2>& direction, std::array<int, 2> adjacent_square) const
00053 {
00054     std::array<int, 2> current_square = { adjacent_square[0] + direction[0],
00055     adjacent_square[1] + direction[1] };
00056
00057     while (game_board.is_move_inside_board(current_square[0], current_square[1]))
00058     {
00059         if (this->game_board.get_space(current_square[0], current_square[1]) == player_piece)
00060             return true;
00061         else if (this->game_board.get_space(current_square[0], current_square[1]) == ' '
00062         || this->game_board.get_space(current_square[0], current_square[1]) == '*')
00063             return false;
00064
00065         current_square[0] += direction[0];
00066         current_square[1] += direction[1];
00067     }
00068     return false;
00069 }
00070
00071
00072 bool Reversi::is_there_direction_that_captures_opponent(const std::array<int, 2>
00073     & move_coordinates, char player_piece_type)
00074 {
00075     char opponent_player = switch_players(player_piece_type);
00076
00077     std::array<int, 2> adjacent_square = { 0, 0 };
00078
00079     for (int i = 1; i > -2; i--)
00080     {
00081         for (int j = 1; j > -2; j--)
00082         {
00083             if (j != 0 || i != 0)
00084             {
00085                 adjacent_square[0] = move_coordinates[0] + i;
00086                 adjacent_square[1] = move_coordinates[1] + j;
00087
00088                 if (this->game_board.is_move_inside_board(adjacent_square[0], adjacent_square[1]) &&
00089                     this->game_board.get_space(adjacent_square[0], adjacent_square[1]) ==
      opponent_player)
00090                     {
00091                     std::array<int, 2> direction = { i , j };
00092
00093                     if (is_there_player_piece_at_the_direction(player_piece_type, direction,
      adjacent_square))
00094                         return true;
00095                     }
00096             }
00097         }
00098     }
00099     return false;
00100 }
00101
00102
00103
00104
00105
00106 bool Reversi::is_valid_move(std::array<int, 2>& move_coordinates, char player_piece_type)
00107 {
00108
00109     if (!this->game_board.is_move_inside_board(move_coordinates[0], move_coordinates[1]))
00110         return false;
00111
00112     if (!this->is_space_free_reversi(move_coordinates[0], move_coordinates[1]))
00113         return false;
00114
00115     if (!this->is_there_direction_that_captures_opponent(move_coordinates, player_piece_type))
00116         return false;
00117
00118     return true;
00119
00120 }
00121
00122
00123 void Reversi::find_all_directions_to_make_move(std::array<int, 2>& move_coordinates,
00124     char player_piece, std::list<std::array<int, 2>>& directions_to_capture_opponents)
00125 {
00126     char opponent_player_piece = switch_players(player_piece);
00127
00128     std::array<int, 2> adjacent_square = { 0, 0 };
00129
00130     for (int i = 1; i > -2; i--)
00131     {
00132         for (int j = 1; j > -2; j--)
00133         {
```

