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 . . . 8

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_game_board() . . . 8

4.1.3.2 get_space() . . . 8

4.1.3.3 is_move_inside_board() . . . 9

4.1.3.4 is_space_free() . . . 9

4.1.3.5 print_game_board() . . . 10

4.1.3.6 set_game_board() . . . 10

4.1.3.7 set_space() . . . 10

4.1.4 Atributos . . . 10

4.1.4.1 game_board . . . 10

4.1.4.2 num_columns . . . 11

4.1.4.3 num_rows . . . 11

4.2 Referência da Classe Connect4 . . . 11

4.2.1 Descrição detalhada . . . 12

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 . . . 13

4.2.3.1 check_win() . . . 13

4.2.3.2 get_current_player() . . . 13

4.2.3.3 get_space() . . . 13

4.2.3.4 is_board_full() . . . 13

4.2.3.5 is_valid_move() [1/2] . . . 14

4.2.3.6 is_valid_move() [2/2] . . . 14

4.2.3.7 make_move() [1/2] . . . 14

4.2.3.8 make_move() [2/2] . . . 14

4.2.3.9 print_game_board() . . . 15

4.2.3.10 set_current_player() . . . 15

4.2.4 Atributos . . . 15

4.2.4.1 current_player . . . 15
4.3 Referência da Classe Game . . . 15
4.3.1 Descrição detalhada . . . 16
4.3.2 Construtores e Destrutores . . . 16
4.3.2.1 Game() . . . 16
4.3.2.2 ~Game() . . . 17
4.3.3 Documentação das funções . . . 17
4.3.3.1 check_win() . . . 17
4.3.3.2 is_valid_move() . . . 17
4.3.3.3 make_move() . . . 17
4.3.3.4 switch_players() . . . 17
4.3.4 Atributos . . . 18
4.3.4.1 game_board . . . 18
4.4 Referência da Classe Player . . . 18
4.4.1 Descrição detalhada . . . 19
4.4.2 Construtores e Destrutores . . . 19
4.4.2.1 Player() [1/3] . . . 19
4.4.2.2 Player() [2/3] . . . 20
4.4.2.3 Player() [3/3] . . . 21
4.4.3 Documentação das funções . . . 21
4.4.3.1 add_loss() . . . 21
4.4.3.2 add_win() . . . 21
4.4.3.3 compare_name() . . . 22
4.4.3.4 compare_username() . . . 22
4.4.3.5 find_player_in_list() . . . 22
4.4.3.6 get_name() . . . 23
4.4.3.7 get_num_loss() . . . 23
4.4.3.8 get_num_win() . . . 23
4.4.3.9 get_username() . . . 24
4.4.3.10 operator==() . . . 24
4.4.3.11 print_player() . . . 24
4.4.3.12 register_player() . . . 24
4.4.3.13 remove_player() . . . 25
4.4.3.14 set_name() . . . 25
4.4.3.15 set_num_loss() . . . 25
4.4.3.16 set_num_win() . . . 26
4.4.3.17 set_username() . . . 26
4.4.4 Atributos . . . 26
4.4.4.1 name . . . 26
4.4.4.2 num_loss . . . 26
4.4.4.3 num_win . . . 27
4.4.4.4 username . . . 27

4.5 Referência da Classe Reversi . . . 27
4.5.1 Descrição detalhada . . . 29
4.5.2 Construtores e Destrutores . . . 29
4.5.2.1 Reversi() . . . 29
4.5.2.2 ∼Reversi() . . . 29
4.5.3 Documentação das funções . . . 29
4.5.3.1 check_win() [1/2] . . . 29
4.5.3.2 check_win() [2/2] . . . 29
4.5.3.3 control_num_pieces_players() . . . 30
4.5.3.4 find_all_directions_to_make_move() . . . 30
4.5.3.5 flip_pieces() . . . 30
4.5.3.6 get_game_board() . . . 31
4.5.3.7 get_num_pieces_player_O() . . . 31
4.5.3.8 get_num_pieces_player_X() . . . 31
4.5.3.9 is_space_free_reversi() . . . 31
4.5.3.10 is_there_direction_that_captures_opponent() . . . 32
4.5.3.11 is_there_player_piece_at_the_direction() . . . 32
4.5.3.12 is_there_valid_move_for_player() . . . 33
4.5.3.13 is_valid_move() [1/2] . . . 33
4.5.3.14 is_valid_move() [2/2] . . . 33
4.5.3.15 make_move() [1/2] . . . 34
4.5.3.16 make_move() [2/2] . . . 34
4.5.3.17 process_move() . . . 34
4.5.3.18 register_win_and_loss() . . . 35
4.5.3.19 set_num_pieces_player_O() . . . 35
4.5.3.20 set_num_pieces_player_X() . . . 35
4.5.3.21 start_reversi_board() . . . 35
4.5.4 Atributos . . . 36
4.5.4.1 num_pieces_player_O . . . 36
4.5.4.2 num_pieces_player_X . . . 36
4.6 Referência da Classe Tic_tac_toe . . . 36
4.6.1 Descrição detalhada . . . 37
4.6.2 Construtores e Destrutores . . . 37
4.6.2.1 Tic_tac_toe() . . . 37
4.6.2.2 ∼Tic_tac_toe() . . . 37
4.6.3 Documentação das funções . . . 38
4.6.3.1 check_tic_tac_toe_win() . . . 38
4.6.3.2 check_tie() . . . 38
4.6.3.3 check_win() . . . 38
4.6.3.4 get_current_player() . . . 38
4.6.3.5 get_game_board() . . . 39
4.6.3.6 is_valid_move() [1/2] . . . 39

4.6.3.7 is_valid_move() [2/2] . . . 39
4.6.3.8 make_move() [1/2] . . . 39
4.6.3.9 make_move() [2/2] . . . 40
4.6.3.10 print_tic_tac_toe_board() . . . 40
4.6.4 Atributos . . . 40
4.6.4.1 current_player . . . 40
4.6.4.2 winner . . . 40

5 Arquivos **41**
5.1 Referência do Arquivo include/Board.hpp . . . 41
5.1.1 Descrição detalhada . . . 41
5.2 Board.hpp . . . 41
5.3 Referência do Arquivo include/Connect4.hpp . . . 42
5.3.1 Descrição detalhada . . . 42
5.4 Connect4.hpp . . . 42
5.5 Referência do Arquivo include/Game.hpp . . . 43
5.5.1 Descrição detalhada . . . 43
5.6 Game.hpp . . . 44
5.7 Referência do Arquivo include/Player.hpp . . . 44
5.7.1 Descrição detalhada . . . 44
5.7.2 Funções . . . 45
5.7.2.1 read_register_file() . . . 45
5.7.2.2 write_register_file() . . . 45
5.8 Player.hpp . . . 45
5.9 Referência do Arquivo include/Reversi.hpp . . . 46
5.9.1 Descrição detalhada . . . 46
5.9.2 Definições e macros . . . 47
5.9.2.1 REVERSI_H . . . 47
5.10 Reversi.hpp . . . 47
5.11 Referência do Arquivo include/Tic_tac_toe.hpp . . . 48
5.11.1 Descrição detalhada . . . 48
5.12 Tic_tac_toe.hpp . . . 48
5.13 Referência do Arquivo src/Board.cpp . . . 49
5.14 Board.cpp . . . 49
5.15 Referência do Arquivo src/Connect4.cpp . . . 50
5.16 Connect4.cpp . . . 50
5.17 Referência do Arquivo src/Game.cpp . . . 52
5.18 Game.cpp . . . 52
5.19 Referência do Arquivo src/main.cpp . . . 52
5.19.1 Funções . . . 53
5.19.1.1 main() . . . 53
5.20 main.cpp . . . 53

5.21 Referência do Arquivo src/Player.cpp . . . 58
5.21.1 Funções . . . 58
5.21.1.1 read_register_file() . . . 58
5.21.1.2 write_register_file() . . . 58
5.22 Player.cpp . . . 59
5.23 Referência do Arquivo src/Reversi.cpp . . . 61
5.23.1 Variáveis . . . 61
5.23.1.1 num_columns_and_rows_reversi . . . 61
5.24 Reversi.cpp . . . 61
5.25 Referência do Arquivo src/Tic_tac_toe.cpp . . . 65
5.25.1 Variáveis . . . 65
5.25.1.1 num_columns_received . . . 65
5.25.1.2 num_rows_received . . . 65
5.26 Tic_tac_toe.cpp . . . 65

Índice Remissivo **69**

Capítulo 1

Índice Hierárquico

1.1 Hierarquia de Classes

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

- Board . . . 7
- Game . . . 15
  - Connect4 . . . 11
  - Reversi . . . 27
  - Tic_tac_toe . . . 36
- Player . . . 18

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 11

Game
: Classe base para jogos com tabuleiro 15

Player 18

Reversi
: Gerencia as regras e funcionalidades do jogo Reversi 27

Tic_tac_toe
: Gerencia as regras e funcionalidades do Jogo da Velha 36

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 . . . 41

include/Connect4.hpp
: Implementa o jogo Connect4 (Lig4), baseado na classe genérica Game . . . 42

include/Game.hpp
: Classe base abstrata para jogos genéricos com tabuleiro . . . 43

include/Player.hpp
: Gerencia informações e ações relacionadas a jogadores . . . 44

include/Reversi.hpp
: Implementa o jogo Reversi, baseado na classe genérica Game . . . 46

include/Tic_tac_toe.hpp
: Implementa o Jogo da Velha (Tic Tac Toe), baseado na classe genérica Game . . . 48

src/Board.cpp . . . 49

src/Connect4.cpp . . . 50

src/Game.cpp . . . 52

src/main.cpp . . . 52

src/Player.cpp . . . 58

src/Reversi.cpp . . . 61

src/Tic_tac_toe.cpp . . . 65

tests/BoardClass_test.cpp . . . **??**

tests/Connect4Class_test.cpp . . . **??**

tests/PlayerClass_test.cpp . . . **??**

tests/ReversiClass_test.cpp . . . **??**

tests/TicTacToeClass_test.cpp . . . **??**

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.
- const std::unique_ptr< std::unique_ptr< char[]>[]> & get_game_board () const
  Retorna o tabuleiro completo.
- void set_game_board (char **board)
  Recebe um tabuleiro pronto e o copia para o tabuleiro vazio interno criado pelo construtor da classe Board.

Atributos Privados

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

4.1.1 Descrição detalhada

Gerencia o tabuleiro do jogo.

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

Definição na linha 20 do arquivo Board.hpp.

4.1.2 Construtores e Destrutores

4.1.2.1 Board()

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

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

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

Parâmetros

num_rows_received	Número de linhas.
num_columns_received	Número de colunas.

Definição na linha 18 do arquivo Board.cpp.

4.1.3 Documentação das funções

4.1.3.1 get_game_board()

```
const std::unique_ptr< std::unique_ptr< char[]>[]> & Board::get_game_board ( ) const
```

Retorna o tabuleiro completo.

Essa função foi criada somente para possibilitar os testes do programa. Assim, sua utilização se restringe aos arquivos do diretório /tests/.

Definição na linha 70 do arquivo Board.cpp.

4.1.3.2 get_space()

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

Retorna o valor de uma posição específica do tabuleiro.

Parâmetros

row	Linha do tabuleiro.
column	Coluna do tabuleiro.

Retorna

O valor presente na posição.

Definição na linha 10 do arquivo Board.cpp.

4.1.3.3 is_move_inside_board()

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

Verifica se uma posição está dentro dos limites do tabuleiro.

Parâmetros

x	Coordenada da linha.
y	Coordenada da coluna.

Retorna

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

Definição na linha 44 do arquivo Board.cpp.

4.1.3.4 is_space_free()

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

Verifica se uma posição no tabuleiro está vazia.

Parâmetros

x	Coordenada da linha.
y	Coordenada da coluna.

Retorna

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

Definição na linha 53 do arquivo Board.cpp.

4.1.3.5 print_game_board()

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

Imprime o estado atual do tabuleiro.

Definição na linha 32 do arquivo Board.cpp.

4.1.3.6 set_game_board()

```
void Board::set_game_board (
            char ** board )
```

Recebe um tabuleiro pronto e o copia para o tabuleiro vazio interno criado pelo construtor da classe Board.

Parâmetros

board	Tabuleiro a ser copiado

Essa função foi criada somente para possibilitar os testes do programa. Assim, sua utilização se restringe aos arquivos do diretório /tests/.

Definição na linha 61 do arquivo Board.cpp.

4.1.3.7 set_space()

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

Define um valor em uma posição específica do tabuleiro.

Parâmetros

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

Definição na linha 4 do arquivo Board.cpp.

4.1.4 Atributos

4.1.4.1 game_board

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

Estrutura que armazena o estado do tabuleiro.

Definição na linha 24 do arquivo Board.hpp.

4.1.4.2 num_columns

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

Número de colunas do tabuleiro.

Definição na linha 23 do arquivo Board.hpp.

4.1.4.3 num_rows

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

Número de linhas do tabuleiro.

Definição na linha 22 do arquivo Board.hpp.

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

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

4.2 Referência da Classe Connect4

Gerencia as regras e funcionalidades do jogo Connect4.

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

Diagrama de hierarquia da classe Connect4:

Diagrama de colaboração para Connect4:

Membros Públicos

- Connect4 ()
  Construtor padrão do Connect4, inicializa o jogo com 6 linhas, 7 colunas e o jogador 'X' como inicial.
- bool is_valid_move () const override
  Função declarada somente para fins de sobregarga.
- void make_move () override
  Função declarada somente para fins de sobregarga.
- bool check_win () override
  Verifica em todas direções válidas se o jogador atual venceu o jogo.
- bool is_valid_move (int column)
  Verifica se a jogada é válida para uma coluna específica.
- void make_move (int column)
  Realiza a jogada na coluna especificada.
- char get_current_player ()
  Retorna o jogador atual.
- bool is_board_full () const
  Verifica se o tabuleiro está completamente cheio.
- void print_game_board () const
  Imprime o estado atual do tabuleiro.
- void set_current_player (char player)
  Define o jogador atual.
- char get_space (int row, int column)
  Retorna o valor de uma casa no tabuleiro.
- ~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 128 do arquivo Connect4.cpp.

4.2.3 Documentação das funções

4.2.3.1 check_win()

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

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

Implementa Game.

Definição na linha 36 do arquivo Connect4.cpp.

4.2.3.2 get_current_player()

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

Retorna o jogador atual.

Retorna

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

Definição na linha 96 do arquivo Connect4.cpp.

4.2.3.3 get_space()

```
char Connect4::get_space (
            int row,
            int column )
```

Retorna o valor de uma casa no tabuleiro.

Parâmetros

row	Linha desejada
column	Coluna desejada

Retorna

O caractere que ocupa a casa

Definição na linha 123 do arquivo Connect4.cpp.

4.2.3.4 is_board_full()

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

Verifica se o tabuleiro está completamente cheio.

Retorna

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

Definição na linha 101 do arquivo Connect4.cpp.

4.2.3.5 is_valid_move() [1/2]

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

Função declarada somente para fins de sobregarga.

Implementa Game.

Definição na linha 8 do arquivo Connect4.cpp.

4.2.3.6 is_valid_move() [2/2]

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

Verifica se a jogada é válida para uma coluna específica.

Parâmetros

column	Coluna onde o jogador deseja jogar.

Retorna

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

Definição na linha 13 do arquivo Connect4.cpp.

4.2.3.7 make_move() [1/2]

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

Função declarada somente para fins de sobregarga.

Implementa Game.

Definição na linha 10 do arquivo Connect4.cpp.

4.2.3.8 make_move() [2/2]

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

Realiza a jogada na coluna especificada.

Parâmetros

column	Coluna onde o jogador deseja jogar.

Definição na linha 21 do arquivo Connect4.cpp.

4.2.3.9 print_game_board()

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

Imprime o estado atual do tabuleiro.

Definição na linha 113 do arquivo Connect4.cpp.

4.2.3.10 set_current_player()

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

Define o jogador atual.

Parâmetros

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

Definição na linha 118 do arquivo Connect4.cpp.

4.2.4 Atributos

4.2.4.1 current_player

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

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

Definição na linha 24 do arquivo Connect4.hpp.

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

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

4.3 Referência da Classe Game

Classe base para jogos com tabuleiro.

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

Diagrama de hierarquia da classe Game:

Diagrama de colaboração para Game:

Membros Públicos

- Game (int num_rows_received, int num_columns_received)

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

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

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

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

  Alterna entre os jogadores.
- ~Game ()

  Destrutor da classe base Game.

Atributos Protegidos

- Board game_board

4.3.1 Descrição detalhada

Classe base para jogos com tabuleiro.

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

Definição na linha 21 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 73 do arquivo Game.hpp.

4.3.3 Documentação das funções

4.3.3.1 check_win()

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

Verifica se há um vencedor no jogo.

Método abstrato que deve ser implementado pelas classes derivadas.

Retorna

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

Implementado por Connect4, Reversi e Tic_tac_toe.

4.3.3.2 is_valid_move()

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

Verifica se uma jogada é válida.

Método abstrato que deve ser implementado pelas classes derivadas.

Retorna

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

Implementado por Connect4, Reversi e Tic_tac_toe.

4.3.3.3 make_move()

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

Realiza uma jogada.

Método abstrato que deve ser implementado pelas classes derivadas.

Implementado por Connect4, Reversi e Tic_tac_toe.

4.3.3.4 switch_players()

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

Alterna entre os jogadores.

Parâmetros

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

Retorna

O caractere do próximo jogador.

Definição na linha 11 do arquivo Game.cpp.

4.3.4 Atributos

4.3.4.1 game_board

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

Representa o tabuleiro utilizado no jogo.

Definição na linha 23 do arquivo Game.hpp.

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

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

4.4 Referência da Classe Player

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

Membros Públicos

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

Retorna o nome de usuário do jogador.

