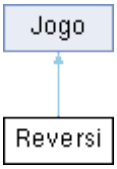
Referência da Classe Reversi

Classe que representa o jogo **Reversi**, derivada de **Jogo**. Mais...

`#include <Reversi.hpp>`

Diagrama de hierarquia da classe Reversi:



Membros Públicos

void	**inicializarTabuleiro** () override Inicializa o tabuleiro do jogo **Reversi**, colocando as peças iniciais.
	Reversi (**Jogador** *_jogador1, **Jogador** *_jogador2) Construtor da classe **Reversi**.
bool	**validarJogada** (int x, int y, **Jogador** *jogador) override Valida se uma jogada é possível de acordo com as regras do **Reversi**.
virtual int	**verificarCondicaoVitoria** () Verifica a condição de vitória.
void	**imprimirTabuleiro** () Imprime o estado atual do tabuleiro no terminal.
void	**realizarJogada** () override Realiza uma jogada no tabuleiro.
void	**capturarPecas** (int x, int y, **Jogador** *jogador) Captura as peças do oponente em todas as direções possíveis (horizontal, vertical e diagonal).
void	**capturarDirecao** (int x, int y, **Jogador** *jogador, int deltaX, int deltaY) Captura as peças do oponente em uma direção específica.

▸ **Membros Públicos herdados de Jogo**

Descrição detalhada

Classe que representa o jogo **Reversi**, derivada de **Jogo**.

A classe **Reversi** contém a lógica específica para o jogo **Reversi**, incluindo métodos para verificar a condição de vitória, realizar jogadas e inicializar o tabuleiro. A classe interage com a classe **Jogador** para rastrear as peças de cada jogador e determinar o vencedor.

Construtores e Destrutores

◆ Reversi()

Reversi::Reversi (**Jogador** * _jogador1,
Jogador * _jogador2)

Construtor da classe **Reversi**.

Parâmetros

_jogador1 Ponteiro para o primeiro jogador.

_jogador2 Ponteiro para o segundo jogador.

```
70     : jogador1(_jogador1), jogador2(_jogador2), vezJogador1(true) {}  // Inicializa vezJogador1 como true
```

Documentação das funções

◆ capturarDirecao()

```
void Reversi::capturarDirecao ( int      x,
                                int      y,
                                Jogador * jogador,
                                int      deltaX,
                                int      deltaY )
```

Captura as peças do oponente em uma direção específica.

Este método verifica se há peças do oponente em uma sequência em uma direção definida pelos deltas (deltaX, deltaY). Se a sequência termina com uma peça do jogador atual, todas as peças do oponente nessa sequência são capturadas.

Parâmetros

x	A coordenada x inicial.
y	A coordenada y inicial.
jogador	O jogador que realizou a jogada.
deltaX	O incremento para a coordenada x na direção.
deltaY	O incremento para a coordenada y na direção.

```
119 
    {
120     int i = x + deltaX;
121     int j = y + deltaY;
122     std::vector<std::pair<int, int>> pecasParaVirar;
123 
124     // verifica a sequência de peças
125     while (tabuleiro.posicaoValida(i, j) && tabuleiro.obterPeca(i, j) ==
    (jogador == jogador1 ? jogador2->minhaPeca() : jogador1->minhaPeca())) {
126         pecasParaVirar.push_back({i, j});
127         i += deltaX;
128         j += deltaY;
129     }
130 
131     // verifica se há uma peça do jogador atual no final da sequência
132     if (tabuleiro.posicaoValida(i, j) && tabuleiro.obterPeca(i, j) ==
    jogador->minhaPeca()) {
133         for (const auto& pos : pecasParaVirar) {
134             tabuleiro.definirPosicao(pos.first, pos.second, jogador-
    >minhaPeca());
135         }
136     }
137 }
```

◆ capturarPecas()

void Reversi::capturarPecas (int x,
int y,
Jogador * jogador)

Captura as peças do oponente em todas as direções possíveis (horizontal, vertical e diagonal).

Este método é chamado após uma peça ser colocada no tabuleiro. Ele verifica em todas as direções a partir da posição (x, y) se existem peças do oponente que podem ser capturadas, ou seja, se estão entre a peça recém-colocada e outra peça do jogador atual.

Parâmetros

x	A coordenada x da peça que foi colocada.
y	A coordenada y da peça que foi colocada.
jogador	Ponteiro para o jogador que realizou a jogada.

```
184                                                     {
185     // chama o método para capturar peças em todas as direções
186     capturarDirecao(x, y, jogador, 0, 1);  // direita
187     capturarDirecao(x, y, jogador, 0, -1); // esquerda
188     capturarDirecao(x, y, jogador, 1, 0);  // baixo
189     capturarDirecao(x, y, jogador, -1, 0); // cima
190     capturarDirecao(x, y, jogador, -1, -1); // diagonal superior esquerda
191     capturarDirecao(x, y, jogador, 1, 1);   // diagonal inferior direita
192     capturarDirecao(x, y, jogador, -1, 1);  // diagonal superior direita
193     capturarDirecao(x, y, jogador, 1, -1);  // diagonal inferior esquerda
194 }
```

◆ imprimirTabuleiro()

void Reversi::imprimirTabuleiro ( )

Imprime o estado atual do tabuleiro no terminal.

```
7                                   {
8     tabuleiro.imprimir();
9 }
```

◆ inicializarTabuleiro()

void Reversi::inicializarTabuleiro ( ) override virtual

Inicializa o tabuleiro do jogo **Reversi**, colocando as peças iniciais.

