

PDS II Final Project

Gerado por Doxygen 1.9.8

1 Índice Hierárquico	1
1.1 Hierarquia de Classes	1
2 Índice dos Componentes	3
2.1 Lista de Classes	3
3 Índice dos Arquivos	5
3.1 Lista de Arquivos	5
4 Classes	7
4.1 Referência da Classe Board	7
4.1.1 Descrição detalhada	7
4.1.2 Construtores e Destrutores	8
4.1.2.1 Board()	8
4.1.3 Documentação das funções	8
4.1.3.1 get_space()	8
4.1.3.2 is_move_inside_board()	8
4.1.3.3 is_space_free()	9
4.1.3.4 print_game_board()	9
4.1.3.5 set_space()	9
4.1.4 Atributos	10
4.1.4.1 game_board	10
4.1.4.2 num_columns	10
4.1.4.3 num_rows	10
4.2 Referência da Classe Connect4	10
4.2.1 Descrição detalhada	11
4.2.2 Construtores e Destrutores	12
4.2.2.1 Connect4()	12
4.2.2.2 ~Connect4()	12
4.2.3 Documentação das funções	12
4.2.3.1 check_win()	12
4.2.3.2 get_current_player()	12
4.2.3.3 is_board_full()	12
4.2.3.4 is_valid_move() [1/2]	13
4.2.3.5 is_valid_move() [2/2]	13
4.2.3.6 make_move() [1/2]	13
4.2.3.7 make_move() [2/2]	13
4.2.3.8 print_game_board()	14
4.2.3.9 set_current_player()	14
4.2.4 Atributos	14
4.2.4.1 current_player	14
4.3 Referência da Classe Game	14
4.3.1 Descrição detalhada	15

4.3.2 Construtores e Destrutores	15
4.3.2.1 Game()	15
4.3.2.2 ~Game()	16
4.3.3 Documentação das funções	16
4.3.3.1 check_win()	16
4.3.3.2 is_valid_move()	16
4.3.3.3 make_move()	16
4.3.3.4 switch_players()	16
4.3.4 Atributos	17
4.3.4.1 game_board	17
4.4 Referência da Classe Player	17
4.4.1 Descrição detalhada	18
4.4.2 Construtores e Destrutores	18
4.4.2.1 Player() [1/3]	18
4.4.2.2 Player() [2/3]	19
4.4.2.3 Player() [3/3]	19
4.4.3 Documentação das funções	19
4.4.3.1 add_loss()	19
4.4.3.2 add_win()	20
4.4.3.3 compare_name()	20
4.4.3.4 compare_username()	20
4.4.3.5 get_name()	21
4.4.3.6 get_num_loss()	21
4.4.3.7 get_num_win()	21
4.4.3.8 get_username()	21
4.4.3.9 print_player()	21
4.4.3.10 register_player()	21
4.4.3.11 remove_player()	22
4.4.3.12 set_name()	22
4.4.3.13 set_num_loss()	22
4.4.3.14 set_num_win()	24
4.4.3.15 set_username()	24
4.4.4 Atributos	24
4.4.4.1 name	24
4.4.4.2 num_loss	24
4.4.4.3 num_win	25
4.4.4.4 username	25
4.5 Referência da Classe Reversi	25
4.5.1 Descrição detalhada	27
4.5.2 Construtores e Destrutores	27
4.5.2.1 Reversi()	27
4.5.2.2 ~Reversi()	27

4.5.3 Documentação das funções	27
4.5.3.1 check_win() [1/2]	27
4.5.3.2 check_win() [2/2]	27
4.5.3.3 control_num_pieces_players()	28
4.5.3.4 find_all_directions_to_make_move()	28
4.5.3.5 flip_pieces()	28
4.5.3.6 get_num_pieces_player_O()	29
4.5.3.7 get_num_pieces_player_X()	29
4.5.3.8 indicate_all_direction_to_make_move()	29
4.5.3.9 is_space_free_reversi()	29
4.5.3.10 is_there_direction_that_captures_opponent()	30
4.5.3.11 is_there_player_piece_at_the_direction()	30
4.5.3.12 is_there_valid_move_for_player()	31
4.5.3.13 is_valid_move() [1/2]	31
4.5.3.14 is_valid_move() [2/2]	31
4.5.3.15 make_move() [1/2]	32
4.5.3.16 make_move() [2/2]	32
4.5.3.17 print_reversi_board()	32
4.5.3.18 process_move()	32
4.5.3.19 register_win_and_loss()	33
4.5.3.20 start_reversi_board()	33
4.5.4 Atributos	33
4.5.4.1 num_pieces_player_O	33
4.5.4.2 num_pieces_player_X	34
4.6 Referência da Classe Tic_tac_toe	34
4.6.1 Descrição detalhada	35
4.6.2 Construtores e Destrutores	35
4.6.2.1 Tic_tac_toe()	35
4.6.2.2 ~Tic_tac_toe()	35
4.6.3 Documentação das funções	36
4.6.3.1 check_tic_tac_toe_win()	36
4.6.3.2 check_tie()	36
4.6.3.3 check_win()	36
4.6.3.4 get_current_player()	36
4.6.3.5 is_valid_move() [1/2]	37
4.6.3.6 is_valid_move() [2/2]	37
4.6.3.7 make_move() [1/2]	37
4.6.3.8 make_move() [2/2]	37
4.6.3.9 print_tic_tac_toe_board()	38
4.6.4 Atributos	38
4.6.4.1 current_player	38
4.6.4.2 winner	38