- std::string get_name ()

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

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

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

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

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

  Incrementa o número de derrotas em um jogo.
- bool operator== (Player &player)

  sobrecarga do operador ==

Membros públicos estáticos

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

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

  Remove um jogador da lista de jogadores.
- static Player * find_player_in_list (std::list< Player > &player_list, const std::string &user_name)

  Procura se um jogador específico existe na lista.
- static bool compare_username (Player &player1, Player &player2)

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

  Compara os nomes de dois jogadores.

Atributos Privados

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

4.4.1 Descrição detalhada

Definição na linha 17 do arquivo Player.hpp.

4.4.2 Construtores e Destrutores

4.4.2.1 Player() [1/3]

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

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

Definição na linha 6 do arquivo Player.cpp.

4.4.2.2 Player() [2/3]

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

Construtor que inicializa o jogador com nome e username.

Parâmetros

name_received	Nome do jogador.
username_received	Nome de usuário.

Definição na linha 9 do arquivo Player.cpp.

4.4.2.3 Player() [3/3]

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

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

Parâmetros

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

Definição na linha 12 do arquivo Player.cpp.

4.4.3 Documentação das funções

4.4.3.1 add_loss()

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

Incrementa o número de derrotas em um jogo.

Parâmetros

key	Nome do jogo.

Definição na linha 61 do arquivo Player.cpp.

4.4.3.2 add_win()

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

Incrementa o número de vitórias em um jogo.

Parâmetros

key	Nome do jogo.

Definição na linha 55 do arquivo Player.cpp.

4.4.3.3 compare_name()

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

Compara os nomes de dois jogadores.

Parâmetros

player1	Primeiro jogador.
player2	Segundo jogador.

Retorna

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

Definição na linha 125 do arquivo Player.cpp.

4.4.3.4 compare_username()

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

Compara os usernames de dois jogadores.

Parâmetros

player1	Primeiro jogador.
player2	Segundo jogador.

Retorna

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

Definição na linha 115 do arquivo Player.cpp.

4.4.3.5 find_player_in_list()

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

Procura se um jogador específico existe na lista.

Parâmetros

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

Retorna

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

Definição na linha 98 do arquivo Player.cpp.

4.4.3.6 get_name()

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

Retorna o nome do jogador.

Retorna

Nome do jogador.

Definição na linha 43 do arquivo Player.cpp.

4.4.3.7 get_num_loss()

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

Retorna o mapa de derrotas por jogo.

Retorna

Mapa com o número de derrotas.

Definição na linha 51 do arquivo Player.cpp.

4.4.3.8 get_num_win()

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

Retorna o mapa de vitórias por jogo.

Retorna

Mapa com o número de vitórias.

Definição na linha 47 do arquivo Player.cpp.

4.4.3.9 get_username()

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

Retorna o nome de usuário do jogador.

Retorna

Nome de usuário.

Definição na linha 39 do arquivo Player.cpp.

4.4.3.10 operator==()

```
bool Player::operator== (
            Player & player )
```

sobrecarga do operador ==

Parâmetros

player	objeto player a ser comparado

Retorna

`true` se os objetos comparados tem os mesmos atributos, `false` caso contrario

Definição na linha 108 do arquivo Player.cpp.

4.4.3.11 print_player()

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

Imprime as informações do jogador no console.

Definição na linha 67 do arquivo Player.cpp.

4.4.3.12 register_player()

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

Registra um jogador em uma lista de jogadores.

Parâmetros

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

Retorna

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

Definição na linha 74 do arquivo Player.cpp.

4.4.3.13 remove_player()

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

Remove um jogador da lista de jogadores.

Parâmetros

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

Retorna

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

Definição na linha 86 do arquivo Player.cpp.

4.4.3.14 set_name()

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

Define o nome do jogador.

Parâmetros

name_received	Nome do jogador.

Definição na linha 15 do arquivo Player.cpp.

4.4.3.15 set_num_loss()

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

Atualiza o número de derrotas de um jogo.

Parâmetros

key	Nome do jogo.
value	Número de derrotas.

Definição na linha 31 do arquivo Player.cpp.

4.4.3.16 set_num_win()

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

Atualiza o número de vitórias de um jogo.

Parâmetros

key	Nome do jogo.
value	Número de vitórias.

Definição na linha 23 do arquivo Player.cpp.

4.4.3.17 set_username()

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

Define o nome de usuário do jogador.

Parâmetros

username_received	Nome de usuário.

Definição na linha 19 do arquivo Player.cpp.

4.4.4 Atributos

4.4.4.1 name

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

Nome do jogador.

Definição na linha 19 do arquivo Player.hpp.

4.4.4.2 num_loss

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

Número de derrotas por jogo.

Definição na linha 22 do arquivo Player.hpp.

4.4.4.3 num_win

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

Número de vitórias por jogo.

Definição na linha 21 do arquivo Player.hpp.

4.4.4.4 username

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

Nome de usuário do jogador.

Definição na linha 20 do arquivo Player.hpp.

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

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

4.5 Referência da Classe Reversi

Gerencia as regras e funcionalidades do jogo Reversi.

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

Diagrama de hierarquia da classe Reversi:

Diagrama de colaboração para Reversi:

Membros Públicos

- Reversi ()

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

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

  Retorna o número de peças do jogador O.
- void set_num_pieces_player_X (int x)

  Define manualmente o número de peças do jogador X.
- void set_num_pieces_player_O (int x)

  Define manualmente o número de peças do jogador O.
- Board & get_game_board ()

  Retorna o tabuleiro de Reversi completo.
- void start_reversi_board ()

  Inicializa o tabuleiro com as peças centrais do Reversi.
- 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_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_space_free_reversi (int x, int y) const

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

  Verifica se uma jogada específica é válida.
- bool is_there_valid_move_for_player (char player_piece)

  Verifica se há uma jogada válida para o jogador.
- bool check_win (bool is_there_move_for_player, char opponent_piece)

  Verifica se o jogo tem vencedor considerando as jogadas restantes.
- 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 (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.
- 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.
- bool process_move (std::array< int, 2 > move_coordinates, char player_piece_type)

  Processa uma jogada e verifica sua validade.
- void register_win_and_loss (Player *player1, Player *player2)

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

  Destrutor do jogo Reversi.
- bool is_valid_move () const override

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

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

  Função declarada somente para fins de sobregarga.

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 9 do arquivo Reversi.cpp.

4.5.2.2 ~Reversi()

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

Destrutor do jogo Reversi.

Definição na linha 261 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 270 do arquivo Reversi.cpp.

4.5.3.2 check_win() [2/2]

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

Verifica se o jogo tem vencedor considerando as jogadas restantes.

Parâmetros

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

Retorna

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

Definição na linha 145 do arquivo Reversi.cpp.

4.5.3.3 control_num_pieces_players()

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

Controla o número de peças dos jogadores após uma jogada.

Parâmetros

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

Definição na linha 171 do arquivo Reversi.cpp.

4.5.3.4 find_all_directions_to_make_move()

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

Encontra todas as direções válidas para capturar peças do oponente.

Parâmetros

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

Definição na linha 201 do arquivo Reversi.cpp.

4.5.3.5 flip_pieces()

```
void Reversi::flip_pieces (
            std::array< int, 2 > directions,
```

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

Captura peças do oponente em uma direção específica.

Parâmetros

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

Definição na linha 152 do arquivo Reversi.cpp.

4.5.3.6 get_game_board()

```
Board & Reversi::get_game_board ( )
```

Retorna o tabuleiro de Reversi completo.

Essa função foi declarada na classe Reversi unicamente para possibilitar os testes de outras funções da classe. Por isso, sua utilização se restringe ao diretório /tests/.

Definição na linha 32 do arquivo Reversi.cpp.

4.5.3.7 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 21 do arquivo Reversi.cpp.

4.5.3.8 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 16 do arquivo Reversi.cpp.

4.5.3.9 is_space_free_reversi()

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

Verifica se uma posição no tabuleiro está livre.

Parâmetros

x	Linha da posição.
y	Coluna da posição.

Retorna

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

Definição na linha 94 do arquivo Reversi.cpp.

4.5.3.10 is_there_direction_that_captures_opponent()

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

Verifica se há alguma direção que captura peças do oponente.

Parâmetros

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

Retorna

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

Definição na linha 64 do arquivo Reversi.cpp.

4.5.3.11 is_there_player_piece_at_the_direction()

```
bool Reversi::is_there_player_piece_at_the_direction (
            const char player_piece,
            const std::array< int, 2 > & direction,
            std::array< int, 2 > adjacent_square ) const
```

Verifica se há peças do jogador na direção especificada.

Parâmetros

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

Retorna

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

Definição na linha 44 do arquivo Reversi.cpp.

4.5.3.12 is_there_valid_move_for_player()

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

Verifica se há uma jogada válida para o jogador.

Parâmetros

player_piece	Tipo de peça do jogador.

Retorna

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

Definição na linha 118 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 266 do arquivo Reversi.cpp.

4.5.3.14 is_valid_move() [2/2]

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

Verifica se uma jogada específica é válida.

Parâmetros

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

Retorna

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

Definição na linha 101 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 263 do arquivo Reversi.cpp.

4.5.3.16 make_move() [2/2]

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

Realiza a jogada do jogador atual.

Parâmetros

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

Definição na linha 185 do arquivo Reversi.cpp.

4.5.3.17 process_move()

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

Processa uma jogada e verifica sua validade.

Parâmetros

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

Retorna

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

Definição na linha 231 do arquivo Reversi.cpp.

4.5.3.18 register_win_and_loss()

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

Registra vitória e derrota dos jogadores.

Parâmetros

player1	Jogador 1.
player2	Jogador 2.

Definição na linha 247 do arquivo Reversi.cpp.

4.5.3.19 set_num_pieces_player_O()

```
void Reversi::set_num_pieces_player_O (
            int x )
```

Define manualmente o número de peças do jogador O.

Essa função foi criada unicamente para possibilitar os testes de outras funções da classe Reversi. Por isso, sua utilização se restringe ao diretório /tests/.

Definição na linha 28 do arquivo Reversi.cpp.

4.5.3.20 set_num_pieces_player_X()

```
void Reversi::set_num_pieces_player_X (
            int x )
```

Define manualmente o número de peças do jogador X.

Essa função foi criada unicamente para possibilitar os testes de outras funções da classe Reversi. Por isso, sua utilização se restringe ao diretório /tests/.

Definição na linha 25 do arquivo Reversi.cpp.

4.5.3.21 start_reversi_board()

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

Inicializa o tabuleiro com as peças centrais do Reversi.

Definição na linha 36 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.
- Board & get_game_board ()
- 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 52 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 85 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 100 do arquivo Tic_tac_toe.cpp.

4.6.3.5 get_game_board()

```
Board & Tic_tac_toe::get_game_board ( )
```

Definição na linha 105 do arquivo Tic_tac_toe.cpp.

4.6.3.6 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.

4.6.3.7 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.

4.6.3.8 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.

4.6.3.9 make_move() [2/2]

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

Realiza uma jogada em uma posição específica.

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

Parâmetros

x	Coordenada da linha.
y	Coordenada da coluna.

Definição na linha 26 do arquivo Tic_tac_toe.cpp.

4.6.3.10 print_tic_tac_toe_board()

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

Imprime o estado atual do tabuleiro.

Definição na linha 20 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:
00027
00033         Board(int num_rows_received, int num_columns_received);
00034
00035
```

```
00042         void set_space(int row, int column, char value);
00043
00044
00051         char get_space(int row, int column) const;
00052
00053
00057         void print_game_board() const;
00058
00059
00066         bool is_move_inside_board(int x, int y) const;
00067
00068
00075         bool is_space_free(int x, int y) const;
00076
00077
00084         const std::unique_ptr<std::unique_ptr<char[]>[]>& get_game_board() const;
00085
00086
00095         void set_game_board(char **board);
00096
00097 };
00098
00099 #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
00100         char get_space(int row, int column);
00101
00102
00106         ~Connect4();
00107 };
00108
00109 #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
00006
00021 class Game {
00022     protected:
00023         Board game_board;
00025     public:
00031         Game(int num_rows_received, int num_columns_received);
00032
00033
00040         virtual bool is_valid_move() const = 0;
00041
00042
00048         virtual void make_move() = 0;
00049
00050
00057         virtual bool check_win() = 0;
00058
00059
00065         char switch_players(char current_player);
00066
00067
00073         ~Game() {}
00074 };
00075
00076 #endif
```

5.7 Referência do Arquivo include/Player.hpp

Gerencia informações e ações relacionadas a jogadores.

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

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

Componentes

- class Player

Funções

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

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

  Escreve os dados de registro de jogadores em um arquivo.

5.7.1 Descrição detalhada

Gerencia informações e ações relacionadas a jogadores.

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

Definição no arquivo Player.hpp.

5.7.2 Funções

5.7.2.1 read_register_file()

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

Lê os dados de registro de jogadores de um arquivo.

Parâmetros

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

Definição na linha 135 do arquivo Player.cpp.

5.7.2.2 write_register_file()

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

Escreve os dados de registro de jogadores em um arquivo.

Parâmetros

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

Definição na linha 157 do arquivo Player.cpp.

5.8 Player.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef PLAYER_H
00002 #define PLAYER_H
00003
00004 #include <iostream>
00005 #include <map>
00006 #include <list>
00007
00017 class Player {
00018     private:
00019         std::string name;
00020         std::string username;
00021         std::map<std::string, int> num_win;
00022         std::map<std::string, int> num_loss;
00024     public:
00028         Player();
00029
00035         Player(std::string name_received, std::string username_received);
00036
00044         Player(std::string name_received, std::string username_received, std::map<std::string, int>
      num_win_received, std::map<std::string, int> num_loss_received);
00045
```

```
00050         void set_name(std::string name_received);
00051
00056         void set_username(std::string username_received);
00057
00063         void set_num_win(std::string key, int value);
00064
00070         void set_num_loss(std::string key, int value);
00071
00076         std::string get_username();
00077
00082         std::string get_name();
00083
00088         std::map<std::string, int> get_num_win();
00089
00094         std::map<std::string, int> get_num_loss();
00095
00099         void print_player();
00100
00105         void add_win(std::string key);
00106
00111         void add_loss(std::string key);
00112
00119         static void register_player(Player player_received, std::list<Player> &player_list);
00120
00127         static void remove_player(std::string username_received, std::list<Player> &player_list);
00128
00136         static Player* find_player_in_list(std::list<Player>& player_list, const std::string&
    user_name);
00137
00143         bool operator == (Player &player);
00144
00151         static bool compare_username(Player &player1, Player &player2);
00152
00159         static bool compare_name(Player &player1, Player &player2);
00160 };
00161
00167 void read_register_file(std::list<Player> &player_list, std::ifstream &file_in);
00168
00174 void write_register_file(std::list<Player> &player_list, std::ofstream &file_out);
00175
00176 #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.

Definições e Macros

- #define REVERSI_H

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.

5.9.2 Definições e macros

5.9.2.1 REVERSI_H

```
#define REVERSI_H
```

Definição na linha 4 do arquivo Reversi.hpp.

5.10 Reversi.hpp

Ir para a documentação desse arquivo.

```
00001 #include "Game.hpp"
00002 #include "Player.hpp"
00003 #ifndef REVERSI_H
00004 #define REVERSI_H
00005 #include <array>
00006 #include <list>
00007
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
00056         void set_num_pieces_player_X(int x);
00057
00058
00065         void set_num_pieces_player_O(int x);
00066
00067
00074         Board& get_game_board();
00075
00076
00080         void start_reversi_board();
00081
00082
00090         bool is_there_player_piece_at_the_direction(const char player_piece, const std::array<int, 2>&
      direction,
00091             std::array<int, 2> adjacent_square) const;
00092
00093
00100         bool is_there_direction_that_captures_opponent(const std::array<int, 2> &move_coordinates,
      char player_piece_type);
00101
00102
00109         bool is_space_free_reversi(int x, int y) const;
00110
00111
00118         bool is_valid_move(std::array<int, 2>& move_coordinates, char player_piece_type);
00119
00120
00126         bool is_there_valid_move_for_player(char player_piece);
00127
00128
00135         bool check_win(bool is_there_move_for_player, char opponent_piece);
00136
00137
00144         void flip_pieces(std::array<int, 2> directions, std::array<int, 2> move_coordinates, char
      player_piece);
00145
00146
00152         void control_num_pieces_players(int num_pieces_flipped, char player_piece);
00153
00154
00161         void make_move(std::array<int, 2> move_coordinates, char player_piece,
      std::list<std::array<int, 2>& directions_to_capture_opponents);
00162
00163
```

```
00170         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);
00171
00172
00179         bool process_move(std::array<int, 2> move_coordinates, char player_piece_type);
00180
00181
00187         void register_win_and_loss(Player *player1, Player *player2);
00188
00189
00193         ~Reversi();
00194
00198         bool is_valid_move() const override;
00199
00203         bool check_win() override;
00204
00208         void make_move() override;
00209 };
00210
00211 #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         Board& get_game_board();
00039
```

```
00048         void make_move(int x, int y);
00049
00050
00054         bool is_valid_move() const override;
00055
00056
00066         bool is_valid_move(int& x, int& y) const;
00067
00068
00072         bool check_win() override;
00073
00074
00081         char check_tic_tac_toe_win() const;
00082
00083
00088         char get_current_player() const;
00089
00090
00097         bool check_tie() const;
00098
00099
00103         void print_tic_tac_toe_board() const;
00104
00105
00109         ~Tic_tac_toe();
00110 };
00111
00112 #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 }
00060
00061 void Board::set_game_board(char **board)
00062 {
00063     for (int i = 0; i < this->num_rows; i++)
00064     {
00065         for (int j = 0; j < this->num_rows; j++)
00066             this->game_board[i][j] = board[i][j];
00067     }
00068 }
00069
00070 const std::unique_ptr<std::unique_ptr<char[]>[]>& Board::get_game_board() const
00071 {
00072     return this->game_board;
00073 }
00074
```

5.15 Referência do Arquivo src/Connect4.cpp

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

Gráfico de dependência de inclusões para Connect4.cpp:

5.16 Connect4.cpp

Ir para a documentação desse arquivo.