```
00134             if (j != 0 || i != 0)
00135             {
00136                 adjacent_square[0] = move_coordinates[0] + i;
00137                 adjacent_square[1] = move_coordinates[1] + j;
00138 
00139                 if (this->game_board.is_move_inside_board(adjacent_square[0], adjacent_square[1]) &&
00140                     this->game_board.get_space(adjacent_square[0], adjacent_square[1]) ==
      opponent_player_piece)
00141                 {
00142                     std::array<int, 2> direction = { i , j };
00143 
00144                     if (is_there_player_piece_at_the_direction(player_piece, direction,
      adjacent_square))
00145                         directions_to_capture_opponents.push_back(direction);
00146 
00147                 }
00148             }
00149         }
00150     }
00151 }
00152 
00153 
00154 void Reversi::flip_pieces(std::array<int, 2> directions, std::array<int, 2> move_coordinates, char
      player_piece)
00155 {
00156     char opponent_piece = switch_players(player_piece);
00157     int num_pieces_flipped = 0;
00158 
00159     std::array<int, 2> current_square = { directions[0] + move_coordinates[0],
00160     directions[1] + move_coordinates[1] };
00161 
00162     while (this->game_board.get_space(current_square[0], current_square[1]) == opponent_piece)
00163     {
00164         this->game_board.set_space(current_square[0], current_square[1], player_piece);
00165         current_square[0] += directions[0];
00166         current_square[1] += directions[1];
00167         num_pieces_flipped++;
00168     }
00169     this->control_num_pieces_players(num_pieces_flipped, player_piece);
00170 }
00171 
00172 
00173 void Reversi::control_num_pieces_players(int num_pieces_flipped, char player_piece)
00174 {
00175     if (player_piece == 'X')
00176     {
00177         this->num_pieces_player_X += num_pieces_flipped;
00178         this->num_pieces_player_O -= num_pieces_flipped;
00179     }
00180     else
00181     {
00182         this->num_pieces_player_X -= num_pieces_flipped;
00183         this->num_pieces_player_O += num_pieces_flipped;
00184     }
00185 }
00186 
00187 
00188 
00189 
00190 
00191 void Reversi::make_move(std::array<int, 2> move_coordinates, char player_piece,
00192 std::list<std::array<int, 2>& directions_to_capture_opponents)
00193 {
00194 
00195     this->game_board.set_space(move_coordinates[0], move_coordinates[1], player_piece);
00196 
00197     if (player_piece == 'X')
00198         this->num_pieces_player_X++;
00199     else
00200         this->num_pieces_player_O++;
00201 
00202 
00203     for (auto direction : directions_to_capture_opponents)
00204         this->flip_pieces(direction, move_coordinates, player_piece);
00205 }
00206 
00207 
00208 bool Reversi::process_move(std::array<int, 2> move_coordinates, char player_piece)
00209 {
00210     move_coordinates[0] = move_coordinates[0] - 1;
00211     move_coordinates[1] = move_coordinates[1] - 1;
00212     std::list<std::array<int, 2>> directions_to_capture_opponents;
00213 
00214     if (this->is_valid_move(move_coordinates, player_piece))
00215     {
00216         find_all_directions_to_make_move(move_coordinates, player_piece,
      directions_to_capture_opponents);
```

```
00217         this->make_move(move_coordinates, player_piece, directions_to_capture_opponents);
00218         return true;
00219     }
00220     return false;
00221 }
00222
00223
00224 bool Reversi::is_there_valid_move_for_player(char player_piece)
00225 {
00226     bool found_valid_move = false;
00227     for (int i = 0; i < num_columns_and_rows_reversi; i++)
00228     {
00229         for (int j = 0; j < num_columns_and_rows_reversi; j++)
00230         {
00231             if (this->game_board.get_space(i, j) == ' ' || this->game_board.get_space(i, j) == '*')
00232             {
00233                 std::array<int, 2> coordinates = { i, j };
00234                 if (this->is_valid_move(coordinates, player_piece))
00235                 {
00236                     this->game_board.set_space(i, j,'*');
00237                     found_valid_move = true;
00238                 }
00239                 else
00240                 {
00241                     if (this->game_board.get_space(i, j) == '*')
00242                         this->game_board.set_space(i, j, ' ');
00243                 }
00244
00245             }
00246         }
00247     }
00248     return found_valid_move;
00249 }
00250
00251
00252 bool Reversi::check_win(bool is_there_move_for_player, char opponent_piece)
00253 {
00254     if (!is_there_move_for_player && !this->is_there_valid_move_for_player(opponent_piece))
00255         return true;
00256     return false;
00257 }
00258
00259
00260 void Reversi::register_win_and_loss(Player *player1, Player *player2)
00261 {
00262     if(this->num_pieces_player_X > this->num_pieces_player_O)
00263     {
00264         player1->add_win("Reversi");
00265         player2->add_loss("Reversi");
00266     }
00267     else if(this->num_pieces_player_X < this->num_pieces_player_O)
00268     {
00269         player2->add_win("Reversi");
00270         player1->add_loss("Reversi");
00271     }
00272 }
00273 Reversi::~Reversi() {};
00274
00275
00276 // Funções declaradas somente para fins de sobregarga
00277 bool Reversi::is_valid_move() const { return false; }
00278
00279 void Reversi::make_move() {}
00280
00281 bool Reversi::check_win() { return false; }
00282
```

5.25 Referência do Arquivo src/Tic_tac_toe.cpp

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

Gráfico de dependência de inclusões para Tic_tac_toe.cpp:

Variáveis

- const int num_rows_received = 3
- const int num_columns_received = 3

5.25.1 Variáveis

5.25.1.1 num_columns_received