5 Arquivos	39
5.1 Referência do Arquivo include/Board.hpp	39
5.1.1 Descrição detalhada	39
5.2 Board.hpp	39
5.3 Referência do Arquivo include/Connect4.hpp	40
5.3.1 Descrição detalhada	40
5.4 Connect4.hpp	40
5.5 Referência do Arquivo include/Game.hpp	41
5.5.1 Descrição detalhada	41
5.6 Game.hpp	41
5.7 Referência do Arquivo include/Player.hpp	41
5.7.1 Descrição detalhada	42
5.7.2 Funções	42
5.7.2.1 read_register_file()	42
5.7.2.2 write_register_file()	42
5.8 Player.hpp	43
5.9 Referência do Arquivo include/Reversi.hpp	44
5.9.1 Descrição detalhada	44
5.10 Reversi.hpp	44
5.11 Referência do Arquivo include/Tic_tac_toe.hpp	45
5.11.1 Descrição detalhada	45
5.12 Tic_tac_toe.hpp	46
5.13 Referência do Arquivo src/Board.cpp	46
5.14 Board.cpp	46
5.15 Referência do Arquivo src/Connect4.cpp	47
5.16 Connect4.cpp	47
5.17 Referência do Arquivo src/Game.cpp	49
5.18 Game.cpp	49
5.19 Referência do Arquivo src/main.cpp	49
5.19.1 Funções	49
5.19.1.1 find_player_in_list()	49
5.19.1.2 main()	50
5.20 main.cpp	50
5.21 Referência do Arquivo src/Player.cpp	54
5.21.1 Funções	54
5.21.1.1 read_register_file()	54
5.21.1.2 write_register_file()	55
5.22 Player.cpp	55
5.23 Referência do Arquivo src/Reversi.cpp	57
5.23.1 Variáveis	57
5.23.1.1 num_columns_and_rows_reversi	57
5.24 Reversi.cpp	57

5.25 Referência do Arquivo src/Tic_tac_toe.cpp	60
5.25.1 Variáveis	61
5.25.1.1 num_columns_received	61
5.25.1.2 num_rows_received	61
5.26 Tic_tac_toe.cpp	61
Índice Remissivo	63

Capítulo 1

Índice Hierárquico

1.1 Hierarquia de Classes

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

Board	7
Game	14
Connect4	10
Reversi	25
Tic_tac_toe	34
Player	17

Capítulo 2

Índice dos Componentes

2.1 Lista de Classes

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

Board	Gerencia o tabuleiro do jogo	7
Connect4	Gerencia as regras e funcionalidades do jogo Connect4	10
Game	Classe base para jogos com tabuleiro	14
Player	Representa um jogador com informações pessoais e estatísticas	17
Reversi	Gerencia as regras e funcionalidades do jogo Reversi	25
Tic_tac_toe	Gerencia as regras e funcionalidades do Jogo da Velha	34

Capítulo 3

Índice dos Arquivos

3.1 Lista de Arquivos

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

include/ Board.hpp	
Representa o tabuleiro de um jogo genérico	39
include/ Connect4.hpp	
Implementa o jogo Connect4 (Lig4), baseado na classe genérica Game	40
include/ Game.hpp	
Classe base abstrata para jogos genéricos com tabuleiro	41
include/ Player.hpp	
Gerencia informações e ações relacionadas a jogadores	41
include/ Reversi.hpp	
Implementa o jogo Reversi , baseado na classe genérica Game	44
include/ Tic_tac_toe.hpp	
Implementa o Jogo da Velha (Tic Tac Toe), baseado na classe genérica Game	45
src/ Board.cpp	46
src/ Connect4.cpp	47
src/ Game.cpp	49
src/ main.cpp	49
src/ Player.cpp	54
src/ Reversi.cpp	57
src/ Tic_tac_toe.cpp	60

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

<i>num_rows_received</i>	Número de linhas.
<i>num_columns_received</i>	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

<i>row</i>	Linha do tabuleiro.
<i>column</i>	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

<i>x</i>	Coordenada da linha.
<i>y</i>	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

<i>x</i>	Coordenada da linha.
<i>y</i>	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

<i>row</i>	Linha do tabuleiro.
<i>column</i>	Coluna do tabuleiro.
<i>value</i>	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 117 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 34 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 87 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 93 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 120 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

<i>column</i>	Coluna onde o jogador deseja jogar.
---------------	-------------------------------------

Retorna

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

Definição na linha 8 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 122 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

<i>column</i>	Coluna onde o jogador deseja jogar.
---------------	-------------------------------------

Definição na linha 19 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 105 do arquivo [Connect4.cpp](#).