```
00001 #include "Connect4.hpp"
00002 #include <iostream>
00003
00004
00005 Connect4::Connect4() : Game(6, 7), current_player('X') {}
00006
00007 // Funções declaradas somente para fins de sobrecarga
00008 bool Connect4::is_valid_move() const { return true; }
00009
00010 void Connect4::make_move() {}
00011 //
00012
00013 bool Connect4::is_valid_move(int column)
00014 {
00015     column--;
00016     if (game_board.get_space(0, column) != ' ') return false;
00017     if (column < 0 || column >= 7) return false;
00018     return true;
00019 }
00020
00021 void Connect4::make_move(int column)
00022 {
00023     column--;
00024     // Encontra a linha mais baixa disponível na coluna e coloca a peça do jogador atual
00025     for(int i = 5; i >= 0; i--)
00026     {
00027         if(game_board.get_space(i, column) == ' ')
00028         {
00029             game_board.set_space(i, column, current_player);
```

```
00030             return;
00031         }
00032     }
00033 }
00034
00035
00036 bool Connect4::check_win()
00037 {
00038     // Verificação de vitória horizontal
00039     for (int row = 0; row < 6; ++row)
00040     {
00041         for (int col = 0; col <= 3; ++col)
00042         {
00043             if (game_board.get_space(row, col) == current_player &&
00044                 game_board.get_space(row, col + 1) == current_player &&
00045                 game_board.get_space(row, col + 2) == current_player &&
00046                 game_board.get_space(row, col + 3) == current_player) {
00047                     return true;
00048             }
00049         }
00050     }
00051
00052     // Verificação de vitória vertical
00053     for (int row = 0; row <= 2; ++row)
00054     {
00055         for (int col = 0; col < 7; ++col)
00056         {
00057             if (game_board.get_space(row, col) == current_player &&
00058                 game_board.get_space(row + 1, col) == current_player &&
00059                 game_board.get_space(row + 2, col) == current_player &&
00060                 game_board.get_space(row + 3, col) == current_player) {
00061                     return true;
00062             }
00063         }
00064     }
00065
00066     // Verificação de vitória diagonal para a direita
00067     for (int row = 0; row <= 2; ++row)
00068     {
00069         for (int col = 0; col <= 3; ++col)
00070         {
00071             if (game_board.get_space(row, col) == current_player &&
00072                 game_board.get_space(row + 1, col + 1) == current_player &&
00073                 game_board.get_space(row + 2, col + 2) == current_player &&
00074                 game_board.get_space(row + 3, col + 3) == current_player) {
00075                     return true;
00076             }
00077         }
00078     }
00079
00080     // Verificação de vitória diagonal para a esquerda
00081     for (int row = 3; row < 6; ++row)
00082     {
00083         for (int col = 0; col <= 3; ++col)
00084         {
00085             if (game_board.get_space(row, col) == current_player &&
00086                 game_board.get_space(row - 1, col + 1) == current_player &&
00087                 game_board.get_space(row - 2, col + 2) == current_player &&
00088                 game_board.get_space(row - 3, col + 3) == current_player) {
00089                     return true;
00090             }
00091         }
00092     }
00093     return false;
00094 }
00095
00096 char Connect4::get_current_player()
00097 {
00098     return current_player;
00099 }
00100
00101 bool Connect4::is_board_full() const
00102 {
00103     for(int col = 0; col < 7; ++col)
00104     {
00105         if(game_board.get_space(0, col) == ' ')
00106         {
00107             return false;
00108         }
00109     }
00110     return true;
00111 }
00112
00113 void Connect4::print_game_board() const
00114 {
00115     game_board.print_game_board();
00116 }
```

```
00117
00118 void Connect4::set_current_player(char player)
00119 {
00120     current_player = player;
00121 }
00122
00123 char Connect4::get_space(int row, int column)
00124 {
00125     return game_board.get_space(row, column);
00126 }
00127
00128 Connect4::~Connect4() {}
```

5.17 Referência do Arquivo src/Game.cpp

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

Gráfico de dependência de inclusões para Game.cpp:

5.18 Game.cpp

Ir para a documentação desse arquivo.

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

5.19 Referência do Arquivo src/main.cpp

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

Gráfico de dependência de inclusões para main.cpp:

Funções

- int main ()

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

5.19.1 Funções

5.19.1.1 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 15 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
00015 int main()
00016 {
00017     std::ifstream file_in;
00018     try
00019     {
00020     file_in.open("teste");
00021     if (!file_in.is_open())
00022         throw std::runtime_error("Erro ao abrir o arquivo");
00023
00024     }
00025     catch (std::runtime_error &e)
00026     {
00027         std::cout « e.what() « std::endl;
00028     }
00029
00030     std::list<Player> player_list;
00031     read_register_file(player_list, file_in);
00032
00033     file_in.close();
00034
00035     std::string command;
00036
00037     // Loop principal que processa os comandos do usuário
00038     while(true)
00039     {
00040         try
00041         {
00042             std::cin » command;
00043             if (command == "LJ")
00044             {
00045                 // Listar jogadores ordenados por nome ou username
00046                 char sort_command;
00047                 try
00048                 {
00049                     std::cin » sort_command;
00050                     if (sort_command == 'A')
00051                         player_list.sort(Player::compare_username);
00052
00053                     else if (sort_command == 'N')
00054                         player_list.sort(Player::compare_name);
00055
00056                     else
00057                         throw std::invalid_argument("ERRO: comando inexistente");
00058
00059                     std::list<Player>::iterator it;
00060                     for (it = player_list.begin(); it != player_list.end(); it++)
00061                     {
00062                         it->print_player();
00063                         continue;
```

```
00064                     }
00065                 }
00066                 catch(std::invalid_argument &e)
00067                 {
00068                     std::cout « e.what() « std::endl;
00069                 }
00070             }
00071             else if (command == "CJ")
00072             {
00073                 // Cadastrar um novo jogador
00074                 std::string line_in, username_in, name_in;
00075                 try
00076                 {
00077                     std::getline(std::cin, line_in);
00078                     std::stringstream stream_in(line_in);
00079                     stream_in » username_in;
00080                     stream_in.ignore();
00081                     std::getline(stream_in, name_in);
00082                     if (name_in == "" || username_in == "")
00083                         throw std::invalid_argument("ERRO: dados incorretos, escreva o apelido e o
      nome do jogador");
00084 
00085                     Player new_player(name_in, username_in);
00086                     try
00087                     {
00088                         Player::register_player(new_player, player_list);
00089                         std::cout « "Jogador " « new_player.get_username() « " cadastrado com sucesso"
      « std::endl;
00090                     }
00091                     catch(std::invalid_argument &e)
00092                     {
00093                         std::cout « e.what() « std::endl;
00094                         continue;
00095                     }
00096                     continue;
00097                 }
00098                 catch(std::invalid_argument &e)
00099                 {
00100                     std::cout « e.what() « std::endl;
00101                 }
00102 
00103             }
00104             else if (command == "RJ")
00105             {
00106                 // Remover um jogador existente
00107                 std::string name_in, username_in;
00108                 try
00109                 {
00110                     std::cin » username_in;
00111                     if (username_in == "")
00112                         throw std::invalid_argument("ERRO: dados incorretos, escreva o apelido do
      jogador");
00113                     try
00114                     {
00115                         Player::remove_player(username_in, player_list);
00116                         std::cout « "Jogador " « username_in « " removido com sucesso" « std::endl;
00117                     }
00118                     catch(std::invalid_argument &e)
00119                     {
00120                         std::cout « e.what() « std::endl;
00121                         continue;
00122                     }
00123                 }
00124                 catch (std::invalid_argument &e)
00125                 {
00126                     std::cout « e.what() « std::endl;
00127                 }
00128             }
00129             else if(command == "EP")
00130             {
00131                 // Iniciar um jogo entre dois jogadores
00132                 char game;
00133                 std::string username_player1, username_player2, line_in;
00134 
00135                 try
00136                 {
00137                     std::getline(std::cin, line_in);
00138                     std::stringstream stream_in(line_in);
00139                     stream_in » game » username_player1 » username_player2;
00140 
00141                     if (game != 'R' && game != 'V' && game != 'L')
00142                         throw std::invalid_argument("ERRO: dados incorretos, selecione um dos jogos
      disponíveis");
00143 
00144                     else if (username_player1 == "" || username_player2 == "")
00145                         throw std::invalid_argument("ERRO: dados incorretos, escreva o apelido dos
      dois jogadores");
```

```
00146
00147                     Player *player1 = Player::find_player_in_list(player_list, username_player1);
00148                     Player *player2 = Player::find_player_in_list(player_list, username_player2);
00149
00150                     if (player1 == nullptr)
00151                         throw std::invalid_argument("ERRO: jogador " + username_player1 + "
      inexistente");
00152
00153                     else if (player2 == nullptr)
00154                         throw std::invalid_argument("ERRO: jogador " + username_player2 + "
      inexistente");
00155
00156                     // Inicialização do jogo com base no tipo selecionado
00157                     if (game == 'R')
00158                     {
00159                         Reversi reversi_game;
00160                         char player_piece = 'X';
00161                         char opponent_piece = 'O';
00162
00163                         while (true)
00164                         {
00165                             int x, y;
00166                             bool is_there_movement_for_player =
      reversi_game.is_there_valid_move_for_player(player_piece);
00167                             bool someone_won = reversi_game.check_win(is_there_movement_for_player,
      player_piece);
00168                             reversi_game.get_game_board().print_game_board();
00169                             std::cout << "X: " << reversi_game.get_num_pieces_player_X() << " " << "O: "
00170                                     << reversi_game.get_num_pieces_player_O() << std::endl;
00171
00172                             if (someone_won)
00173                             {
00174                                 if (reversi_game.get_num_pieces_player_X() >
      reversi_game.get_num_pieces_player_O())
00175                                 std::cout << username_player1 << " ganhou!" << std::endl;
00176
00177                                 else if (reversi_game.get_num_pieces_player_X() <
      reversi_game.get_num_pieces_player_O())
00178                                 std::cout << username_player2 << " ganhou!" << std::endl;
00179
00180                                 else
00181                                 std::cout << "Houve empate!" << std::endl;
00182
00183                                 break;
00184                             }
00185                             else if (is_there_movement_for_player && !someone_won)
00186                             {
00187
00188                                 if (player_piece == 'X')
00189                                     std::cout << username_player1 << " " << "[X]" << ": " << std::ends;
00190                                 else
00191                                     std::cout << username_player2 << " " << "[O]" << ": " << std::ends;
00192
00193                                 try {
00194
00195                                     if (!(std::cin >> x )) {
00196                                         std::cin.clear();
00197                                         std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
      '\n');
00198                                         throw std::invalid_argument("Entrada inválida. Por favor
      forneça dois números inteiros.");
00199                                     }
00200
00201                                     if(!(std::cin >> y)){
00202                                         std::cin.clear();
00203                                         std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
      '\n');
00204                                         throw std::invalid_argument("Entrada inválida, vez passada
      para o oponente");
00205                                     }
00206
00207                                     if (!reversi_game.process_move({x, y}, player_piece)) {
00208                                         throw std::invalid_argument("Jogada inválida, vez passada para
      o oponente.");
00209                                     }
00210
00211                                 }
00212                                 catch (const std::invalid_argument &e) {
00213
00214                                     std::cout << "Error: " << e.what() << std::endl;
00215                                 }
00216
00217                                 player_piece = reversi_game.switch_players(player_piece);
00218                                 opponent_piece = reversi_game.switch_players(opponent_piece);
00219
00220                             }
00221                             else if (!is_there_movement_for_player && !someone_won)
```

```
00222                             {
00223                                 player_piece = reversi_game.switch_players(player_piece);
00224                                 opponent_piece = reversi_game.switch_players(opponent_piece);
00225                                 std::cout « "Não há jogadas válidas, vez passada para o oponente" «
    std::endl;
00226                             }
00227                         }
00228                     }
00229                     else if (game == 'L')
00230                     {
00231                         Connect4 connect4_game;
00232                         bool game_over = false;
00233 
00234                         while (!game_over)
00235                         {
00236                             int column;
00237                             connect4_game.print_game_board();
00238                             char current_player = connect4_game.get_current_player();
00239 
00240                             std::cout « "Turno de jogador <" « current_player « ">:" « std::endl;
00241 
00242                             try {
00243                                 if (!(std::cin » column))
00244                                 {
00245                                     std::cin.clear();
00246                                     std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
    '\n');
00247                                     throw std::invalid_argument("Entrada inválida, insira um número
    inteiro.");
00248                                 }
00249 
00250                                 if (!connect4_game.is_valid_move(column))
00251                                 {
00252                                     throw std::out_of_range("Entrada inválida, insira um número entre
    1 e 7");
00253                                 }
00254 
00255                                 if (std::cin.peek() != '\n')
00256                                 {
00257                                     std::cin.clear();
00258                                     std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
    '\n');
00259                                     throw std::invalid_argument("Entrada inválida, inspira apenas um
    número que é referente a coluna");
00260                                 }
00261 
00262                                 connect4_game.make_move(column);
00263 
00264                                 if (connect4_game.check_win())
00265                                 {
00266                                     connect4_game.print_game_board();
00267                                     if (current_player == 'X')
00268                                     {
00269                                         player1->add_win("Lig4");
00270                                         player2->add_loss("Lig4");
00271                                         std::cout « username_player1 « " ganhou!" « std::endl;
00272                                     }
00273                                     else
00274                                     {
00275                                         player2->add_win("Lig4");
00276                                         player1->add_loss("Lig4");
00277                                         std::cout « username_player2 « " ganhou!" « std::endl;
00278                                     }
00279                                     game_over = true;
00280                                 }
00281                                 else if (connect4_game.is_board_full())
00282                                 {
00283                                     std::cout « "Houve empate!" « std::endl;
00284                                     game_over = true;
00285                                 }
00286                             }
00287                             catch (const std::out_of_range& e)
00288                             {
00289                                 std::cout « "Erro: " « e.what() « std::endl;
00290                             }
00291                             catch (const std::runtime_error& e)
00292                             {
00293                                 std::cout « "Erro: " « e.what() « std::endl;
00294                             }
00295                             catch (const std::invalid_argument& e)
00296                             {
00297                                 std::cout « "Erro: " « e.what() « std::endl;
00298                             }
00299                             connect4_game.set_current_player((current_player == 'X') ? 'O' : 'X');
00300                         }
00301                     }
00302                     else if (game == 'V')
```

```
00303                     {
00304                         Tic_tac_toe tic_tac_toe_game;
00305                         int x, y;
00306 
00307                         std::cout << username_player1 << " eh X e " << username_player2 << " eh O" <<
      std::endl;
00308 
00309                         while (true)
00310                         {
00311                             if (tic_tac_toe_game.check_tic_tac_toe_win() != 'F')
00312                             {
00313                                 if (tic_tac_toe_game.get_current_player() == 'X')
00314                                 {
00315                                 tic_tac_toe_game.print_tic_tac_toe_board();
00316                                 std::cout << username_player1 << " ganhou!" << std::endl;
00317                                 player1->add_win("Velha");
00318                                 player2->add_loss("Velha");
00319                                 break;
00320                                 }
00321 
00322                                 tic_tac_toe_game.print_tic_tac_toe_board();
00323                                 std::cout << username_player2 << " ganhou!" << std::endl;
00324                                 player2->add_win("Velha");
00325                                 player1->add_loss("Velha");
00326                                 break;
00327 
00328                             }
00329 
00330                             if (tic_tac_toe_game.check_tie())
00331                             {
00332                                 std::cout << "Houve empate!" << std::endl;
00333                                 break;
00334                             }
00335 
00336                             std::cout << "Turno de jogador " << tic_tac_toe_game.get_current_player() <<
      std::endl;
00337                             tic_tac_toe_game.print_tic_tac_toe_board();
00338 
00339                             try
00340                             {
00341                                 if (!(std::cin >> x >> y))
00342                                     throw std::invalid_argument("Entrada inválida. Por favor forneça
      dois números inteiros.");
00343                             }
00344 
00345                             catch (const std::invalid_argument &e)
00346                             {
00347                                 std::cerr << "Erro: " << e.what() << std::endl;
00348                                 std::cin.clear();
00349                                 std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00350                                 continue;
00351                             }
00352 
00353                             tic_tac_toe_game.make_move(x, y);
00354                         }
00355                     }
00356 
00357                 }
00358                 catch (std::invalid_argument &e)
00359                 {
00360                     std::cout << e.what() << std::endl;
00361                 }
00362 
00363             } else if (command == "FS"){
00364                 break;
00365 
00366             } else {
00367                 throw std::invalid_argument("ERRO: comando inexiste");
00368             }
00369 
00370         } catch(std::invalid_argument &e) {
00371             std::cout << e.what() << std::endl;
00372         }
00373     }
00374 
00375     // Escrita do arquivo de registro atualizado
00376     std::ofstream file_out;
00377     try {
00378     file_out.open("teste");
00379     if (!file_out.is_open())
00380         throw std::runtime_error("Erro ao abrir o arquivo");
00381 
00382     } catch (std::runtime_error &e){
00383         std::cout << e.what() << std::endl;
00384     }
00385     write_register_file(player_list, file_out);
00386 
```

```
00387     file_out.close();
00388
00389     return 0;
00390 }
```

5.21 Referência do Arquivo src/Player.cpp

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

Gráfico de dependência de inclusões para Player.cpp:

Funções

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

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

  Escreve os dados de registro de jogadores em um arquivo.

5.21.1 Funções

5.21.1.1 read_register_file()

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

Lê os dados de registro de jogadores de um arquivo.

Parâmetros

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

Definição na linha 135 do arquivo Player.cpp.

5.21.1.2 write_register_file()

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

Escreve os dados de registro de jogadores em um arquivo.

Parâmetros

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

Definição na linha 157 do arquivo Player.cpp.

5.22 Player.cpp

Ir para a documentação desse arquivo.