```
const int num_columns_received = 3
```

Definição na linha 6 do arquivo Tic_tac_toe.cpp.

5.25.1.2 num_rows_received

```
const int num_rows_received = 3
```

Definição na linha 5 do arquivo Tic_tac_toe.cpp.

5.26 Tic_tac_toe.cpp

Ir para a documentação desse arquivo.

```
00001 #include "Tic_tac_toe.hpp"
00002 #include "Player.hpp"
00003 #include <iostream>
00004
00005 const int num_rows_received = 3;
00006 const int num_columns_received = 3;
00007
00008 Tic_tac_toe::Tic_tac_toe() : Game(num_rows_received, num_columns_received), current_player('X'),
      winner('F') {}
00009
00010
00011 bool Tic_tac_toe::is_valid_move(int& x, int& y) const
00012 {
00013     game_board.is_move_inside_board(x, y);
00014     game_board.is_space_free(x, y);
00015     return true;
00016 }
00017
00018
00019 void Tic_tac_toe::print_tic_tac_toe_board() const
00020 {
00021     game_board.print_game_board();
00022 }
00023
00024
00025 void Tic_tac_toe::make_move(int x, int y)
00026 {
00027     x -= 1;
00028     y -= 1;
00029
00030     try
00031     {
00032         if (is_valid_move(x, y))
00033         {
00034             game_board.set_space(x, y, current_player);
00035
00036             if (check_tic_tac_toe_win() != 'F')
00037                 winner = current_player;
00038
00039             else
00040                 current_player = switch_players(current_player);
00041         }
00042     }
00043
00044     catch (const std::out_of_range& e)
00045     {
00046         std::cout << "Error: " << e.what() << std::endl;
00047     }
00048
00049     catch(const std::runtime_error& e)
00050     {
00051         std::cout << "Error: " << e.what() << std::endl;
00052     }
00053
```

```
00054 }
00055
00056 char Tic_tac_toe::check_tic_tac_toe_win() const
00057 {
00058     // Verifica se há vitória nas linhas ou colunas
00059     for (int i = 0; i < 3; ++i) {
00060         if (game_board.get_space(i, 0) == current_player &&
00061             game_board.get_space(i, 1) == current_player &&
00062             game_board.get_space(i, 2) == current_player)
00063             return current_player;
00064
00065
00066         if (game_board.get_space(0, i) == current_player &&
00067             game_board.get_space(1, i) == current_player &&
00068             game_board.get_space(2, i) == current_player)
00069             return current_player;
00070
00071     }
00072
00073     // Verifica se há vitória nas diagonais
00074     if (game_board.get_space(0, 0) == current_player &&
00075         game_board.get_space(1, 1) == current_player &&
00076         game_board.get_space(2, 2) == current_player)
00077         return current_player;
00078
00079
00080     if (game_board.get_space(0, 2) == current_player &&
00081         game_board.get_space(1, 1) == current_player &&
00082         game_board.get_space(2, 0) == current_player)
00083         return current_player;
00084
00085     return 'F';
00086 }
00087
00088
00089 bool Tic_tac_toe::check_tie() const
00090 {
00091     for (int i = 0; i < 3; i++)
00092     {
00093         for (int j = 0; j < 3; j++)
00094         {
00095             if (game_board.get_space(i, j) == ' ')
00096                 return false;
00097         }
00098     }
00099
00100 return true;
00101 }
00102
00103
00104 char Tic_tac_toe::get_current_player() const
00105 {
00106     return current_player;
00107 }
00108
00109 Tic_tac_toe::~Tic_tac_toe() {}
00110
00111
00112 // Funções declaradas somente para fins de sobregarga.
00113 bool Tic_tac_toe::is_valid_move() const { return true; }
00114
00115 void Tic_tac_toe::make_move() {}
00116
00117 bool Tic_tac_toe::check_win() { return false; }
00118
```

Índice Remissivo