4.2.3.9 set_current_player()

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

Define o jogador atual.

Parâmetros

<i>player</i>	Caractere representando o jogador ('X' ou 'O').
---------------	---

Definição na linha 111 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

<code>current_player</code>	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

Representa um jogador com informações pessoais e estatísticas.

```
#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, mantendo estatísticas zeradas.
- [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 recebidas.
- void [set_name](#) (std::string name_received)
Define o nome do jogador.
- void [set_username](#) (std::string username_received)
Define o nome de usuário.
- void [set_num_win](#) (std::string key, int value)
Atualiza o número de vitórias para um jogo.
- void [set_num_loss](#) (std::string key, int value)
Atualiza o número de derrotas para um jogo.

- `std::string get_username ()`
Retorna o nome de usuário.
- `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.
- `void add_win (std::string key)`
Adiciona uma vitória ao jogador em um jogo específico.
- `void add_loss (std::string key)`
Adiciona uma derrota ao jogador em um jogo específico.

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 o username de dois jogadores.
- `static bool compare_name (Player &player1, Player &player2)`
Compara o nome 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

Representa um jogador com informações pessoais e estatísticas.

Oferece métodos para gerenciar dados como nome, username, vitórias e derrotas, além de funcionalidades estáticas para manipular listas de jogadores.

Definição na linha 23 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, mantendo estatísticas zeradas.

Parâmetros

<i>name_received</i>	Nome do jogador.
<i>username_received</i>	Nome de usuário.

Definição na linha 10 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 recebidas.

Parâmetros

<i>name_received</i>	Nome do jogador.
<i>username_received</i>	Nome de usuário.
<i>num_win_received</i>	Mapa de vitórias por jogo.
<i>num_loss_received</i>	Mapa de derrotas por jogo.

Definição na linha 14 do arquivo [Player.cpp](#).

4.4.3 Documentação das funções

4.4.3.1 add_loss()

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

Adiciona uma derrota ao jogador em um jogo específico.

Parâmetros

<i>key</i>	Nome do jogo.
------------	---------------

Definição na linha 46 do arquivo [Player.cpp](#).

4.4.3.2 add_win()

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

Adiciona uma vitória ao jogador em um jogo específico.

Parâmetros

<i>key</i>	Nome do jogo.
------------	---------------

Definição na linha 38 do arquivo [Player.cpp](#).

4.4.3.3 compare_name()

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

Compara o nome de dois jogadores.

Parâmetros

<i>player1</i>	Primeiro jogador.
<i>player2</i>	Segundo jogador.

Retorna

`true` se os nomes forem iguais, `false` caso contrário.

4.4.3.4 compare_username()

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

Compara o username de dois jogadores.

Parâmetros

<i>player1</i>	Primeiro jogador.
<i>player2</i>	Segundo jogador.

Retorna

`true` se os usernames forem iguais, `false` caso contrário.

4.4.3.5 get_name()

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

Retorna o nome do jogador.

Retorna

Nome do jogador.

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.

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.

4.4.3.8 get_username()

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

Retorna o nome de usuário.

Retorna

Nome de usuário do jogador.

4.4.3.9 print_player()

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

Imprime as informações do jogador.

Definição na linha 54 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

<i>player_received</i>	Jogador a ser registrado.
<i>player_list</i>	Lista onde o jogador será registrado.

Retorna

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

Definição na linha 63 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

<i>username_received</i>	Nome de usuário do jogador a ser removido.
<i>player_list</i>	Lista de onde o jogador será removido.

Retorna

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

Definição na linha 76 do arquivo [Player.cpp](#).

4.4.3.12 set_name()

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

Define o nome do jogador.

Parâmetros

<i>name_received</i>	Nome do jogador.
----------------------	------------------

4.4.3.13 set_num_loss()

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

Atualiza o número de derrotas para um jogo.

Parâmetros

<i>key</i>	Nome do jogo.
<i>value</i>	Número de derrotas.

Definição na linha 28 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 para um jogo.

Parâmetros

<i>key</i>	Nome do jogo.
<i>value</i>	Número de vitórias.

Definição na linha 18 do arquivo [Player.cpp](#).

4.4.3.15 set_username()

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

Define o nome de usuário.

Parâmetros

<i>username_received</i>	Nome de usuário.
--------------------------	------------------

4.4.4 Atributos**4.4.4.1 name**

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

Nome do jogador.

Definição na linha 25 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 28 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 27 do arquivo [Player.hpp](#).

4.4.4.4 username

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

Nome de usuário do jogador.

Definição na linha 26 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

<i>is_there_move_for_player</i>	Indica se há jogadas válidas para o jogador.
<i>opponent_piece</i>	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

<i>num_pieces_flipped</i>	Número de peças capturadas.
<i>player_piece</i>	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

<i>move_coordinates</i>	Coordenadas do movimento.
<i>player_piece</i>	Tipo de peça do jogador.
<i>directions_to_capture_opponents</i>	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

<i>directions</i>	Direção onde as peças serão capturadas.
<i>move_coordinates</i>	Coordenadas do movimento.
<i>player_piece</i>	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

<i>x</i>	Linha da posição.
<i>y</i>	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

<i>move_coordinates</i>	Coordenadas do movimento.
<i>player_piece_type</i>	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

<i>player_piece</i>	Tipo de peça do jogador.
<i>direction</i>	Direção a ser verificada.
<i>adjacent_square</i>	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

<i>player_piece</i>	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

<i>move_coordinates</i>	Coordenadas do movimento.
<i>player_piece_type</i>	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

<i>move_coordinates</i>	Coordenadas do movimento.
<i>player_piece</i>	Tipo de peça do jogador.
<i>directions_to_capture_opponents</i>	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

<i>move_coordinates</i>	Coordenadas do movimento.