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

```
00071     std::cout « "VELHA" « "\t" « "- V: " « this->num_win.find("Velha")->second « " D: " « this->num_loss.find("Velha")->second « std::endl;
00072 }
00073
00074 void Player::register_player(Player player_received, std::list<Player> &player_list){
00075     std::list<Player>::iterator it;
00076     for (it = player_list.begin(); it != player_list.end(); it++){
00077         if (it->get_username() == player_received.get_username()){
00078             throw std::invalid_argument("ERRO: Jogador repetido");
00079             return;
00080         }
00081     }
00082     player_list.push_back(player_received);
00083     return;
00084 }
00085
00086 void Player::remove_player(std::string username_received, std::list<Player> &player_list){
00087     std::list<Player>::iterator it;
00088     for (it = player_list.begin(); it != player_list.end(); it++){
00089         if (it->get_username() == username_received){
00090             it = player_list.erase(it);
00091             return;
00092         }
00093     }
00094     throw std::invalid_argument("ERRO: Jogador inexistente");
00095     return;
00096 }
00097
00098 Player* Player::find_player_in_list(std::list<Player>& player_list, const std::string& username) {
00099     for (auto& player : player_list) {
00100         if (player.get_username() == username) {
00101             return &player;
00102         }
00103     }
00104     throw std::invalid_argument("ERRO: Jogador inexistente");
00105     return nullptr;
00106 }
00107
00108 bool Player::operator == (Player &player){
00109     if (this->username == player.get_username() && this->name == player. get_name() && this->num_loss == player.get_num_loss() && this->num_win == player.get_num_win())
00110         return true;
00111     else
00112         return false;
00113 }
00114
00115 bool Player::compare_username(Player &player1, Player &player2){
00116     for (unsigned int i = 0; (i < player1.get_username().size()) && (i < player2.get_username().size()); i++){
00117         if (tolower(player1.get_username()[i]) < tolower(player2.get_username()[i]))
00118             return true;
00119         else if (tolower(player1.get_username()[i]) > tolower(player2.get_username()[i]))
00120             return false;
00121     }
00122     return player1.get_username().size() < player2.get_username().size();
00123 }
00124
00125 bool Player::compare_name(Player &player1, Player &player2){
00126     for (unsigned int i = 0; (i < player1.get_name().size()) && (i < player2.get_name().size()); i++){
00127         if (tolower(player1.get_name()[i]) < tolower(player2.get_name()[i]))
00128             return true;
00129         else if (tolower(player1.get_name()[i]) > tolower(player2.get_name()[i]))
00130             return false;
00131     }
00132     return player1.get_name().size() < player2.get_name().size();
00133 }
00134
00135 void read_register_file(std::list<Player> &player_list, std::ifstream &file_in) {
00136     std::string file_line[5];
00137     Player player_in;
00138     int i = 0;
00139     while (getline(file_in, file_line[i])) {
00140         i++;
00141         if (i == 5) {
00142         player_in.set_username(file_line[0]);
00143         player_in.set_name(file_line[1]);
00144         std::string key_in, num_win_in, num_loss_in;
00145         for (int j = 2; j < 5; j++){
00146             std::stringstream file_stream(file_line[j]);
00147             file_stream » key_in » num_win_in » num_loss_in;
00148             player_in.set_num_win(key_in, stoi(num_win_in));
00149             player_in.set_num_loss(key_in, stoi(num_loss_in));
00150         }
00151         player_list.push_back(player_in);
00152         i = 0;
00153         }
00154     }
```

```
00155 }
00156
00157 void write_register_file(std::list<Player> &player_list, std::ofstream &file_out){
00158     std::list<Player>::iterator it;
00159     for (it = player_list.begin(); it != player_list.end(); it++){
00160         file_out « it->get_username() « std::endl;
00161         file_out « it->get_name() « std::endl;
00162         file_out « "Reversi" « " " « it->get_num_win().find("Reversi")->second « " " «
      it->get_num_loss().find("Reversi")->second « std::endl;
00163         file_out « "Lig4" « " " « it->get_num_win().find("Lig4")->second « " " «
      it->get_num_loss().find("Lig4")->second « std::endl;
00164         file_out « "Velha" « " " « it->get_num_win().find("Velha")->second « " " «
      it->get_num_loss().find("Velha")->second « std::endl;
00165     }
00166 }
```

5.23 Referência do Arquivo src/Reversi.cpp

```
#include "Reversi.hpp"
#include "Board.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 7 do arquivo Reversi.cpp.

5.24 Reversi.cpp

Ir para a documentação desse arquivo.

```
00001 #include "Reversi.hpp"
00002 #include "Board.hpp"
00003 #include <iostream>
00004 #include "list"
00005 #include "array"
00006 #include "Player.hpp"
00007 const int num_columns_and_rows_reversi = 8;
00008
00009 Reversi::Reversi() : Game(num_columns_and_rows_reversi, num_columns_and_rows_reversi)
00010 {
00011     this->start_reversi_board();
00012     this->num_pieces_player_X = 2;
00013     this->num_pieces_player_O = 2;
00014 }
00015
00016 int Reversi::get_num_pieces_player_X()
00017 {
00018     return this->num_pieces_player_X;
00019 }
00020
```

```
00021 int Reversi::get_num_pieces_player_O()
00022 {
00023     return this->num_pieces_player_O;
00024 }
00025 void Reversi::set_num_pieces_player_X(int x){
00026     this->num_pieces_player_X = x;
00027 }
00028 void Reversi::set_num_pieces_player_O(int x){
00029     this->num_pieces_player_O = x;
00030 }
00031
00032 Board& Reversi::get_game_board(){
00033     return this->game_board;
00034 }
00035
00036 void Reversi::start_reversi_board()
00037 {
00038     this->game_board.set_space(3, 3, 'X');
00039     this->game_board.set_space(4, 4, 'X');
00040     this->game_board.set_space(3, 4, 'O');
00041     this->game_board.set_space(4, 3, 'O');
00042 }
00043
00044 bool Reversi::is_there_player_piece_at_the_direction(const char player_piece,
00045     const std::array<int, 2>& direction, std::array<int, 2> adjacent_square) const
00046 {
00047     std::array<int, 2> current_square = { adjacent_square[0] + direction[0],
00048     adjacent_square[1] + direction[1] };
00049
00050     while (game_board.is_move_inside_board(current_square[0], current_square[1]))
00051     {
00052         if (this->game_board.get_space(current_square[0], current_square[1]) == player_piece)
00053             return true;
00054         else if (this->game_board.get_space(current_square[0], current_square[1]) == ' '
00055         || this->game_board.get_space(current_square[0], current_square[1]) == '*')
00056             return false;
00057
00058         current_square[0] += direction[0];
00059         current_square[1] += direction[1];
00060     }
00061     return false;
00062 }
00063
00064 bool Reversi::is_there_direction_that_captures_opponent(const std::array<int, 2>
00065     & move_coordinates, char player_piece_type)
00066 {
00067     char opponent_player = switch_players(player_piece_type);
00068
00069     std::array<int, 2> adjacent_square = { 0, 0 };
00070
00071     for (int i = 1; i > -2; i--)
00072     {
00073         for (int j = 1; j > -2; j--)
00074         {
00075             if (j != 0 || i != 0)
00076             {
00077                 adjacent_square[0] = move_coordinates[0] + i;
00078                 adjacent_square[1] = move_coordinates[1] + j;
00079
00080                 if (this->game_board.is_move_inside_board(adjacent_square[0], adjacent_square[1]) &&
00081                     this->game_board.get_space(adjacent_square[0], adjacent_square[1]) ==
      opponent_player)
00082                     {
00083                     std::array<int, 2> direction = { i , j };
00084
00085                     if (is_there_player_piece_at_the_direction(player_piece_type, direction,
      adjacent_square))
00086                         return true;
00087                     }
00088             }
00089         }
00090     }
00091     return false;
00092 }
00093
00094 bool Reversi::is_space_free_reversi(int x, int y) const
00095 {
00096     if (this->game_board.is_space_free(x, y) || this->game_board.get_space(x, y) == '*')
00097         return true;
00098     return false;
00099 }
00100
00101 bool Reversi::is_valid_move(std::array<int, 2>& move_coordinates, char player_piece_type)
00102 {
00103
00104     if (!this->game_board.is_move_inside_board(move_coordinates[0], move_coordinates[1]))
00105         return false;
```

```
00106
00107     if (!this->is_space_free_reversi(move_coordinates[0], move_coordinates[1]))
00108         return false;
00109
00110     if (!this->is_there_direction_that_captures_opponent(move_coordinates, player_piece_type))
00111         return false;
00112
00113     return true;
00114
00115 }
00116
00117
00118 bool Reversi::is_there_valid_move_for_player(char player_piece)
00119 {
00120     bool found_valid_move = false;
00121     for (int i = 0; i < num_columns_and_rows_reversi; i++)
00122     {
00123         for (int j = 0; j < num_columns_and_rows_reversi; j++)
00124         {
00125             if (this->game_board.get_space(i, j) == ' ' || this->game_board.get_space(i, j) == '*')
00126             {
00127                 std::array<int, 2> coordinates = { i, j };
00128                 if (this->is_valid_move(coordinates, player_piece))
00129                 {
00130                     this->game_board.set_space(i, j,'*');
00131                     found_valid_move = true;
00132                 }
00133                 else
00134                 {
00135                     if (this->game_board.get_space(i, j) == '*')
00136                         this->game_board.set_space(i, j, ' ');
00137                 }
00138
00139             }
00140         }
00141     }
00142     return found_valid_move;
00143 }
00144
00145 bool Reversi::check_win(bool is_there_move_for_player, char opponent_piece)
00146 {
00147     if (!is_there_move_for_player && !this->is_there_valid_move_for_player(opponent_piece))
00148         return true;
00149     return false;
00150 }
00151
00152 void Reversi::flip_pieces(std::array<int, 2> directions, std::array<int, 2> move_coordinates, char
    player_piece)
00153 {
00154     char opponent_piece = switch_players(player_piece);
00155     int num_pieces_flipped = 0;
00156
00157     std::array<int, 2> current_square = { directions[0] + move_coordinates[0],
00158     directions[1] + move_coordinates[1] };
00159
00160     while (this->game_board.get_space(current_square[0], current_square[1]) == opponent_piece)
00161     {
00162         this->game_board.set_space(current_square[0], current_square[1], player_piece);
00163         current_square[0] += directions[0];
00164         current_square[1] += directions[1];
00165         num_pieces_flipped++;
00166     }
00167     this->control_num_pieces_players(num_pieces_flipped, player_piece);
00168 }
00169
00170
00171 void Reversi::control_num_pieces_players(int num_pieces_flipped, char player_piece)
00172 {
00173     if (player_piece == 'X')
00174     {
00175         this->num_pieces_player_X += num_pieces_flipped;
00176         this->num_pieces_player_O -= num_pieces_flipped;
00177     }
00178     else
00179     {
00180         this->num_pieces_player_X -= num_pieces_flipped;
00181         this->num_pieces_player_O += num_pieces_flipped;
00182     }
00183 }
00184
00185 void Reversi::make_move(std::array<int, 2> move_coordinates, char player_piece,
00186 std::list<std::array<int, 2>>& directions_to_capture_opponents)
00187 {
00188
00189     this->game_board.set_space(move_coordinates[0], move_coordinates[1], player_piece);
00190
00191     if (player_piece == 'X')
```

```
00192         this->num_pieces_player_X++;
00193     else
00194         this->num_pieces_player_O++;
00195 
00196 
00197     for (auto direction : directions_to_capture_opponents)
00198         this->flip_pieces(direction, move_coordinates, player_piece);
00199 }
00200 
00201 void Reversi::find_all_directions_to_make_move(std::array<int, 2>& move_coordinates,
00202     char player_piece, std::list<std::array<int, 2>& directions_to_capture_opponents)
00203 {
00204     char opponent_player_piece = switch_players(player_piece);
00205 
00206     std::array<int, 2> adjacent_square = { 0, 0 };
00207 
00208     for (int i = 1; i > -2; i--)
00209     {
00210         for (int j = 1; j > -2; j--)
00211         {
00212             if (j != 0 || i != 0)
00213             {
00214                 adjacent_square[0] = move_coordinates[0] + i;
00215                 adjacent_square[1] = move_coordinates[1] + j;
00216 
00217                 if (this->game_board.is_move_inside_board(adjacent_square[0], adjacent_square[1]) &&
00218                     this->game_board.get_space(adjacent_square[0], adjacent_square[1]) ==
    opponent_player_piece)
00219                 {
00220                     std::array<int, 2> direction = { i , j };
00221 
00222                     if (is_there_player_piece_at_the_direction(player_piece, direction,
    adjacent_square))
00223                         directions_to_capture_opponents.push_back(direction);
00224 
00225                 }
00226             }
00227         }
00228     }
00229 }
00230 
00231 bool Reversi::process_move(std::array<int, 2> move_coordinates, char player_piece)
00232 {
00233     move_coordinates[0] = move_coordinates[0] - 1;
00234     move_coordinates[1] = move_coordinates[1] - 1;
00235     std::list<std::array<int, 2» directions_to_capture_opponents;
00236 
00237     if (this->is_valid_move(move_coordinates, player_piece))
00238     {
00239         find_all_directions_to_make_move(move_coordinates, player_piece,
    directions_to_capture_opponents);
00240         this->make_move(move_coordinates, player_piece, directions_to_capture_opponents);
00241         return true;
00242     }
00243     return false;
00244 }
00245 
00246 
00247 void Reversi::register_win_and_loss(Player *player1, Player *player2)
00248 {
00249     if(this->num_pieces_player_X > this->num_pieces_player_O)
00250     {
00251         player1->add_win("Reversi");
00252         player2->add_loss("Reversi");
00253     }
00254     else if(this->num_pieces_player_X < this->num_pieces_player_O)
00255     {
00256         player2->add_win("Reversi");
00257         player1->add_loss("Reversi");
00258     }
00259 }
00260 
00261 Reversi::~Reversi() {};
00262 
00263 void Reversi::make_move() {
00264     return;
00265 }
00266 bool Reversi::is_valid_move() const {
00267     return false;
00268 }
00269 
00270 bool Reversi::check_win() {
00271     return false;
00272 }
```

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     if (!game_board.is_move_inside_board(x, y) || !game_board.is_space_free(x, y))
00014         throw std::runtime_error("Coordenada invalida, vez passada para o oponente");
00015
00016     return true;
00017 }
00018
00019
00020 void Tic_tac_toe::print_tic_tac_toe_board() const
00021 {
00022     game_board.print_game_board();
00023 }
00024
00025
00026 void Tic_tac_toe::make_move(int x, int y)
00027 {
00028     x -= 1;
00029     y -= 1;
00030
```

```
00031     try
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::runtime_error& e)
00045     {
00046         std::cout << "Erro: " << e.what() << std::endl;
00047         current_player = switch_players(current_player);
00048     }
00049 
00050 }
00051 
00052 char Tic_tac_toe::check_tic_tac_toe_win() const
00053 {
00054     // Verifica se há vitória nas linhas ou colunas
00055     for (int i = 0; i < 3; ++i) {
00056         if (game_board.get_space(i, 0) == current_player &&
00057             game_board.get_space(i, 1) == current_player &&
00058             game_board.get_space(i, 2) == current_player)
00059             return current_player;
00060 
00061 
00062         if (game_board.get_space(0, i) == current_player &&
00063             game_board.get_space(1, i) == current_player &&
00064             game_board.get_space(2, i) == current_player)
00065             return current_player;
00066 
00067     }
00068 
00069     // Verifica se há vitória nas diagonais
00070     if (game_board.get_space(0, 0) == current_player &&
00071         game_board.get_space(1, 1) == current_player &&
00072         game_board.get_space(2, 2) == current_player)
00073         return current_player;
00074 
00075 
00076     if (game_board.get_space(0, 2) == current_player &&
00077         game_board.get_space(1, 1) == current_player &&
00078         game_board.get_space(2, 0) == current_player)
00079         return current_player;
00080 
00081     return 'F';
00082 }
00083 
00084 
00085 bool Tic_tac_toe::check_tie() const
00086 {
00087     for (int i = 0; i < 3; i++)
00088     {
00089         for (int j = 0; j < 3; j++)
00090         {
00091             if (game_board.get_space(i, j) == ' ')
00092                 return false;
00093         }
00094     }
00095 
00096 return true;
00097 }
00098 
00099 
00100 char Tic_tac_toe::get_current_player() const
00101 {
00102     return current_player;
00103 }
00104 
00105 Board& Tic_tac_toe::get_game_board(){
00106     return this->game_board;
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
```

5.27 Referência do Arquivo tests/BoardClass_test.cpp

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

Gráfico de dependência de inclusões para BoardClass_test.cpp:

5.28 BoardClass_test.cpp

Ir para a documentação desse arquivo.

```
00001 #define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
00002 #include "doctest.h"
00003 #include "Board.hpp"
00004 #include <array>
00005 const int num_columns_and_rows_reversi = 8;
00006 const int num_columns_and_rows_tic_tac_toe = 3;
00007 const int num_columns_lig4 = 7;
00008 const int num_rows_lig4 = 6;
00009
00010 TEST_CASE("Function set_game_board test")
00011 {
00012     SUBCASE("Reversi board")
00013     {
00014
00015         // Cria o tabuleiro como ponteiro duplo de char
00016         char** board_sample = new char*[num_columns_and_rows_reversi];
00017         for (int i = 0; i < num_columns_and_rows_reversi; ++i)
00018             board_sample[i] = new char[num_columns_and_rows_reversi];
00019
00020         // Inicializa os valores do tabuleiro
00021         char board_initial_values[num_columns_and_rows_reversi][num_columns_and_rows_reversi] =
00022         {
00023             {' ', ' ', ' ', '*', ' ', ' ', ' ', ' ', ' '},
00024             {' ', ' ', 'O', '*', ' ', ' ', ' ', ' ', ' '},
00025             {' ', ' ', 'X', 'O', 'X', ' ', ' ', ' ', ' '},
00026             {' ', ' ', '*', 'X', 'O', '*', '*', ' ', ' '},
00027             {' ', ' ', '*', 'O', 'X', 'O', '*', ' ', ' '},
00028             {' ', ' ', ' ', '*', 'X', ' ', ' ', ' ', ' '},
00029             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00030             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '}
00031         };
00032
00033         // Iguala o tabuleiro de **char com os valores inciais
00034         for (int i = 0; i < num_columns_and_rows_reversi; ++i)
00035         {
00036             for (int j = 0; j < num_columns_and_rows_reversi; ++j)
00037                 board_sample[i][j] = board_initial_values[i][j];
00038
00039         }
00040
00041         // Utiliza a função para igualar o tabuleiro criado pelo construtor da classe
00042         // com o tabuleiro **char
00043         Board board(num_columns_and_rows_reversi, num_columns_and_rows_reversi);
00044         board.set_game_board(board_sample);
00045
00046         auto& class_board = board.get_game_board();
00047         for (int i = 0; i < num_columns_and_rows_reversi; ++i)
00048         {
00049             for (int j = 0; j < num_columns_and_rows_reversi; ++j)
00050             // Confere se cada elemento foi copiado adequadamente
00051                 CHECK(class_board[i][j] == board_initial_values[i][j]);
00052
00053         }
00054     }
00055     SUBCASE("Lig4 board")
00056     {
00057         char** board_sample = new char*[num_rows_lig4];
00058         for (int i = 0; i < num_rows_lig4; ++i)
00059             board_sample[i] = new char[num_columns_lig4];
00060
00061         char board_initial_values[num_rows_lig4][num_columns_lig4] =
00062         {
```

```
00063             {' ', ' ', ' ', ' ', ' ', ' ', ' '},
00064             {' ', ' ', 'O', ' ', ' ', ' ', ' '},
00065             {' ', ' ', 'X', 'O', 'X', ' ', ' '},
00066             {' ', ' ', 'O', 'X', 'O', ' ', ' '},
00067             {'O', ' ', 'O', 'O', 'X', 'O', ' '},
00068             {'X', 'X', 'X', 'X', 'X', 'O', ' '},
00069         };
00070 
00071         for (int i = 0; i < num_rows_lig4; ++i)
00072         {
00073             for (int j = 0; j < num_columns_lig4; ++j)
00074                 board_sample[i][j] = board_initial_values[i][j];
00075 
00076         }
00077 
00078         Board board(num_rows_lig4, num_columns_lig4);
00079         board.set_game_board(board_sample);
00080 
00081         auto& class_board = board.get_game_board();
00082         for (int i = 0; i < num_rows_lig4; ++i)
00083         {
00084             for (int j = 0; j < num_columns_lig4; ++j)
00085                 CHECK(class_board[i][j] == board_initial_values[i][j]);
00086         }
00087     }
00088     SUBCASE("Tic tac toe board")
00089     {
00090         char** board_sample = new char*[num_columns_and_rows_tic_tac_toe];
00091         for (int i = 0; i < num_columns_and_rows_tic_tac_toe; ++i)
00092             board_sample[i] = new char[num_columns_and_rows_tic_tac_toe];
00093 
00094         char board_initial_values[num_columns_and_rows_tic_tac_toe][num_columns_and_rows_tic_tac_toe]
      =
00095         {
00096             {'O', ' ', ' '},
00097             {'X', 'O', 'O'},
00098             {'X', ' ', 'X'},
00099 
00100         };
00101 
00102         for (int i = 0; i < num_columns_and_rows_tic_tac_toe; ++i)
00103         {
00104             for (int j = 0; j < num_columns_and_rows_tic_tac_toe; ++j)
00105                 board_sample[i][j] = board_initial_values[i][j];
00106         }
00107 
00108         Board board(num_columns_and_rows_tic_tac_toe, num_columns_and_rows_tic_tac_toe);
00109         board.set_game_board(board_sample);
00110 
00111         auto& class_board = board.get_game_board();
00112         for (int i = 0; i < num_columns_and_rows_tic_tac_toe; ++i)
00113         {
00114             for (int j = 0; j < num_columns_and_rows_tic_tac_toe; ++j)
00115                 CHECK(class_board[i][j] == board_initial_values[i][j]);
00116         }
00117     }
00118 }
00119 
00120 
00121 
00122 
00123 TEST_CASE("Function is_move_inside_board test")
00124 {
00125     SUBCASE("Reversi board")
00126     {
00127     // Constrói um tabuleiro vazio na dimensão do Reversi
00128     Board board(num_columns_and_rows_reversi, num_columns_and_rows_reversi);
00129 
00130     CHECK(board.is_move_inside_board(0,0) == 1);
00131     CHECK(board.is_move_inside_board(0,7) == 1);
00132     CHECK(board.is_move_inside_board(7,0) == 1);
00133     CHECK(board.is_move_inside_board(7,7) == 1);
00134     CHECK(board.is_move_inside_board(3,5) == 1);
00135     CHECK(board.is_move_inside_board(0,-1) == 0);
00136     CHECK(board.is_move_inside_board(5,12) == 0);
00137     CHECK(board.is_move_inside_board(9,3) == 0);
00138     CHECK(board.is_move_inside_board(-15,12) == 0);
00139     }
00140 
00141     SUBCASE("Lig4 board")
00142     {
00143     // Constroi um tabuleiro vazio na dimensão do Lig4
00144     Board board(num_rows_lig4, num_columns_lig4);
00145 
00146     CHECK(board.is_move_inside_board(0,0) == 1);
00147     CHECK(board.is_move_inside_board(0,6) == 1);
00148     CHECK(board.is_move_inside_board(5,0) == 1);
```

```
00149     CHECK(board.is_move_inside_board(5,6) == 1);
00150     CHECK(board.is_move_inside_board(3,5) == 1);
00151     CHECK(board.is_move_inside_board(0,-1) == 0);
00152     CHECK(board.is_move_inside_board(5,12) == 0);
00153     CHECK(board.is_move_inside_board(6,7) == 0);
00154     CHECK(board.is_move_inside_board(-1,12) == 0);
00155     }
00156
00157     SUBCASE("Tic tac toe board")
00158     {
00159     // Constroi um tabuleiro vazio na dimensão do Jogo da Velha
00160     Board board(num_columns_and_rows_tic_tac_toe, num_columns_and_rows_tic_tac_toe);
00161
00162     CHECK(board.is_move_inside_board(0,0) == 1);
00163     CHECK(board.is_move_inside_board(0,2) == 1);
00164     CHECK(board.is_move_inside_board(2,0) == 1);
00165     CHECK(board.is_move_inside_board(2,2) == 1);
00166     CHECK(board.is_move_inside_board(0,1) == 1);
00167     CHECK(board.is_move_inside_board(0,-1) == 0);
00168     CHECK(board.is_move_inside_board(3,3) == 0);
00169     CHECK(board.is_move_inside_board(4,3) == 0);
00170     CHECK(board.is_move_inside_board(-1,1) == 0);
00171     }
00172 }
00173
00174
00175
00176 TEST_CASE("Function is_space_free test")
00177 {
00178     //Passos iniciais seguem a mesma lógica do teste da função set_game_board
00179     char** board_sample = new char*[num_columns_and_rows_reversi];
00180     for (int i = 0; i < num_columns_and_rows_reversi; ++i)
00181         board_sample[i] = new char[num_columns_and_rows_reversi];
00182
00183     char board_initial_values[num_columns_and_rows_reversi][num_columns_and_rows_reversi] =
00184     {
00185         {' ', ' ', ' ', '*', ' ', ' ', ' ', ' ', ' '},
00186         {' ', ' ', 'O', '*', ' ', ' ', ' ', ' ', ' '},
00187         {' ', ' ', 'X', 'O', 'X', ' ', ' ', ' ', ' '},
00188         {' ', 'O', '*', 'X', 'O', '*', '*', ' ', ' '},
00189         {' ', ' ', '*', 'O', 'X', 'O', '*', ' ', ' '},
00190         {' ', ' ', ' ', '*', 'X', ' ', ' ', ' ', ' '},
00191         {' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00192         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '}
00193     };
00194
00195     for (int i = 0; i < num_columns_and_rows_reversi; ++i)
00196     {
00197         for (int j = 0; j < num_columns_and_rows_reversi; ++j)
00198             board_sample[i][j] = board_initial_values[i][j];
00199     }
00200
00201     // Utiliza set_game_board (já testada) para igualar o tabuleiro criado pelo construtor
00202     //da classe com o tabuleiro **char
00203     Board board(num_columns_and_rows_reversi, num_columns_and_rows_reversi);
00204     board.set_game_board(board_sample);
00205
00206     CHECK(board.is_space_free(6,1) == 0);
00207     CHECK(board.is_space_free(3,4) == 0);
00208     CHECK(board.is_space_free(3,2) == 0);
00209     CHECK(board.is_space_free(0,0) == 1);
00210     CHECK(board.is_space_free(5,5) == 1);
00211 }
00212
```

5.29 Referência do Arquivo tests/Connect4Class_test.cpp

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

Gráfico de dependência de inclusões para Connect4Class_test.cpp:

Definições e Macros

- #define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN

Funções

- TEST_CASE ("Connect4 Class Tests")

5.29.1 Definições e macros

5.29.1.1 DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN

```
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
```

Definição na linha 1 do arquivo Connect4Class_test.cpp.

5.29.2 Funções

5.29.2.1 TEST_CASE()

```
TEST_CASE (
            "Connect4 Class Tests"  )
```

Definição na linha 6 do arquivo Connect4Class_test.cpp.

5.30 Connect4Class_test.cpp

Ir para a documentação desse arquivo.

```
00001 #define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
00002 #include "doctest.h"
00003 #include "Connect4.hpp"
00004 #include <iostream>
00005
00006 TEST_CASE("Connect4 Class Tests")
00007 {
00008     Connect4 game;
00009
00010     SUBCASE("Initialization Test")
00011     {
00012         CHECK(game.get_current_player() == 'X'); // Jogador inicial deve ser 'X', que é referente ao
      player 1
00013         CHECK(game.is_board_full() == false);    // Tabuleiro não deve estar cheio no início
00014     }
00015
00016     SUBCASE("Is valid Move Test")
00017     {
00018         // Verifica movimentos válidos e inválidos
00019         CHECK(game.is_valid_move(1) == true);
00020         CHECK(game.is_valid_move(7) == true);
00021         CHECK(game.is_valid_move(0) == false);
00022         CHECK(game.is_valid_move(8) == false);
00023         CHECK(game.is_valid_move(99) == false);
00024     }
00025
00026     SUBCASE("Make Move Test")
00027     {
00028         // Verifica o jogadore inicial
00029         CHECK(game.get_current_player() == 'X');
00030
00031         game.make_move(1);
00032
00033         // Verifica se o local foi ocupado da maneira certa
00034         CHECK(game.get_space(5, 0) == 'X');
00035
00036         // Jogador deve alternar para 'O'
00037         game.set_current_player('O');
00038
00039         // Verifica se o jogador atual foi alterado corretamente
```

```
00040         CHECK(game.get_current_player() == 'O');
00041
00042         // Verifica outras casas do tabuleiro
00043         CHECK(game.get_space(5, 1) == ' ');
00044         CHECK(game.get_space(4, 0) == ' ');
00045
00046         // Verifica após mais uma jogada
00047         game.make_move(1);
00048         CHECK(game.get_space(5, 1) == ' ');
00049         CHECK(game.get_space(4, 0) == 'O');
00050     }
00051
00052     SUBCASE("Win Condition Test - Horizontal")
00053     {
00054         // Teste 1: Vitória horizontal
00055         Connect4 game_horizontal1;
00056         game_horizontal1.make_move(1); // X
00057         game_horizontal1.make_move(1); // O
00058         game_horizontal1.make_move(2); // X
00059         game_horizontal1.make_move(2); // O
00060         game_horizontal1.make_move(3); // X
00061         game_horizontal1.make_move(3); // O
00062         game_horizontal1.make_move(4); // X
00063         CHECK(game_horizontal1.check_win() == true); // X deve vencer
00064
00065         // Teste 2: Vitória horizontal
00066         Connect4 game_horizontal2;
00067         game_horizontal2.make_move(1); // X
00068         game_horizontal2.make_move(1); // O
00069         game_horizontal2.make_move(1); // X
00070         game_horizontal2.make_move(1); // O
00071         game_horizontal2.make_move(1); // X
00072         game_horizontal2.make_move(1); // O
00073         game_horizontal2.make_move(2); // X
00074         game_horizontal2.make_move(2); // O
00075         game_horizontal2.make_move(2); // X
00076         game_horizontal2.make_move(2); // O
00077         game_horizontal2.make_move(3); // X
00078         game_horizontal2.make_move(3); // O
00079         game_horizontal2.make_move(3); // X
00080         game_horizontal2.make_move(3); // O
00081         game_horizontal2.make_move(4); // X
00082         CHECK(game_horizontal2.check_win() == true); // X deve vencer
00083
00084         // Teste 3: Vitória horizontal na linha 3
00085         Connect4 game_horizontal3;
00086         game_horizontal3.make_move(1); // X
00087         game_horizontal3.make_move(2); // O
00088         game_horizontal3.make_move(3); // X
00089         game_horizontal3.make_move(4); // O
00090         game_horizontal3.make_move(5); // X
00091         game_horizontal3.make_move(6); // O
00092         game_horizontal3.make_move(1); // X
00093         game_horizontal3.make_move(2); // O
00094         game_horizontal3.make_move(3); // X
00095         game_horizontal3.make_move(4); // O
00096         game_horizontal3.make_move(5); // X
00097         game_horizontal3.make_move(6); // O
00098         game_horizontal3.make_move(4); // X
00099         CHECK(game_horizontal3.check_win() == true); // X deve vencer
00100     }
00101
00102     SUBCASE("Win Condition Test - Vertical")
00103     {
00104         // Teste 1: Vitória vertical na coluna 1
00105         Connect4 game_vertical1;
00106         game_vertical1.make_move(1); // X
00107         game_vertical1.make_move(2); // O
00108         game_vertical1.make_move(1); // X
00109         game_vertical1.make_move(2); // O
00110         game_vertical1.make_move(1); // X
00111         game_vertical1.make_move(2); // O
00112         game_vertical1.make_move(1); // X
00113         CHECK(game_vertical1.check_win() == true); // X deve vencer
00114
00115         // Teste 2: Vitória vertical na coluna 4
00116         Connect4 game_vertical2;
00117         game_vertical2.make_move(4); // X
00118         game_vertical2.make_move(1); // O
00119         game_vertical2.make_move(4); // X
00120         game_vertical2.make_move(1); // O
00121         game_vertical2.make_move(4); // X
00122         game_vertical2.make_move(1); // O
00123         game_vertical2.make_move(4); // X
00124         CHECK(game_vertical2.check_win() == true); // X deve vencer
00125
00126         // Teste 3: Vitória vertical na coluna 7
```

```
00127         Connect4 game_vertical3;
00128         game_vertical3.make_move(7); // X
00129         game_vertical3.make_move(1); // O
00130         game_vertical3.make_move(7); // X
00131         game_vertical3.make_move(1); // O
00132         game_vertical3.make_move(7); // X
00133         game_vertical3.make_move(1); // O
00134         game_vertical3.make_move(7); // X
00135         CHECK(game_vertical3.check_win() == true); // X deve vencer
00136     }
00137
00138     SUBCASE("Win Condition Test - Diagonal Direita")
00139     {
00140         // Teste 1: Vitória diagonal direita
00141         Connect4 game_diagonal_right1;
00142         game_diagonal_right1.make_move(1); // X
00143         game_diagonal_right1.make_move(2); // O
00144         game_diagonal_right1.make_move(2); // X
00145         game_diagonal_right1.make_move(3); // O
00146         game_diagonal_right1.make_move(3); // X
00147         game_diagonal_right1.make_move(4); // O
00148         game_diagonal_right1.make_move(3); // X
00149         game_diagonal_right1.make_move(4); // O
00150         game_diagonal_right1.make_move(4); // X
00151         game_diagonal_right1.make_move(5); // O
00152         game_diagonal_right1.make_move(4); // X
00153         CHECK(game_diagonal_right1.check_win() == true); // X deve vencer
00154
00155         // Teste 2: Vitória diagonal direita
00156         Connect4 game_diagonal_right2;
00157         game_diagonal_right2.make_move(2); // X
00158         game_diagonal_right2.make_move(3); // O
00159         game_diagonal_right2.make_move(3); // X
00160         game_diagonal_right2.make_move(4); // O
00161         game_diagonal_right2.make_move(4); // X
00162         game_diagonal_right2.make_move(5); // O
00163         game_diagonal_right2.make_move(4); // X
00164         game_diagonal_right2.make_move(5); // O
00165         game_diagonal_right2.make_move(5); // X
00166         game_diagonal_right2.make_move(6); // O
00167         game_diagonal_right2.make_move(5); // X
00168         CHECK(game_diagonal_right2.check_win() == true); // X deve vencer
00169
00170         // Teste 3: Vitória diagonal direita
00171         Connect4 game_diagonal_right3;
00172         game_diagonal_right3.make_move(3); // X
00173         game_diagonal_right3.make_move(4); // O
00174         game_diagonal_right3.make_move(4); // X
00175         game_diagonal_right3.make_move(5); // O
00176         game_diagonal_right3.make_move(5); // X
00177         game_diagonal_right3.make_move(6); // O
00178         game_diagonal_right3.make_move(5); // X
00179         game_diagonal_right3.make_move(6); // O
00180         game_diagonal_right3.make_move(6); // X
00181         game_diagonal_right3.make_move(7); // O
00182         game_diagonal_right3.make_move(6); // X
00183         CHECK(game_diagonal_right3.check_win() == true); // X deve vencer
00184     }
00185
00186     SUBCASE("Win Condition Test - Diagonal Esquerda")
00187     {
00188         // Teste 1: Vitória diagonal esquerda
00189         Connect4 game_diagonal_left1;
00190         game_diagonal_left1.make_move(4); // X
00191         game_diagonal_left1.make_move(3); // O
00192         game_diagonal_left1.make_move(3); // X
00193         game_diagonal_left1.make_move(2); // O
00194         game_diagonal_left1.make_move(2); // X
00195         game_diagonal_left1.make_move(1); // O
00196         game_diagonal_left1.make_move(2); // X
00197         game_diagonal_left1.make_move(1); // O
00198         game_diagonal_left1.make_move(1); // X
00199         game_diagonal_left1.make_move(7); // O
00200         game_diagonal_left1.make_move(1); // X
00201         CHECK(game_diagonal_left1.check_win() == true); // X deve vencer
00202
00203         // Teste 2: Vitória diagonal esquerda
00204         Connect4 game_diagonal_left2;
00205         game_diagonal_left2.make_move(5); // X
00206         game_diagonal_left2.make_move(4); // O
00207         game_diagonal_left2.make_move(4); // X
00208         game_diagonal_left2.make_move(3); // O
00209         game_diagonal_left2.make_move(3); // X
00210         game_diagonal_left2.make_move(2); // O
00211         game_diagonal_left2.make_move(3); // X
00212         game_diagonal_left2.make_move(2); // O
00213         game_diagonal_left2.make_move(2); // X
```

```
00214         game_diagonal_left2.make_move(7); // O
00215         game_diagonal_left2.make_move(2); // X
00216         CHECK(game_diagonal_left2.check_win() == true); // X deve vencer
00217 
00218         // Teste 3: Vitória diagonal esquerda
00219         Connect4 game_diagonal_left3;
00220         game_diagonal_left3.make_move(6); // X
00221         game_diagonal_left3.make_move(5); // O
00222         game_diagonal_left3.make_move(5); // X
00223         game_diagonal_left3.make_move(4); // O
00224         game_diagonal_left3.make_move(4); // X
00225         game_diagonal_left3.make_move(3); // O
00226         game_diagonal_left3.make_move(4); // X
00227         game_diagonal_left3.make_move(3); // O
00228         game_diagonal_left3.make_move(3); // X
00229         game_diagonal_left3.make_move(7); // O
00230         game_diagonal_left3.make_move(3); // X
00231         CHECK(game_diagonal_left3.check_win() == true); // X deve vencer
00232     }
00233 
00234     SUBCASE("Board Full Test")
00235     {
00236         Connect4 game_full;
00237 
00238         // Verificação antes de completar
00239         CHECK(game_full.is_board_full() == false);
00240 
00241         // Preenche o tabuleiro completamente
00242         for (int col = 1; col <= 7; ++col) {
00243             for (int row = 0; row < 6; ++row) {
00244                 game_full.make_move(col);
00245             }
00246         }
00247 
00248         // Tabuleiro deve estar cheio
00249         CHECK(game_full.is_board_full() == true);
00250     }
00251 }
```