<i>player_piece_type</i>	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

<i>player1</i>	Jogador 1.
<i>player2</i>	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

<i>x</i>	Coordenada da linha.
<i>y</i>	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](#)
Representa um jogador com informações pessoais e estatísticas.

Funções

- void [read_register_file](#) (std::list< [Player](#) > &player_list, std::ifstream &file_in)
Lê os dados de registro de jogadores a partir 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, registros e funcionalidades associadas.

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 a partir de um arquivo.

Parâmetros

<i>player_list</i>	Lista de jogadores a ser preenchida.
<i>file_in</i>	Arquivo de entrada contendo os registros.

Definição na linha 91 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

<i>player_list</i>	Lista de jogadores a ser registrada.
<i>file_out</i>	Arquivo de saída para armazenar os registros.

Definição na linha 115 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
00023 class Player {
00024     private:
00025         std::string name;
00026         std::string username;
00027         std::map<std::string, int> num_win;
00028         std::map<std::string, int> num_loss;
00030     public:
00034         Player();
00035
00041         Player(std::string name_received, std::string username_received);
00042
00043
00051         Player(std::string name_received, std::string username_received,
00052             std::map<std::string, int> num_win_received, std::map<std::string, int>
num_loss_received);
00053
00054
00059         void set_name(std::string name_received);
00060
00061
00066         void set_username(std::string username_received);
00067
00068
00074         void set_num_win(std::string key, int value);
00075
00076
00082         void set_num_loss(std::string key, int value);
00083
00084
00089         std::string get_username();
00090
```

```

00091
00096     std::string get_name();
00097
00098
00103     std::map<std::string, int> get_num_win();
00104
00105
00110     std::map<std::string, int> get_num_loss();
00111
00112
00116     void print_player();
00117
00118
00123     void add_win(std::string key);
00124
00125
00130     void add_loss(std::string key);
00131
00132
00139     static bool register_player(Player player_received, std::list<Player> &player_list);
00140
00141
00148     static bool remove_player(std::string username_received, std::list<Player> &player_list);
00149
00150
00157     static bool compare_username(Player &player1, Player &player2);
00158
00159
00166     static bool compare_name(Player &player1, Player &player2);
00167 };
00168
00174 void read_register_file(std::list<Player> &player_list, std::ifstream &file_in);
00175
00176
00182 void write_register_file(std::list<Player> &player_list, std::ofstream &file_out);
00183
00184 #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,
std::list<std::array<int, 2>&
00119             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,
std::list<std::array<int, 2>& directions_to_capture_opponents);
00152
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
00008 bool Connect4::is_valid_move(int column) {
00009     column--;
00010     int rows = 6;
00011     // Verifica se a coluna está cheia ou fora dos limites do tabuleiro
00012     if (game_board.get_space(0, column) != ' ' || game_board.is_move_inside_board(rows, column)) {
00013         return false;
00014     }
00015     return true;
00016 }
00017
00018
00019 void Connect4::make_move(int column)
00020 {
00021     column--;
00022     // Encontra a linha mais baixa disponível na coluna e coloca a peça do jogador atual
00023     for(int i = 5; i >= 0; i--)
00024     {
00025         if(game_board.get_space(i, column) == ' ')
00026         {
00027             game_board.set_space(i, column, current_player);
00028             return;
00029         }
00030     }
00031 }
00032
00033
00034 bool Connect4::check_win()
00035 {
00036     // Verificação de vitória horizontal
00037     for (int row = 0; row < 6; ++row) {
00038         for (int col = 0; col <= 3; ++col) {
00039             if (game_board.get_space(row, col) == current_player &&
00040                 game_board.get_space(row, col + 1) == current_player &&
00041                 game_board.get_space(row, col + 2) == current_player &&
00042                 game_board.get_space(row, col + 3) == current_player) {
00043                 return true;
00044             }
00045         }
00046     }
00047
00048     // Verificação de vitória vertical
00049     for (int row = 0; row <= 2; ++row) {
00050         for (int col = 0; col < 7; ++col) {
00051             if (game_board.get_space(row, col) == current_player &&
00052                 game_board.get_space(row + 1, col) == current_player &&
00053                 game_board.get_space(row + 2, col) == current_player &&
00054                 game_board.get_space(row + 3, col) == current_player) {
00055                 return true;
00056             }
00057         }
00058     }
00059
00060     // Verificação de vitória diagonal para a direita
00061     for (int row = 0; row <= 2; ++row) {
00062         for (int col = 0; col <= 3; ++col) {
00063             if (game_board.get_space(row, col) == current_player &&
00064                 game_board.get_space(row + 1, col + 1) == current_player &&
00065                 game_board.get_space(row + 2, col + 2) == current_player &&
00066                 game_board.get_space(row + 3, col + 3) == current_player) {
00067                 return true;
00068             }
00069         }
00070     }
00071 }
```



```

00070     }
00071
00072     // Verificação de vitória diagonal para a esquerda
00073     for (int row = 3; row < 6; ++row) {
00074         for (int col = 0; col <= 3; ++col) {
00075             if (game_board.get_space(row, col) == current_player &&
00076                 game_board.get_space(row - 1, col + 1) == current_player &&
00077                 game_board.get_space(row - 2, col + 2) == current_player &&
00078                 game_board.get_space(row - 3, col + 3) == current_player) {
00079                 return true;
00080             }
00081         }
00082     }
00083     return false;
00084 }
00085
00086
00087 char Connect4::get_current_player()
00088 {
00089     return current_player;
00090 }
00091
00092
00093 bool Connect4::is_board_full() const
00094 {
00095     for(int col = 0; col < 7; ++col)
00096     {
00097         if(game_board.get_space(0, col) == ' ') {
00098             return false;
00099         }
00100     }
00101     return true;
00102 }
00103
00104
00105 void Connect4::print_game_board() const
00106 {
00107     game_board.print_game_board();
00108 }
00109
00110
00111 void Connect4::set_current_player(char player)
00112 {
00113     current_player = player;
00114 }
00115
00116
00117 Connect4::~Connect4() {}
00118
00119 // Funções declaradas somente para fins de sobrecarga
00120 bool Connect4::is_valid_move() const { return true; }
00121
00122 void Connect4::make_move() {}

```

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

<code>player_list</code>	Lista que registra todos os jogadores.
<code>user_name</code>	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/julialuna/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 =
00141                         reversi_game.is_there_valid_move_for_player(player_piece);
00142                     bool someone_won = reversi_game.check_win(is_there_movement_for_player,
00143                         player_piece);
00144                     reversi_game.print_reversi_board();
00145                     std::cout << "X: " << reversi_game.get_num_pieces_player_X() << " " << "O: "
00146                         << reversi_game.get_num_pieces_player_O() << std::endl;
00147
00148                     if (someone_won)
00149                     {
00150                         reversi_game.register_win_and_loss(player1, player2);
00151
00152                         if (reversi_game.get_num_pieces_player_X() >
00153                             reversi_game.get_num_pieces_player_O())
00154                             std::cout << username_player1 << " ganhou!" << std::endl;
00155
00156                         else if (reversi_game.get_num_pieces_player_X() <
00157                             reversi_game.get_num_pieces_player_O())
00158                             std::cout << username_player2 << " ganhou!" << std::endl;
00159
00160                         else
00161                             std::cout << "Houve empate!" << std::endl;
00162
00163                         break;
00164                     }
00165                     else if (is_there_movement_for_player && !someone_won)
00166                     {
00167                         if (player_piece == 'X')
00168                             std::cout << username_player1 << " " << "[X]" << ": " << std::ends;
00169                         else
00170                             std::cout << username_player2 << " " << "[O]" << ": " << std::ends;
00171
00172                         try {
00173                             if (!(std::cin >> x)) {
00174                                 std::cin.clear();
00175                                 std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
00176                                     '\n');
00177                                 throw std::invalid_argument("Entrada inválida. Por favor forneça
00178                                     dois números inteiros.");
00179                             }
00180                         }
00181                     }
00182                 }
00183             }
00184         }
00185     }

```

```

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 = 'X';
00214
00215                                             std::cout << "Turno de jogador <" << connect4_game.get_current_player() << ">:" <<
std::endl;
00216
00217                                             std::cin >> column;
00218
00219                                             if (connect4_game.is_valid_move(column))
00220                                             {
00221                                                 connect4_game.make_move(column);
00222                                                 if (connect4_game.check_win())
00223                                                 {
00224                                                     connect4_game.print_game_board();
00225                                                     if (connect4_game.get_current_player() == 'X')
00226                                                     {
00227                                                         player1->add_win("Lig4");
00228                                                         player2->add_loss("Lig4");
00229                                                         std::cout << "Parabéns, " << username_player1 << "! Você venceu!" <<
std::endl;
00230                                                     }
00231                                                     else
00232                                                     {
00233                                                         player2->add_win("Lig4");
00234                                                         player1->add_loss("Lig4");
00235                                                         std::cout << "Parabéns, " << username_player2 << "! Você venceu!" <<
std::endl;
00236                                                     }
00237                                                     game_over = true;
00238                                                 }
00239                                                 else if (connect4_game.is_board_full())
00240                                                 {
00241                                                     std::cout << "O jogo terminou em empate!" << std::endl;
00242                                                     game_over = true;
00243                                                 }
00244
00245                                             connect4_game.switch_players(current_player);
00246                                         }
00247                                         else
00248                                         {
00249                                             std::cout << "Movimento inválido. Tente novamente." << std::endl;
00250                                         }
00251                                     }
00252                                     else if (game == 'V')
00253                                     {
00254                                         Tic_tac_toe tic_tac_toe_game;
00255                                         int x, y;
00256                                     }

```