5.31 Referência do Arquivo tests/PlayerClass_test.cpp

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

Gráfico de dependência de inclusões para PlayerClass_test.cpp:

Definições e Macros

- #define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN

Funções

- TEST_CASE ("method register_player test")
- TEST_CASE ("method remove_player test")
- TEST_CASE ("method add_win test")
- TEST_CASE ("method add_loss test")
- TEST_CASE ("method find_player_in_list test")

5.31.1 Definições e macros

5.31.1.1 DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN

```
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
```

Definição na linha 1 do arquivo PlayerClass_test.cpp.

5.31.2 Funções

5.31.2.1 TEST_CASE() [1/5]

```
TEST_CASE (
            "method add_loss test"  )
```

Definição na linha 91 do arquivo PlayerClass_test.cpp.

5.31.2.2 TEST_CASE() [2/5]

```
TEST_CASE (
            "method add_win test"  )
```

Definição na linha 64 do arquivo PlayerClass_test.cpp.

5.31.2.3 TEST_CASE() [3/5]

```
TEST_CASE (
            "method find_player_in_list test"  )
```

Definição na linha 118 do arquivo PlayerClass_test.cpp.

5.31.2.4 TEST_CASE() [4/5]

```
TEST_CASE (
            "method register_player test"  )
```

Definição na linha 6 do arquivo PlayerClass_test.cpp.

5.31.2.5 TEST_CASE() [5/5]

```
TEST_CASE (
            "method remove_player test"  )
```

Definição na linha 33 do arquivo PlayerClass_test.cpp.

5.32 PlayerClass_test.cpp

Ir para a documentação desse arquivo.

```
00001 #define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
00002 #include "doctest.h"
00003 #include "Player.hpp"
00004 #include <algorithm>
00005
00006 TEST_CASE("method register_player test")
00007 {
00008     std::list<Player> player_list;
00009     Player player1;
00010     Player::register_player(player1, player_list);
00011     CHECK(player1 == player_list.back());
00012
00013     player1.set_username("Apelido"); player1.set_name("Nome");
00014     player1.set_num_loss("Reversi", -1); player1.set_num_loss("Lig4", 2);
      player1.set_num_loss("Velha", -3);
00015     player1.set_num_win("Reversi", 6); player1.set_num_win("Lig4", -5); player1.set_num_win("Velha",
      4);
00016     Player::register_player(player1, player_list);
00017     CHECK(player1 == player_list.back());
00018
00019     Player player2("Nome sobrenome", "Apelido1");
00020     Player::register_player(player2, player_list);
00021     CHECK(player2 == player_list.back());
00022
00023     Player player3("Nome sobrenome", "Apelido2", {{"Reversi", 1}, {"Lig4", 2}, {"Velha", 3}},
      {{"Reversi", 4}, {"Lig4", 5}, {"Velha", 6}});
00024     Player::register_player(player3, player_list);
00025     CHECK(player3 == player_list.back());
00026
00027     Player *player4 = new Player("Nome sobrenome", "Apelido3");
00028     Player::register_player(*player4, player_list);
00029     CHECK(*player4 == player_list.back());
00030     delete player4;
00031 }
00032
00033 TEST_CASE("method remove_player test")
00034 {
00035     Player player1, player2("nome", "Apelido1"), player3("nome sobrenome", "Apelido2");
00036     std::list<Player> player_list = {player1, player2, player3};
00037     Player::remove_player("", player_list);
00038     std::list<Player>::iterator it;
00039     for (it = player_list.begin(); it != player_list.end(); it++)
00040         if (*it == player1)
00041             CHECK(false);
00042
00043     Player::remove_player("Apelido2", player_list);
00044     for (it = player_list.begin(); it != player_list.end(); it++)
00045         if (*it == player3)
00046             CHECK(false);
00047
00048     Player player4("Nome sobrenome", "Apelido4", {{"Reversi", 1}, {"Lig4", 2}, {"Velha", 3}},
      {{"Reversi", 4}, {"Lig4", 5}, {"Velha", 6}});
00049     player_list.push_back(player4);
00050     Player::remove_player("Apelido4", player_list);
00051     for (it = player_list.begin(); it != player_list.end(); it++)
00052         if (*it == player4)
00053             CHECK(false);
00054
00055     Player *player5 = new Player("nome", "Apelido3");
00056     player_list.push_back(*player5);
00057     Player::remove_player("Apelido3", player_list);
00058     for (it = player_list.begin(); it != player_list.end(); it++)
00059         if (*it == *player5)
00060             CHECK(false);
00061     delete player5;
00062 }
00063
00064 TEST_CASE("method add_win test"){
00065     Player player;
00066     player.add_win("Reversi");
00067     player.add_win("Lig4");
00068     player.add_win("Velha");
00069     std::map<std::string, int> win_test = {{"Reversi", 1}, {"Lig4", 1}, {"Velha", 1}};
00070     CHECK(player.get_num_win() == win_test);
00071
00072     player.set_num_win("Velha", -13);
00073     player.add_win("Velha");
00074     win_test["Velha"] = -12;
00075     CHECK(player.get_num_win() == win_test);
00076
00077     player.set_num_win("Reversi", 2147483647);
00078     player.add_win("Reversi");
```

```
00079     win_test["Reversi"] = 2147483648;
00080     CHECK(player.get_num_win() == win_test);
00081
00082     player.set_num_win("Lig4", -2147483648);
00083     player.add_win("Lig4");
00084     win_test["Lig4"] = -2147483647;
00085     CHECK(player.get_num_win() == win_test);
00086
00087     player.add_win("Game");
00088     CHECK(player.get_num_win() == win_test);
00089 }
00090
00091 TEST_CASE("method add_loss test"){
00092     Player player;
00093     player.add_loss("Reversi");
00094     player.add_loss("Lig4");
00095     player.add_loss("Velha");
00096     std::map<std::string, int> loss_test = {{"Reversi", 1}, {"Lig4", 1}, {"Velha", 1}};
00097     CHECK(player.get_num_loss() == loss_test);
00098
00099     player.set_num_loss("Velha", -13);
00100     player.add_loss("Velha");
00101     loss_test["Velha"] = -12;
00102     CHECK(player.get_num_loss() == loss_test);
00103
00104     player.set_num_loss("Reversi", 2147483647);
00105     player.add_loss("Reversi");
00106     loss_test["Reversi"] = 2147483648;
00107     CHECK(player.get_num_loss() == loss_test);
00108
00109     player.set_num_loss("Lig4", -2147483648);
00110     player.add_loss("Lig4");
00111     loss_test["Lig4"] = -2147483647;
00112     CHECK(player.get_num_loss() == loss_test);
00113
00114     player.add_loss("Game");
00115     CHECK(player.get_num_loss() == loss_test);
00116 }
00117
00118 TEST_CASE("method find_player_in_list test"){
00119     Player player1("", "Apelido1");
00120     std::list<Player> player_list;
00121     player_list.push_back(player1);
00122     CHECK(Player::find_player_in_list(player_list, "Apelido1") == &(*player_list.begin()));
00123
00124     Player player2("nome", "Apelido2"), player3("nome sobrenome", "Apelido3");
00125     player_list.push_back(player2);
00126     player_list.push_back(player3);
00127     std::list<Player>::iterator it;
00128     it = std::next(player_list.begin(), 1);
00129     CHECK(Player::find_player_in_list(player_list, "Apelido2") == &(*it));
00130     CHECK(Player::find_player_in_list(player_list, "Apelido3") == &(*player_list.rbegin()));
00131
00132     Player *player4 = new Player("", "Apelido4");
00133     player_list.push_back(*player4);
00134     CHECK(Player::find_player_in_list(player_list, "Apelido4") == &(*player_list.rbegin()));
00135     delete player4;
00136 }
```

5.33 Referência do Arquivo tests/ReversiClass_test.cpp

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

Gráfico de dependência de inclusões para ReversiClass_test.cpp:

Definições e Macros

- #define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN

Funções

- void create_game_board_situation (char game_board_situation[num_columns_and_rows_reversi][num_columns_and_rows_re
  Reversi &reversi_game)

  Aloca um tabuleiro com situações de teste no Reversi sem a necessidade de uma partida real.
- TEST_CASE ("Function is_there_player_piece_at_the_direction test")
- TEST_CASE ("Function is_space_free_reversi test")
- TEST_CASE ("Function is_valid_move test")
- TEST_CASE ("Function is_there_direction_that_captures_opponent")
- TEST_CASE ("Function is_there_valid_move_for_player test")
- TEST_CASE ("Function check_win test")
- TEST_CASE ("Function control_num_pieces_player test")
- TEST_CASE ("Function flip_pieces test")
- TEST_CASE ("Function make_move test")
- TEST_CASE ("Function find_all_directions_that_make_move test")
- TEST_CASE ("Function process_move")

Variáveis

- const int num_columns_and_rows_reversi = 8

5.33.1 Definições e macros

5.33.1.1 DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN

```
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
```

Definição na linha 1 do arquivo ReversiClass_test.cpp.

5.33.2 Funções

5.33.2.1 create_game_board_situation()

```
void create_game_board_situation (
            char game_board_situation[num_columns_and_rows_reversi][num_columns_and_rows_↩
reversi],
            Reversi & reversi_game )
```

Aloca um tabuleiro com situações de teste no Reversi sem a necessidade de uma partida real.

Parâmetros

game_board_situation	Tabuleiro que simula situações de jogo
reversi_game	Instância da classe Reversi

Definição na linha 16 do arquivo ReversiClass_test.cpp.

5.33.2.2 TEST_CASE() [1/11]

```
TEST_CASE (
             "Function check_win test"  )
```

Definição na linha 217 do arquivo ReversiClass_test.cpp.

5.33.2.3 TEST_CASE() [2/11]

```
TEST_CASE (
             "Function control_num_pieces_player test"  )
```

Definição na linha 294 do arquivo ReversiClass_test.cpp.

5.33.2.4 TEST_CASE() [3/11]

```
TEST_CASE (
             "Function find_all_directions_that_make_move test"  )
```

Definição na linha 464 do arquivo ReversiClass_test.cpp.

5.33.2.5 TEST_CASE() [4/11]

```
TEST_CASE (
             "Function flip_pieces test"  )
```

Definição na linha 312 do arquivo ReversiClass_test.cpp.

5.33.2.6 TEST_CASE() [5/11]

```
TEST_CASE (
             "Function is_space_free_reversi test"  )
```

Definição na linha 56 do arquivo ReversiClass_test.cpp.

5.33.2.7 TEST_CASE() [6/11]

```
TEST_CASE (
             "Function is_there_direction_that_captures_opponent"  )
```

Definição na linha 146 do arquivo ReversiClass_test.cpp.

5.33.2.8 TEST_CASE() [7/11]

```
TEST_CASE (
             "Function is_there_player_piece_at_the_direction test"  )
```

Definição na linha 25 do arquivo ReversiClass_test.cpp.

5.33.2.9 TEST_CASE() [8/11]

```
TEST_CASE (
            "Function is_there_valid_move_for_player test"  )
```

Definição na linha 169 do arquivo ReversiClass_test.cpp.

5.33.2.10 TEST_CASE() [9/11]

```
TEST_CASE (
            "Function is_valid_move test"  )
```

Definição na linha 82 do arquivo ReversiClass_test.cpp.

5.33.2.11 TEST_CASE() [10/11]

```
TEST_CASE (
            "Function make_move test"  )
```

Definição na linha 406 do arquivo ReversiClass_test.cpp.

5.33.2.12 TEST_CASE() [11/11]

```
TEST_CASE (
            "Function process_move"  )
```

Definição na linha 511 do arquivo ReversiClass_test.cpp.

5.33.3 Variáveis

5.33.3.1 num_columns_and_rows_reversi

```
const int num_columns_and_rows_reversi = 8
```

Definição na linha 8 do arquivo ReversiClass_test.cpp.