```

00255         std::cout << username_player1 << " is X and " << username_player2 << " is O" <<
00256         std::endl;
00257         while (true)
00258         {
00259             if (tic_tac_toe_game.check_tic_tac_toe_win() != 'F')
00260             {
00261                 tic_tac_toe_game.print_tic_tac_toe_board();
00262                 std::cout << username_player1 << " won!" << std::endl;
00263                 player1->add_win("Velha");
00264                 player2->add_loss("Velha");
00265                 break;
00266             }
00267             if (tic_tac_toe_game.check_tie())
00268             {
00269                 std::cout << "Draw! The board is full" << std::endl;
00270                 break;
00271             }
00272             std::cout << "Player " << tic_tac_toe_game.get_current_player() << " turn:" <<
00273             std::endl;
00274             tic_tac_toe_game.print_tic_tac_toe_board();
00275             try
00276             {
00277                 if (!(std::cin > x > y))
00278                     throw std::invalid_argument("Invalid input. Please enter two integers
00279 for your move");
00280             }
00281             catch (const std::invalid_argument &e)
00282             {
00283                 std::cerr << "Error: " << e.what() << std::endl;
00284                 std::cin.clear();
00285                 std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00286                 continue;
00287             }
00288             tic_tac_toe_game.make_move(x, y);
00289         }
00290     }
00291     catch (std::invalid_argument &e)
00292     {
00293         std::cout << e.what() << std::endl;
00294     }
00295     else if (command == "FS")
00296     {
00297         break;
00298     }
00299     else
00300     {
00301         if (error == false)
00302         {
00303             std::cout << "ERRO: comando inexistente" << std::endl;
00304             error = true;
00305             continue;
00306         }
00307     }
00308 }
00309 // Escrita do arquivo de registro atualizado
00310 std::ofstream file_out;
00311 file_out.open("teste");
00312 if (!file_out.is_open())
00313 {
00314     std::cout << "Erro ao abrir o arquivo" << std::endl;
00315     return 1;
00316 }
00317 write_register_file(player_list, file_out);
00318 file_out.close();
00319 return 0;
00320 }

```

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 a partir 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 a partir de um arquivo.

Parâmetros

<code>player_list</code>	Lista de jogadores a ser preenchida.
<code>file_in</code>	Arquivo de entrada contendo os registros.

Definição na linha 91 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

<code>player_list</code>	Lista de jogadores a ser registrada.
<code>file_out</code>	Arquivo de saída para armazenar os registros.

Definição na linha 115 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},
00008     {"Velha", 0}) {}
00009
00010 Player::Player(std::string name_received, std::string username_received):
00011     Player(name_received, username_received, {"Reversi", 0}, {"Lig4", 0}, {"Velha", 0},
00012     {"Reversi", 0}, {"Lig4", 0}, {"Velha", 0}) {}
00013
00014 Player::Player(std::string name_received, std::string username_received, std::map<std::string, int>
00015     num_win_received, std::map<std::string, int> num_loss_received):
00016     name(name_received), username(username_received), num_win(num_win_received),
00017     num_loss(num_loss_received) {}
00018
00019 void Player::set_num_win(std::string key, int value)
00020 {
00021     std::map<std::string, int>::iterator it = this->num_win.find(key);
00022     if (it == this->num_win.end())
00023         this->num_win.insert({key, value});
00024     else
00025         it->second = value;
00026 }
00027
00028 void Player::set_num_loss(std::string key, int value)
00029 {
00030     std::map<std::string, int>::iterator it = this->num_loss.find(key);
00031     if (it == this->num_loss.end())
00032         this->num_loss.insert({key, value});
00033     else
00034         it->second = value;
00035 }
00036
00037 void Player::add_win(std::string key)
00038 {
00039     std::map<std::string, int>::iterator it = this->num_win.find(key);
00040     if (it != this->num_win.end())
00041         it->second++;
00042 }
00043
00044 void Player::add_loss(std::string key)
00045 {
00046     std::map<std::string, int>::iterator it = this->num_loss.find(key);
00047     if (it != this->num_loss.end())
00048         it->second++;
00049 }
00050
00051 void Player::print_player()
00052 {
00053     std::cout << this->username << " " << this->name << std::endl;
00054     std::cout << "REVERSI" << "\t" << "- V: " << this->num_win.find("Reversi")->second << " D: " <<
00055     this->num_loss.find("Reversi")->second << std::endl;
00056     std::cout << "LIG4" << "\t" << "- V: " << this->num_win.find("Lig4")->second << " D: " <<
00057     this->num_loss.find("Lig4")->second << std::endl;
00058     std::cout << "VELHA" << "\t" << "- V: " << this->num_win.find("Velha")->second << " D: " <<
00059     this->num_loss.find("Velha")->second << std::endl;
00060 }
00061
00062
00063 bool Player::register_player(Player player_received, std::list<Player> &player_list)
00064 {
00065     std::list<Player>::iterator it;
00066     for (it = player_list.begin(); it != player_list.end(); it++)
00067     {
00068         if (it->get_username() == player_received.get_username())
00069             return false;
00070     }
00071     player_list.push_back(player_received);
00072     return true;
00073 }
00074
00075
00076 bool Player::remove_player(std::string username_received, std::list<Player> &player_list)
00077 {
00078     std::list<Player>::iterator it;
00079     for (it = player_list.begin(); it != player_list.end(); it++)
00080     {
00081         if (it->get_username() == username_received)