5.34 ReversiClass_test.cpp

Ir para a documentação desse arquivo.

```
00001 #define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
00002 #include "doctest.h"
00003 #include "Reversi.hpp"
00004 #include <iostream>
00005 #include <array>
00006 using namespace std;
00007
00008 const int num_columns_and_rows_reversi = 8;
00009
00016 void create_game_board_situation(char
      game_board_situation[num_columns_and_rows_reversi][num_columns_and_rows_reversi], Reversi
      &reversi_game){
00017     char *board[num_columns_and_rows_reversi];
00018     for (int i = 0; i < num_columns_and_rows_reversi; i++){
00019         board[i] = game_board_situation[i];
00020     }
00021
00022     reversi_game.get_game_board().set_game_board(board);
00023 }
00024
00025 TEST_CASE("Function is_there_player_piece_at_the_direction test"){
00026     Reversi reversi_game;
00027     char game_board_situation[num_columns_and_rows_reversi][num_columns_and_rows_reversi] = {
00028         {' ', ' ', ' ', '*', ' ', ' ', ' ', ' '},
00029         {' ', ' ', 'O', '*', ' ', ' ', ' ', ' '},
00030         {' ', ' ', 'X', 'O', 'X', ' ', ' ', ' '},
00031         {' ', ' ', '*', 'X', 'O', '*', '*', ' '},
00032         {' ', ' ', '*', 'O', 'X', 'O', '*', ' '},
00033         {' ', ' ', ' ', '*', 'X', ' ', ' ', ' '},
00034         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00035         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '}
00036
00037     };
00038     create_game_board_situation(game_board_situation, reversi_game);
00039
00040     SUBCASE("There is * at the direciton"){
00041         (reversi_game.is_there_player_piece_at_the_direction('O', {-1, 0}, {1, 3}) == 0);
00042     }
00043     SUBCASE("There is player piece at the direction"){
00044         CHECK(reversi_game.is_there_player_piece_at_the_direction('X', {-1, -1}, {5, 5}) == 1);
00045         CHECK(reversi_game.is_there_player_piece_at_the_direction('O', {-1, 1}, {5, 2}) == 1);
00046     }
00047     SUBCASE("There is empty spaces at the direction"){
00048         CHECK(reversi_game.is_there_player_piece_at_the_direction('X', {0, 1}, {2, 4}) == 0);
00049     }
00050     SUBCASE("There is opponents pieces at the direction"){
00051
00052         CHECK(reversi_game.is_there_player_piece_at_the_direction('X', {0, -1}, {4, 4}) == 0);
00053     }
00054 }
00055
00056 TEST_CASE("Function is_space_free_reversi test"){
00057     Reversi reversi_game;
00058     char game_board_situation[num_columns_and_rows_reversi][num_columns_and_rows_reversi] = {
00059         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00060         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00061         {' ', 'X', ' ', '*', ' ', 'X', ' ', ' '},
00062         {' ', ' ', 'X', 'O', 'X', ' ', ' ', ' '},
00063         {' ', '*', 'O', 'X', 'O', '*', ' ', ' '},
00064         {' ', 'O', '*', ' ', '*', 'O', ' ', ' '},
00065         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00066         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00067     };
00068     create_game_board_situation(game_board_situation, reversi_game);
00069     SUBCASE("The space has * "){
00070
00071         CHECK(reversi_game.is_space_free_reversi(2, 3) == 1);
00072     }
00073     SUBCASE("Space has a player piece"){
00074         CHECK(reversi_game.is_space_free_reversi(2, 5) == 0);
00075         CHECK(reversi_game.is_space_free_reversi(5, 5) == 0);
00076     }
00077     SUBCASE("Space is empty"){
00078         CHECK(reversi_game.is_space_free_reversi(0, 3) == 1);
00079     }
00080 }
00081
00082 TEST_CASE("Function is_valid_move test"){
00083     Reversi reversi_game;
00084     char game_board_situation1[num_columns_and_rows_reversi][num_columns_and_rows_reversi] = {
00085         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00086         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
```

```
00087         {' ', 'X', ' ', ' ', ' ', 'X', ' ', ' '},
00088         {' ', ' ', 'O', 'O', 'X', ' ', ' ', ' '},
00089         {' ', ' ', 'X', 'X', 'O', ' ', ' ', ' '},
00090         {' ', 'O', ' ', ' ', ' ', 'O', ' ', ' '},
00091         {' ', ' ', ' ', ' ', ' ', ' ', 'X', ' '},
00092         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00093 
00094     };
00095     create_game_board_situation(game_board_situation1, reversi_game);
00096 
00097     std::array<int, 2> coordinates = {1, 7};
00098 
00099     SUBCASE("X Move doesn't capture opponents pieces"){
00100         CHECK(reversi_game.is_valid_move(coordinates, 'X') == 0);
00101     }
00102     SUBCASE("X captures opponents pieces"){
00103         coordinates = {2, 3};
00104         CHECK(reversi_game.is_valid_move(coordinates, 'X') == 1);
00105     }
00106     SUBCASE("X captures opponents pieces in more tha one direction"){
00107         coordinates = {2, 2};
00108         CHECK(reversi_game.is_valid_move(coordinates, 'X') == 1);
00109     }
00110     SUBCASE("X move is out of the board"){
00111         coordinates = {3, 9};
00112         CHECK(reversi_game.is_valid_move(coordinates, 'X') == 0);
00113     }
00114 
00115     char game_board_situation2[num_columns_and_rows_reversi][num_columns_and_rows_reversi] = {
00116         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00117         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00118         {' ', ' ', 'O', ' ', ' ', 'X', ' ', ' '},
00119         {'X', ' ', 'O', 'O', 'X', ' ', ' ', ' '},
00120         {' ', 'X', 'O', 'O', 'O', ' ', ' ', ' '},
00121         {' ', ' ', 'X', 'X', 'O', ' ', ' ', ' '},
00122         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00123         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '}};
00124     create_game_board_situation(game_board_situation2, reversi_game);
00125 
00126     SUBCASE("O Move doesn't capture opponents pieces"){
00127         coordinates = {2, 1};
00128         CHECK(reversi_game.is_valid_move(coordinates, 'O') == 0);
00129     }
00130     SUBCASE("O captures opponents pieces"){
00131         coordinates = {2, 4};
00132         CHECK(reversi_game.is_valid_move(coordinates, 'O') == 1);
00133         coordinates = {3, 5};
00134         CHECK(reversi_game.is_valid_move(coordinates, 'O') == 1);
00135     }
00136     SUBCASE("O captures opponents pieces in more tha one direction"){
00137         coordinates = {6, 2};
00138         CHECK(reversi_game.is_valid_move(coordinates, 'O') == 1);
00139     }
00140     SUBCASE("O move is out of the board"){
00141         coordinates = {9, 0};
00142         CHECK(reversi_game.is_valid_move(coordinates, 'O') == 0);
00143     }
00144 }
00145 
00146 TEST_CASE("Function is_there_direction_that_captures_opponent"){
00147     Reversi reversi_game;
00148     char game_board_situation[num_columns_and_rows_reversi][num_columns_and_rows_reversi] = {
00149         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00150         {' ', ' ', ' ', ' ', ' ', 'X', ' ', '*'},
00151         {' ', ' ', 'X', 'X', ' ', ' ', '*', 'O'},
00152         {' ', 'O', 'X', 'O', ' ', ' ', ' ', ' '},
00153         {'X', ' ', 'O', 'O', 'O', ' ', 'X', ' '},
00154         {' ', ' ', ' ', ' ', ' ', 'O', ' ', ' '},
00155         {' ', ' ', ' ', ' ', ' ', ' ', 'X', ' '},
00156         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00157     };
00158     create_game_board_situation(game_board_situation, reversi_game);
00159     CHECK(reversi_game.is_there_direction_that_captures_opponent({1, 2}, 'O') == 1);
00160     CHECK(reversi_game.is_there_direction_that_captures_opponent({5, 3}, 'X') == 1);
00161     CHECK(reversi_game.is_there_direction_that_captures_opponent({7, 7}, 'O') == 1);
00162     CHECK(reversi_game.is_there_direction_that_captures_opponent({6, 4}, 'X') == 1);
00163     CHECK(reversi_game.is_there_direction_that_captures_opponent({4, 5}, 'X') == 0);
00164     CHECK(reversi_game.is_there_direction_that_captures_opponent({7, 3}, 'O') == 0);
00165     CHECK(reversi_game.is_there_direction_that_captures_opponent({7, 7}, 'X') == 0);
00166     CHECK(reversi_game.is_there_direction_that_captures_opponent({1, 6}, 'O') == 0);
00167 }
00168 
00169 TEST_CASE("Function is_there_valid_move_for_player test"){
00170     Reversi reversi_game;
00171     SUBCASE("A player Won"){
00172         char game_board_situation1[num_columns_and_rows_reversi][num_columns_and_rows_reversi] = {
00173             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
```

```
00174             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00175             {' ', 'X', ' ', ' ', ' ', 'X', ' ', ' '},
00176             {' ', ' ', 'X', 'X', 'X', ' ', ' ', ' '},
00177             {' ', ' ', 'X', 'X', 'X', ' ', ' ', ' '},
00178             {' ', 'X', ' ', ' ', ' ', 'X', ' ', ' '},
00179             {' ', ' ', ' ', ' ', ' ', ' ', 'X', ' '},
00180             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00181
00182         };
00183         create_game_board_situation(game_board_situation1, reversi_game);
00184         CHECK(reversi_game.is_there_valid_move_for_player('O') == 0);
00185         CHECK(reversi_game.is_there_valid_move_for_player('X') == 0);
00186     }
00187     SUBCASE("A player has valid move and the oponnent doesn't"){
00188         char game_board_situation2[num_columns_and_rows_reversi][num_columns_and_rows_reversi] = {
00189             {'X', ' ', 'X', 'X', 'X', 'X', 'X', ' '},
00190             {'X', 'X', 'X', 'X', 'X', 'X', ' ', 'X'},
00191             {'X', 'X', 'X', 'X', 'X', 'X', 'X', ' '},
00192             {'X', 'X', 'X', 'X', 'X', 'X', ' ', 'X'},
00193             {'X', 'X', 'X', 'X', 'X', 'X', 'X', ' '},
00194             {' ', ' ', ' ', 'O', ' ', 'O', ' ', ' '},
00195             {' ', ' ', ' ', ' ', ' ', ' ', 'O', ' '},
00196             {' ', ' ', ' ', ' ', ' ', ' ', ' ', 'O'}};
00197         create_game_board_situation(game_board_situation2, reversi_game);
00198         CHECK(reversi_game.is_there_valid_move_for_player('O') == 0);
00199         CHECK(reversi_game.is_there_valid_move_for_player('X') == 1);
00200     }
00201     SUBCASE("Both players has valid moves"){
00202         char game_board_situation3[num_columns_and_rows_reversi][num_columns_and_rows_reversi] = {
00203             {'X', ' ', 'X', ' ', ' ', ' ', ' ', ' '},
00204             {'X', 'X', 'X', ' ', ' ', ' ', ' ', ' '},
00205             {'X', 'X', 'X', ' ', 'X', ' ', ' ', ' '},
00206             {'X', 'X', 'X', 'X', 'O', 'O', '*', ' '},
00207             {'X', 'O', 'O', 'O', 'O', 'O', '*', ' '},
00208             {'*', '*', '*', 'O', '*', 'O', ' ', ' '},
00209             {' ', ' ', ' ', ' ', '*', '*', ' ', 'O', ' '},
00210             {' ', ' ', ' ', ' ', ' ', ' ', ' ', 'O'}};
00211         create_game_board_situation(game_board_situation3, reversi_game);
00212         CHECK(reversi_game.is_there_valid_move_for_player('O') == 1);
00213         CHECK(reversi_game.is_there_valid_move_for_player('X') == 1);
00214     }
00215 }
00216
00217 TEST_CASE("Function check_win test"){
00218     Reversi reversi_game;
00219     SUBCASE("O won"){
00220         char game_board_situation1[num_columns_and_rows_reversi][num_columns_and_rows_reversi] = {
00221             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00222             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00223             {' ', 'O', ' ', ' ', ' ', 'O', ' ', ' '},
00224             {' ', ' ', 'O', 'O', 'O', ' ', ' ', ' '},
00225             {' ', ' ', ' ', 'O', 'O', 'O', ' ', ' '},
00226             {' ', 'O', ' ', ' ', ' ', ' ', 'O', ' '},
00227             {' ', ' ', ' ', ' ', ' ', ' ', ' ', 'O'},
00228             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00229
00230         };
00231         create_game_board_situation(game_board_situation1, reversi_game);
00232         CHECK(reversi_game.check_win(reversi_game.is_there_valid_move_for_player('O'), 'X') == 1);
00233     }
00234
00235     SUBCASE("X won"){
00236         char game_board_situation2[num_columns_and_rows_reversi][num_columns_and_rows_reversi] = {
00237             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00238             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00239             {' ', 'X', ' ', ' ', ' ', 'X', ' ', ' '},
00240             {' ', ' ', 'X', 'X', 'X', ' ', ' ', ' '},
00241             {' ', ' ', 'X', 'X', 'X', ' ', ' ', ' '},
00242             {' ', 'X', ' ', ' ', ' ', 'X', ' ', ' '},
00243             {' ', ' ', ' ', ' ', ' ', ' ', 'X', ' '},
00244             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00245
00246         };
00247         create_game_board_situation(game_board_situation2, reversi_game);
00248         CHECK(reversi_game.check_win(reversi_game.is_there_valid_move_for_player('X'), 'O') == 1);
00249     }
00250
00251     SUBCASE("No one won yet"){
00252         char game_board_situation3[num_columns_and_rows_reversi][num_columns_and_rows_reversi] = {
00253             {'X', ' ', 'X', ' ', ' ', ' ', ' ', ' '},
00254             {'X', 'X', 'X', ' ', ' ', ' ', ' ', ' '},
00255             {'X', 'X', 'X', ' ', 'X', ' ', ' ', ' '},
00256             {'X', 'X', 'X', 'X', 'O', 'O', '*', ' '},
00257             {'X', 'O', 'O', 'O', 'O', 'O', '*', ' '},
00258             {'*', '*', '*', 'O', '*', 'O', ' ', ' '},
00259             {' ', ' ', ' ', ' ', '*', '*', ' ', 'O', ' '},
00260             {' ', ' ', ' ', ' ', ' ', ' ', ' ', 'O'}};
```

```
00261         create_game_board_situation(game_board_situation3, reversi_game);
00262         CHECK(reversi_game.check_win(reversi_game.is_there_valid_move_for_player('O'), 'X') == 0);
00263         CHECK(reversi_game.check_win(reversi_game.is_there_valid_move_for_player('X'), 'O') == 0);
00264     }
00265     SUBCASE("X has valid moves, but O doesn't. Game is not over yet"){
00266         char game_board_situation4[num_columns_and_rows_reversi][num_columns_and_rows_reversi] = {
00267             {'X', 'X', 'X', 'X', 'X', 'X', ' ', ' '},
00268             {'X', 'X', 'X', 'X', 'X', 'X', ' ', ' '},
00269             {'X', 'O', 'X', 'X', 'X', 'X', ' ', 'X'},
00270             {' ', ' ', 'O', 'X', 'X', 'X', 'X', ' '},
00271             {'X', 'X', 'X', 'O', 'X', 'X', 'X', 'X'},
00272             {' ', ' ', 'X', ' ', 'O', 'X', 'X', 'X'},
00273             {' ', ' ', 'X', ' ', ' ', ' ', ' ', ' '},
00274             {' ', ' ', 'X', ' ', ' ', ' ', ' ', ' '}};
00275         create_game_board_situation(game_board_situation4, reversi_game);
00276         CHECK(reversi_game.check_win(reversi_game.is_there_valid_move_for_player('O'), 'X') == 0);
00277     }
00278 
00279     SUBCASE("O has valid moves, but X doesn't. Game is not over yet "){
00280         char game_board_situation5[num_columns_and_rows_reversi][num_columns_and_rows_reversi]{
00281             {'X', ' ', 'O', 'O', ' ', ' ', 'O', 'O'},
00282             {'X', ' ', 'O', 'O', ' ', 'O', 'O', ' '},
00283             {'X', 'O', 'O', 'O', 'O', 'O', 'O', 'O'},
00284             {'X', 'O', 'O', 'O', 'O', 'O', 'O', 'O'},
00285             {'X', ' ', 'X', 'X', 'O', 'O', 'O', 'O'},
00286             {' ', ' ', ' ', ' ', ' ', 'O', ' ', ' '},
00287             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00288             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '}};
00289         create_game_board_situation(game_board_situation5, reversi_game);
00290         CHECK(reversi_game.check_win(reversi_game.is_there_valid_move_for_player('X'), 'O') == 0);
00291     }
00292 }
00293 
00294 TEST_CASE("Function control_num_pieces_player test"){
00295     Reversi reversi_game;
00296     reversi_game.set_num_pieces_player_O(10);
00297     reversi_game.set_num_pieces_player_X(2);
00298 
00299     reversi_game.control_num_pieces_players(3, 'X');
00300     CHECK(reversi_game.get_num_pieces_player_X() == 5);
00301     CHECK(reversi_game.get_num_pieces_player_O() == 7);
00302 
00303     reversi_game.control_num_pieces_players(5, 'O');
00304     CHECK(reversi_game.get_num_pieces_player_X() == 0);
00305     CHECK(reversi_game.get_num_pieces_player_O() == 12);
00306 
00307     reversi_game.control_num_pieces_players(0, 'O');
00308     CHECK(reversi_game.get_num_pieces_player_X() == 0);
00309     CHECK(reversi_game.get_num_pieces_player_O() == 12);
00310 }
00311 
00312 TEST_CASE("Function flip_pieces test"){
00313     Reversi reversi_game;
00314     char game_board_situation[num_columns_and_rows_reversi][num_columns_and_rows_reversi] =
00315         {
00316             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00317             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00318             {' ', ' ', 'X', 'O', ' ', ' ', ' ', ' '},
00319             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00320             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00321             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00322             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00323             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '}};
00324     create_game_board_situation(game_board_situation, reversi_game);
00325     reversi_game.set_num_pieces_player_O(1);
00326     reversi_game.set_num_pieces_player_X(1);
00327     // Armazena todas as direções em que é possível capturar um oponente
00328     std::list<std::array<int, 2» directions_to_capture_opponents;
00329     directions_to_capture_opponents.push_back({0, 1});
00330 
00331     std::array<int, 2> move_coordinates = {2, 1};
00332 
00333     for (auto direction : directions_to_capture_opponents)
00334         reversi_game.flip_pieces(direction, move_coordinates, 'O');
00335 
00336     // Verifica se a peça do jogador adversário foi tomada
00337     CHECK(reversi_game.get_game_board().get_space(2, 2) == 'O');
00338     // Verifica se as peças flippadas foram adicionadas e subtraídas corretamente
00339     CHECK(reversi_game.get_num_pieces_player_O() == 2);
00340     CHECK(reversi_game.get_num_pieces_player_X() == 0);
00341     // Limpa a lista para iniciar os próximos testes
00342     directions_to_capture_opponents.clear();
00343 
00344     char game_board_situation2[num_columns_and_rows_reversi][num_columns_and_rows_reversi] =
00345         {
00346             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00347             {' ', 'O', ' ', ' ', ' ', ' ', ' ', ' '},
```

```
00348             {' ', ' ', 'X', ' ', ' ', ' ', ' ', ' '},
00349             {' ', ' ', ' ', 'X', ' ', ' ', ' ', ' '},
00350             {' ', ' ', ' ', ' ', 'X', ' ', ' ', ' '},
00351             {' ', ' ', ' ', 'O', 'X', ' ', ' ', ' '},
00352             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00353             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '}};
00354     create_game_board_situation(game_board_situation2, reversi_game);
00355     reversi_game.set_num_pieces_player_O(2);
00356     reversi_game.set_num_pieces_player_X(4);
00357     directions_to_capture_opponents.push_back({-1, -1});
00358     directions_to_capture_opponents.push_back({0, -1});
00359     move_coordinates = {5, 5};
00360     for (auto direction : directions_to_capture_opponents)
00361         reversi_game.flip_pieces(direction, move_coordinates, 'O');
00362 
00363     CHECK(reversi_game.get_game_board().get_space(5, 4) == 'O');
00364     CHECK(reversi_game.get_game_board().get_space(4, 4) == 'O');
00365     CHECK(reversi_game.get_game_board().get_space(3, 3) == 'O');
00366     CHECK(reversi_game.get_game_board().get_space(2, 2) == 'O');
00367 
00368     CHECK(reversi_game.get_num_pieces_player_O() == 6);
00369     CHECK(reversi_game.get_num_pieces_player_X() == 0);
00370     directions_to_capture_opponents.clear();
00371 
00372     char game_board_situation3[num_columns_and_rows_reversi][num_columns_and_rows_reversi] =
00373         {
00374             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00375             {' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X'},
00376             {' ', ' ', ' ', 'X', ' ', ' ', 'O', ' '},
00377             {' ', ' ', ' ', 'O', ' ', 'O', ' ', ' '},
00378             {' ', ' ', ' ', 'O', 'O', ' ', ' ', ' '},
00379             {' ', 'X', 'O', ' ', ' ', ' ', ' ', ' '},
00380             {' ', ' ', ' ', 'O', 'O', ' ', ' ', ' '},
00381             {' ', ' ', ' ', 'X', ' ', ' ', ' ', ' '}};
00382     create_game_board_situation(game_board_situation3, reversi_game);
00383     reversi_game.set_num_pieces_player_X(4);
00384     reversi_game.set_num_pieces_player_O(8);
00385     directions_to_capture_opponents.push_back({-1, 1});
00386     directions_to_capture_opponents.push_back({-1, 0});
00387     directions_to_capture_opponents.push_back({1, 0});
00388     directions_to_capture_opponents.push_back({0, -1});
00389     move_coordinates = {5, 3};
00390     for (auto direction : directions_to_capture_opponents)
00391         reversi_game.flip_pieces(direction, move_coordinates, 'X');
00392 
00393     CHECK(reversi_game.get_game_board().get_space(4, 4) == 'X');
00394     CHECK(reversi_game.get_game_board().get_space(3, 5) == 'X');
00395     CHECK(reversi_game.get_game_board().get_space(2, 6) == 'X');
00396     CHECK(reversi_game.get_game_board().get_space(4, 3) == 'X');
00397     CHECK(reversi_game.get_game_board().get_space(3, 3) == 'X');
00398     CHECK(reversi_game.get_game_board().get_space(6, 3) == 'X');
00399     CHECK(reversi_game.get_game_board().get_space(5, 2) == 'X');
00400     CHECK(reversi_game.get_game_board().get_space(6, 4) == 'O');
00401 
00402     CHECK(reversi_game.get_num_pieces_player_X() == 11);
00403     CHECK(reversi_game.get_num_pieces_player_O() == 1);
00404 }
00405 
00406 TEST_CASE("Function make_move test"){
00407     Reversi reversi_game;
00408     char game_board_situation[num_columns_and_rows_reversi][num_columns_and_rows_reversi] =
00409         {
00410             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00411             {' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X'},
00412             {' ', ' ', ' ', ' ', 'X', ' ', 'O', ' '},
00413             {' ', ' ', ' ', 'O', ' ', 'O', ' ', ' '},
00414             {' ', ' ', ' ', 'O', 'O', ' ', ' ', ' '},
00415             {' ', 'X', 'O', ' ', ' ', ' ', ' ', ' '},
00416             {' ', ' ', ' ', 'O', 'O', ' ', ' ', ' '},
00417             {' ', ' ', ' ', 'X', ' ', ' ', ' ', ' '}};
00418     create_game_board_situation(game_board_situation, reversi_game);
00419     std::list<std::array<int, 2» directions_to_capture_opponents;
00420     std::array<int, 2> move_coordinates;
00421 
00422     directions_to_capture_opponents.push_back({-1, 1});
00423     directions_to_capture_opponents.push_back({-1, 0});
00424     directions_to_capture_opponents.push_back({1, 0});
00425     directions_to_capture_opponents.push_back({0, -1});
00426     move_coordinates = {5, 3};
00427 
00428     reversi_game.set_num_pieces_player_X(4);
00429     reversi_game.set_num_pieces_player_O(8);
00430 
00431     reversi_game.make_move(move_coordinates, 'X', directions_to_capture_opponents);
00432     // Verifica se as peças flippadas e a jogada foram somadas e subtraídas adequadamente
00433     CHECK(reversi_game.get_num_pieces_player_X() == 12);
00434     CHECK(reversi_game.get_num_pieces_player_O() == 1);
```

```
00435     // Verifica se a peça jogada foi posicionada no tabuleiro
00436     CHECK(reversi_game.get_game_board().get_space(5, 3) == 'X');
00437     directions_to_capture_opponents.clear();
00438 
00439     char game_board_situation2[num_columns_and_rows_reversi][num_columns_and_rows_reversi] =
00440         {
00441             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00442             {' ', 'O', ' ', ' ', ' ', ' ', ' ', ' '},
00443             {' ', ' ', 'X', ' ', ' ', ' ', ' ', ' '},
00444             {' ', ' ', ' ', 'X', ' ', ' ', ' ', ' '},
00445             {' ', ' ', ' ', ' ', 'X', ' ', ' ', ' '},
00446             {' ', ' ', ' ', 'O', 'X', ' ', ' ', ' '},
00447             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00448             {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '}};
00449     create_game_board_situation(game_board_situation2, reversi_game);
00450     directions_to_capture_opponents.push_back({-1, -1});
00451     directions_to_capture_opponents.push_back({0, -1});
00452     move_coordinates = {5, 5};
00453 
00454     reversi_game.set_num_pieces_player_O(2);
00455     reversi_game.set_num_pieces_player_X(4);
00456 
00457     reversi_game.make_move(move_coordinates, 'O', directions_to_capture_opponents);
00458 
00459     CHECK(reversi_game.get_num_pieces_player_O() == 7);
00460     CHECK(reversi_game.get_num_pieces_player_X() == 0);
00461     CHECK(reversi_game.get_game_board().get_space(5, 5) == 'O');
00462 }
00463 
00464 TEST_CASE("Function find_all_directions_that_make_move test"){
00465     Reversi reversi_game;
00466 
00467     char game_board_situation1[num_columns_and_rows_reversi][num_columns_and_rows_reversi]{
00468         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00469         {' ', ' ', '*', '*', 'X', 'O', '*', ' '},
00470         {' ', 'O', 'X', '*', 'X', 'X', 'O', ' '},
00471         {' ', ' ', 'O', 'X', 'X', 'X', 'X', '*'},
00472         {' ', '*', 'X', 'O', 'X', '*', ' ', ' '},
00473         {' ', '*', 'O', ' ', ' ', ' ', ' ', ' '},
00474         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00475         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '}};
00476     create_game_board_situation(game_board_situation1, reversi_game);
00477 
00478     SUBCASE("Multiple directions"){
00479         std::list<std::array<int, 2» directions_to_capture_opponents_function;
00480         std::array<int, 2> move = {2, 3}, direction1 = {1, 0}, direction2 = {0, 1}, direction3 = {0,
    -1};
00481         std::list<std::array<int, 2» directions_to_capture_opponents_test = {direction1, direction2,
    direction3};
00482         reversi_game.find_all_directions_to_make_move(move, 'O',
    directions_to_capture_opponents_function);
00483 
00484         CHECK(directions_to_capture_opponents_function == directions_to_capture_opponents_test);
00485 
00486         move = {4, 5};
00487         directions_to_capture_opponents_function.clear();
00488         directions_to_capture_opponents_test.clear();
00489         direction1 = {0, -1};
00490         direction2 = {-1, 0};
00491         directions_to_capture_opponents_test = {direction1, direction2};
00492         reversi_game.find_all_directions_to_make_move(move, 'O',
    directions_to_capture_opponents_function);
00493         CHECK(directions_to_capture_opponents_function == directions_to_capture_opponents_test);
00494     }
00495 
00496     SUBCASE("One direction"){
00497         std::array<int, 2> move = {2, 7}, direction = {0, -1};
00498         std::list<std::array<int, 2» directions_to_capture_opponents_test = {direction};
00499         std::list<std::array<int, 2» directions_to_capture_opponents_function;
00500         reversi_game.find_all_directions_to_make_move(move, 'X',
    directions_to_capture_opponents_function);
00501         CHECK(directions_to_capture_opponents_function == directions_to_capture_opponents_test);
00502     }
00503     SUBCASE("No direction"){
00504         std::list<std::array<int, 2» directions_to_capture_opponents_function;
00505         std::array<int, 2> move = {0, 4};
00506         reversi_game.find_all_directions_to_make_move(move, 'X',
    directions_to_capture_opponents_function);
00507         CHECK(directions_to_capture_opponents_function.empty() == 1);
00508     }
00509 }
00510 
00511 TEST_CASE("Function process_move"){
00512     Reversi reversi_game;
00513 
00514     char game_board_situation1[num_columns_and_rows_reversi][num_columns_and_rows_reversi]{
00515         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
```

```
00516         {' ', ' ', '*', '*', 'X', 'O', '*', ' '},
00517         {' ', 'O', 'X', '*', 'X', 'X', 'O', ' '},
00518         {' ', ' ', 'O', 'X', 'X', 'X', 'X', '*'},
00519         {' ', '*', 'X', 'O', 'X', '*', ' ', ' '},
00520         {' ', '*', 'O', ' ', ' ', ' ', ' ', ' '},
00521         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
00522         {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '}
00523 
00524     };
00525     create_game_board_situation(game_board_situation1, reversi_game);
00526     std::array<int, 2> move = {2, 3};
00527     SUBCASE("Valid moves"){
00528         CHECK(reversi_game.process_move(move, 'O') == 1);
00529         move = {6, 4};
00530         CHECK(reversi_game.process_move(move, 'X') == 1);
00531     }
00532     SUBCASE("Invalid moves"){
00533         move = {8, 8};
00534         CHECK(reversi_game.process_move(move, 'X') == 0);
00535         move = {2, 2};
00536         CHECK(reversi_game.process_move(move, 'O') == 0);
00537     }
00538 }
```