```



```

00082     {
00083         it = player_list.erase(it);
00084         return true;
00085     }
00086 }
00087 return false;
00088 }
00089
00090
00091 void read_register_file(std::list<Player> &player_list, std::ifstream &file_in)
00092 {
00093     std::string file_line[5];
00094     Player player_in;
00095     int i = 0;
00096     while (getline(file_in, file_line[i])) {
00097         i++;
00098         if (i == 5) {
00099             player_in.set_username(file_line[0]);
00100             player_in.set_name(file_line[1]);
00101             std::string key_in, num_win_in, num_loss_in;
00102             for (int j = 2; j < 5; j++){
00103                 std::stringstream file_stream(file_line[j]);
00104                 file_stream >> key_in >> num_win_in >> num_loss_in;
00105                 player_in.set_num_win(key_in, stoi(num_win_in));
00106                 player_in.set_num_loss(key_in, stoi(num_loss_in));
00107             }
00108             player_list.push_back(player_in);
00109             i = 0;
00110         }
00111     }
00112 }
00113
00114
00115 void write_register_file(std::list<Player> &player_list, std::ofstream &file_out)
00116 {
00117     std::list<Player>::iterator it;
00118     for (it = player_list.begin(); it != player_list.end(); it++){
00119         file_out << it->get_username() << std::endl;
00120         file_out << it->get_name() << std::endl;
00121         file_out << "Reversi" << " " << it->get_num_win().find("Reversi")->second << " " <<
00122         it->get_num_loss().find("Reversi")->second << std::endl;
00123         file_out << "Lig4" << " " << it->get_num_win().find("Lig4")->second << " " <<
00124         it->get_num_loss().find("Lig4")->second << std::endl;
00125         file_out << "Velha" << " " << it->get_num_win().find("Velha")->second << " " <<
00126         it->get_num_loss().find("Velha")->second << std::endl;
00127     }
00128 }

```

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]) ==
00090                 opponent_player)
00091             {
00092                 std::array<int, 2> direction = { i , j };
00093                 if (is_there_player_piece_at_the_direction(player_piece_type, direction,
00094                     adjacent_square))
00095                     return true;
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     if (!this->game_board.is_move_inside_board(move_coordinates[0], move_coordinates[1]))
00109         return false;
00110
00111     if (!this->is_space_free_reversi(move_coordinates[0], move_coordinates[1]))
00112         return false;
00113
00114     if (!this->is_there_direction_that_captures_opponent(move_coordinates, player_piece_type))
00115         return false;
00116
00117     return true;
00118 }
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]) ==
00141                     opponent_player_piece)
00142                 {
00143                     std::array<int, 2> direction = { i , j };
00144                     if (is_there_player_piece_at_the_direction(player_piece, direction,
00145                         adjacent_square))
00146                         directions_to_capture_opponents.push_back(direction);
00147                 }
00148             }
00149         }
00150     }
00151 }
00152
00153
00154 void Reversi::flip_pieces(std::array<int, 2> directions, std::array<int, 2> move_coordinates, char
00155     player_piece)
00156 {
00157     char opponent_piece = switch_players(player_piece);
00158     int num_pieces_flipped = 0;
00159
00160     std::array<int, 2> current_square = { directions[0] + move_coordinates[0],
00161         directions[1] + move_coordinates[1] };
00162
00163     while (this->game_board.get_space(current_square[0], current_square[1]) == opponent_piece)
00164     {
00165         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,
00217 directions_to_capture_opponents);
00218         this->make_move(move_coordinates, player_piece, directions_to_capture_opponents);
00219         return true;
00220     }
00221     return false;
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     return found_valid_move;
00248 }
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         if (game_board.get_space(0, i) == current_player &&
00066             game_board.get_space(1, i) == current_player &&
00067             game_board.get_space(2, i) == current_player)
00068             return current_player;
00069     }
00070
00071     // Verifica se há vitória nas diagonais
00072     if (game_board.get_space(0, 0) == current_player &&
00073         game_board.get_space(1, 1) == current_player &&
00074         game_board.get_space(2, 2) == current_player)
00075         return current_player;
00076
00077     if (game_board.get_space(0, 2) == current_player &&
00078         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