5.35 Referência do Arquivo tests/TicTacToeClass_test.cpp

```
#include "doctest.h"
#include "Player.hpp"
#include "Tic_tac_toe.hpp"
```

Gráfico de dependência de inclusões para TicTacToeClass_test.cpp:

Definições e Macros

- #define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN

Funções

- void create_game_board_situation (char game_board_situation[num_rows_received][num_columns_received], Tic_tac_toe &tic_tac_toe_game)

  Aloca um tabuleiro com situações de teste no Tic Tac Toe sem a necessidade de uma partida real.
- TEST_CASE ("Function is_valid_move test")
- TEST_CASE ("Function make_move test")
- TEST_CASE ("Function check_tie test")
- TEST_CASE ("Function check_tic_tac_toe_win test")

Variáveis

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

5.35.1 Definições e macros

5.35.1.1 DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN

```
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
```

Definição na linha 1 do arquivo TicTacToeClass_test.cpp.

5.35.2 Funções

5.35.2.1 create_game_board_situation()

```
void create_game_board_situation (
            char game_board_situation[num_rows_received][num_columns_received],
            Tic_tac_toe & tic_tac_toe_game )
```

Aloca um tabuleiro com situações de teste no Tic Tac Toe sem a necessidade de uma partida real.

Parâmetros

game_board_situation	Tabuleiro que simula situações de jogo
reversi_game	Instância da classe Tic_tac_toe

Definição na linha 15 do arquivo TicTacToeClass_test.cpp.

5.35.2.2 TEST_CASE() [1/4]

```
TEST_CASE (
            "Function check_tic_tac_toe_win test"  )
```

Definição na linha 122 do arquivo TicTacToeClass_test.cpp.

5.35.2.3 TEST_CASE() [2/4]

```
TEST_CASE (
            "Function check_tie test"  )
```

Definição na linha 91 do arquivo TicTacToeClass_test.cpp.

5.35.2.4 TEST_CASE() [3/4]

```
TEST_CASE (
            "Function is_valid_move test"  )
```

Definição na linha 24 do arquivo TicTacToeClass_test.cpp.

5.35.2.5 TEST_CASE() [4/4]

```
TEST_CASE (
            "Function make_move test"  )
```

Definição na linha 59 do arquivo TicTacToeClass_test.cpp.

5.35.3 Variáveis

5.35.3.1 num_columns_received

```
const int num_columns_received = 3
```

Definição na linha 7 do arquivo TicTacToeClass_test.cpp.

5.35.3.2 num_rows_received

```
const int num_rows_received = 3
```

Definição na linha 6 do arquivo TicTacToeClass_test.cpp.

5.36 TicTacToeClass_test.cpp

Ir para a documentação desse arquivo.

```
00001 #define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
00002 #include "doctest.h"
00003 #include "Player.hpp"
00004 #include "Tic_tac_toe.hpp"
00005
00006 const int num_rows_received = 3;
00007 const int num_columns_received = 3;
00008
00015 void create_game_board_situation(char game_board_situation[num_rows_received][num_columns_received],
      Tic_tac_toe &tic_tac_toe_game){
00016     char *board[num_columns_received];
00017     for (int i = 0; i < num_columns_received; i++){
00018         board[i] = game_board_situation[i];
00019     }
00020
00021     tic_tac_toe_game.get_game_board().set_game_board(board);
00022 }
00023
00024 TEST_CASE("Function is_valid_move test")
00025 {
00026     Tic_tac_toe jogo;
00027
00028     char game_board_situation[num_rows_received][num_columns_received] = {
00029         {'X', ' ', ' '},
00030         {' ', 'O', ' '},
00031         {' ', ' ', ' '}
00032     };
00033
00034     create_game_board_situation(game_board_situation, jogo);
00035
00036     SUBCASE("Move inside board and position free") {
00037         int x = 1, y = 1;
00038         CHECK(jogo.is_valid_move(x, y) == true);
00039     }
00040
00041     SUBCASE("Move out of the board") {
00042         int x = 4, y = 4;
00043         CHECK(jogo.is_valid_move(x, y) == false);
00044
00045         x = INT_MAX + 1, y = INT_MAX + 1;
00046         CHECK(jogo.is_valid_move(x, y) == false);
00047
00048         x = INT_MIN - 1, y = INT_MIN - 1;
00049         CHECK(jogo.is_valid_move(x, y) == false);
00050     }
00051
00052     SUBCASE("Move on a ocuppied position") {
00053         jogo.make_move(1, 1);
00054         int x = 1, y = 1;
00055         CHECK(jogo.is_valid_move(x, y) == false);
00056     }
00057 }
00058
00059 TEST_CASE("Function make_move test")
```

```
00060 {
00061     Tic_tac_toe jogo;
00062
00063     char game_board_situation[num_rows_received][num_columns_received] = {
00064         {'X', ' ', ' '},
00065         {' ', 'O', ' '},
00066         {' ', ' ', ' '}
00067     };
00068
00069     create_game_board_situation(game_board_situation, jogo);
00070
00071     SUBCASE("Valid move switches the player and update the board")
00072     {
00073         int x = 2, y = 2;
00074         jogo.make_move(x, y);
00075         CHECK(jogo.get_game_board().get_space(x - 1, y - 1) == 'X');
00076         CHECK(jogo.get_current_player() == 'O');
00077         int x = 3, y = 3;
00078         CHECK(jogo.get_game_board().get_space(x, y) == 'O');
00079         CHECK(jogo.get_current_player() == 'X');
00080     }
00081
00082     SUBCASE("Invalid move switch the player and does not update the board")
00083     {
00084         int x = 2, y = 2;
00085         jogo.make_move(x, y);
00086         CHECK(jogo.get_game_board().get_space(x - 1, y - 1) == 'O');
00087         CHECK(jogo.get_current_player() == 'O');
00088     }
00089 }
00090
00091 TEST_CASE("Function check_tie test")
00092 {
00093     Tic_tac_toe jogo;
00094
00095     SUBCASE("Game is a tie (board full, no winner)")
00096     {
00097         char tie_board[num_rows_received][num_columns_received] =
00098         {
00099             {'X', 'O', 'X'},
00100             {'X', 'X', 'O'},
00101             {'O', 'X', 'O'}
00102         };
00103
00104         create_game_board_situation(tie_board, jogo);
00105         CHECK(jogo.check_tie() == true);
00106     }
00107
00108     SUBCASE("Game is not a tie (there is a winner)")
00109     {
00110         char winner_board[num_rows_received][num_columns_received] =
00111         {
00112             {'X', 'X', 'X'},
00113             {'O', 'O', ' '},
00114             {' ', ' ', ' '}
00115         };
00116
00117         create_game_board_situation(winner_board, jogo);
00118         CHECK(jogo.check_tie() == false);
00119     }
00120 }
00121
00122 TEST_CASE("Function check_tic_tac_toe_win test")
00123 {
00124     Tic_tac_toe jogo;
00125
00126     SUBCASE("Win by row")
00127     {
00128         char row_win_board[num_rows_received][num_columns_received] =
00129         {
00130             {'X', 'X', 'X'},
00131             {'O', 'O', ' '},
00132             {' ', ' ', ' '}
00133         };
00134
00135         create_game_board_situation(row_win_board, jogo);
00136         CHECK(jogo.check_tic_tac_toe_win() == 'X');
00137     }
00138
00139     SUBCASE("Win by column")
00140     {
00141         char column_win_board[num_rows_received][num_columns_received] =
00142         {
00143             {'X', 'O', ' '},
00144             {'X', 'O', ' '},
00145             {'X', ' ', ' '}
00146         };
```

```
00147
00148         create_game_board_situation(column_win_board, jogo);
00149         CHECK(jogo.check_tic_tac_toe_win() == 'X');
00150     }
00151
00152     SUBCASE("Win by main diagonal")
00153     {
00154         char main_diag_win_board[num_rows_received][num_columns_received] =
00155         {
00156             {'O', 'X', 'X'},
00157             {'X', 'O', ' '},
00158             {' ', ' ', 'O'}
00159         };
00160
00161         create_game_board_situation(main_diag_win_board, jogo);
00162         CHECK(jogo.check_tic_tac_toe_win() == 'O');
00163     }
00164
00165     SUBCASE("Win by secondary diagonal")
00166     {
00167         char sec_diag_win_board[num_rows_received][num_columns_received] =
00168         {
00169             {'X', 'O', 'O'},
00170             {' ', 'O', ' '},
00171             {'O', ' ', 'X'}
00172         };
00173
00174         create_game_board_situation(sec_diag_win_board, jogo);
00175         CHECK(jogo.check_tic_tac_toe_win() == 'O');
00176     }
00177
00178     SUBCASE("No winner (game in progress)")
00179     {
00180         char no_winner_board[num_rows_received][num_columns_received] =
00181         {
00182             {'X', 'O', ' '},
00183             {'X', 'O', ' '},
00184             {'O', ' ', ' '}
00185         };
00186
00187         create_game_board_situation(no_winner_board, jogo);
00188         CHECK(jogo.check_tic_tac_toe_win() == 'F');
00189     }
00190
00191     SUBCASE("No winner (tie)")
00192     {
00193         char tie_board[num_rows_received][num_columns_received] =
00194         {
00195             {'X', 'O', 'X'},
00196             {'X', 'X', 'O'},
00197             {'O', 'X', 'O'}
00198         };
00199
00200         create_game_board_situation(tie_board, jogo);
00201         CHECK(jogo.check_tic_tac_toe_win() == 'F');
00202     }
00203 }
00204
00205
00206
00207
00208
00209
00210
00211
```

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