- ~Game
 - Game, [11](#)
- ~Reversi
 - Reversi, [20](#)
- ~Tic_tac_toe
 - Tic_tac_toe, [26](#)
- add_loss
 - Player, [14](#)
- add_win
 - Player, [14](#)
- Board, [7](#)
 - Board, [7](#)
 - get_space, [8](#)
 - is_move_inside_board, [8](#)
 - is_space_free, [9](#)
 - print_game_board, [9](#)
 - set_space, [9](#)
- check_tic_tac_toe_win
 - Tic_tac_toe, [26](#)
- check_tie
 - Tic_tac_toe, [26](#)
- check_win
 - Game, [11](#)
 - Reversi, [20](#)
 - Tic_tac_toe, [27](#)
- compare_name
 - Player, [14](#)
- compare_username
 - Player, [15](#)
- Connect4, [10](#)
- control_num_pieces_players
 - Reversi, [20](#)
- find_all_directions_to_make_move
 - Reversi, [20](#)
- flip_pieces
 - Reversi, [21](#)
- Game, [10](#)
 - ~Game, [11](#)
 - check_win, [11](#)
 - Game, [10](#)
 - game_board, [12](#)
 - is_valid_move, [11](#)
 - make_move, [11](#)
 - switch_players, [11](#)
- game_board
 - Game, [12](#)
- get_current_player
 - Tic_tac_toe, [27](#)
- get_name
 - Player, [15](#)
- get_num_loss
 - Player, [15](#)
- get_num_pieces_player_O
 - Reversi, [21](#)
- get_num_pieces_player_X
 - Reversi, [21](#)
- get_num_win
 - Player, [15](#)
- get_space
 - Board, [8](#)
- get_username
 - Player, [16](#)
- include/Board.hpp, [29](#)
- include/Connect4.hpp, [29](#)
- include/Game.hpp, [30](#)
- include/Player.hpp, [30](#)
- include/Reversi.hpp, [30](#)
- include/Tic_tac_toe.hpp, [31](#)
- is_move_inside_board
 - Board, [8](#)
- is_space_free
 - Board, [9](#)
- is_space_free_reversi
 - Reversi, [22](#)
- is_there_direction_that_captures_opponent
 - Reversi, [22](#)
- is_there_player_piece_at_the_direction
 - Reversi, [22](#)
- is_there_valid_move_for_player
 - Reversi, [23](#)
- is_valid_move
 - Game, [11](#)
 - Reversi, [23](#)
 - Tic_tac_toe, [27](#)
- make_move
 - Game, [11](#)
 - Reversi, [24](#)
 - Tic_tac_toe, [28](#)
- Player, [12](#)
 - add_loss, [14](#)
 - add_win, [14](#)
 - compare_name, [14](#)

- compare_username, 15
- get_name, 15
- get_num_loss, 15
- get_num_win, 15
- get_username, 16
- Player, 13
- print_player, 16
- register_player, 16
- remove_player, 17
- set_name, 17
- set_num_loss, 17
- set_num_win, 17
- set_username, 18
- print_game_board
 - Board, 9
- print_player
 - Player, 16
- print_reversi_board
 - Reversi, 24
- print_tic_tac_toe_board
 - Tic_tac_toe, 28
- process_move
 - Reversi, 24
- register_player
 - Player, 16
- register_win_and_loss
 - Reversi, 24
- remove_player
 - Player, 17
- Reversi, 18
 - ~Reversi, 20
 - check_win, 20
 - control_num_pieces_players, 20
 - find_all_directions_to_make_move, 20
 - flip_pieces, 21
 - get_num_pieces_player_O, 21
 - get_num_pieces_player_X, 21
 - is_space_free_reversi, 22
 - is_there_direction_that_captures_opponent, 22
 - is_there_player_piece_at_the_direction, 22
 - is_there_valid_move_for_player, 23
 - is_valid_move, 23
 - make_move, 24
 - print_reversi_board, 24
 - process_move, 24
 - register_win_and_loss, 24
 - Reversi, 20
 - start_reversi_board, 25
- set_name
 - Player, 17
- set_num_loss
 - Player, 17
- set_num_win
 - Player, 17
- set_space
 - Board, 9
- set_username
 - Player, 18
- src/Board.cpp, 31
- src/Connect4.cpp, 32
- src/Game.cpp, 33
- src/main.cpp, 34
- src/Player.cpp, 36
- src/Reversi.cpp, 38
- src/Tic_tac_toe.cpp, 41
- start_reversi_board
 - Reversi, 25
- switch_players
 - Game, 11
- Tic_tac_toe, 25
 - ~Tic_tac_toe, 26
 - check_tic_tac_toe_win, 26
 - check_tie, 26
 - check_win, 27
 - get_current_player, 27
 - is_valid_move, 27
 - make_move, 28
 - print_tic_tac_toe_board, 28
 - Tic_tac_toe, 26