Reimplementa **Jogo**.

```
11                                      {
12      tabuleiro.configurarTabuleiro(8, 8); // configura pra ter 8 linhas e 8
    colunas
13      tabuleiro.definirPosicao(3, 3, 'B');
14      tabuleiro.definirPosicao(4, 4, 'B');
15      tabuleiro.definirPosicao(3, 4, 'W');
16      tabuleiro.definirPosicao(4, 3, 'W');
17  }
```

◆ realizarJogada()

void Reversi::realizarJogada ( ) override virtual

Realiza uma jogada no tabuleiro.

Implementa **Jogo**.

```
139                                 {
140     Jogador* jogadorAtual = (vezJogador1) ? jogador1 : jogador2; // alterna entre jogador1 e jogador2
141     int x, y;
142 
143     while (true) {
144         std::cout << jogadorAtual->getNome() << ", digite a linha a ser jogada (0-7): ";
145         std::cin >> x;
146 
147         if (std::cin.fail() || x < 0 || x > 7) {
148             std::cin.clear(); // limpa o estado de erro
149             std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'); // descarta a entrada inválida
150             std::cout << "Entrada inválida. Por favor, digite um número entre 0 e 7." << std::endl;
151             continue;
152         }
153 
154         std::cout << jogadorAtual->getNome() << ", digite a coluna a ser jogada (0-7): ";
155         std::cin >> y;
156 
157         if (std::cin.fail() || y < 0 || y > 7) {
158             std::cin.clear(); // limpa o estado de erro
159             std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'); // descarta a entrada inválida
160             std::cout << "Entrada inválida. Por favor, digite um número entre 0 e 7." << std::endl;
161             continue;
162         }
163 
164         // se ambas as entradas forem válidas, sai do loop
165         break;
166     }
167 
168     // verifica se a jogada é válida
169     if (!validarJogada(x, y, jogadorAtual)) {
170         std::cout << "Jogada inválida. Tente novamente." << std::endl;
171         return;
172     }
173 
174     // coloca a peça do jogador no tabuleiro
175     tabuleiro.definirPosicao(x, y, jogadorAtual->minhaPeca());
176 
177     // lógica para capturar as peças do oponente
178     capturarPecas(x, y, jogadorAtual);
179 
180     // alterna a vez do jogador
181     vezJogador1 = !vezJogador1;
182 }
```

◆ validarJogada()

bool Reversi::validarJogada (int x,
int y,
Jogador * jogador) override virtual

Valida se uma jogada é possível de acordo com as regras do **Reversi**.

Parâmetros

x	Posição X no tabuleiro.
y	Posição Y no tabuleiro.
jogador	Ponteiro para o jogador que está fazendo a jogada.

Retorna

true se a jogada for válida, false caso contrário.

Implementa **Jogo**.

```
19                                                      {
20      // verifica se a posição está dentro do tabuleiro e está vazia
21      if (!tabuleiro.posicaoValida(x, y) || tabuleiro.obterPeca(x, y) != '.')
    {
22          return false;
23      }
24
25      // direções: direita, esquerda, baixo, cima, diagonais
26      int direcoes[8][2] = {
27          {0, 1}, {0, -1}, {1, 0}, {-1, 0},
28          {1, 1}, {-1, -1}, {1, -1}, {-1, 1}
29      };
30
31      // determina as peças do jogador e do oponente
32      char pecaJogador = jogador->minhaPeca();  // 'W' ou 'B'
33      char pecaOponente = (pecaJogador == 'B') ? 'W' : 'B';
34
35      // verifica cada direção
36      for (int i = 0; i < 8; i++) {
37          int dx = direcoes[i][0];
38          int dy = direcoes[i][1];
39          int nx = x + dx;
40          int ny = y + dy;
41
42          bool encontrouOponente = false;
43
44          // avança na direção até encontrar uma peça do jogador ou sair do
    tabuleiro
45          while (tabuleiro.posicaoValida(nx, ny)) {
46              char pecaAtual = tabuleiro.obterPeca(nx, ny);
47              if (pecaAtual == pecaOponente) {
48                  encontrouOponente = true;
49              }
50              else if (pecaAtual == pecaJogador) {
51                  // a jogada é válida se encontrar uma peça do jogador após
    peças do oponente
52                  if (encontrouOponente) {
53                      return true;
54                  }
55                  break;
56              }
57              else {
58                  break; // encontra uma casa vazia ou fim do tabuleiro
```

```
59                  }
60                  nx += dx;
61                  ny += dy;
62              }
63          }
64
65          // se nenhuma direção for válida, a jogada não é válida
66          return false;
67      }
```

- verificarCondicaoVitoria()

int Reversi::verificarCondicaoVitoria ( ) virtual

Verifica a condição de vitória.

Analisa o tabuleiro para determinar se algum jogador cumpriu as condições de vitória.

Retorna

1 se um jogador venceu, -1 em caso de empate, 0 se o jogo continua.

Reimplementa **Jogo**.

```
72                                          {
73      bool temMovimentoParaJogador1 = false;
74      bool temMovimentoParaJogador2 = false;
75
76      // Itera por todas as posições do tabuleiro
77      for (int x = 0; x < 8; x++) {
78          for (int y = 0; y < 8; y++) {
79              if (tabuleiro.obterPeca(x, y) == '.') {
80                  if (validarJogada(x, y, jogador1)) {
81                      temMovimentoParaJogador1 = true;
82                  }
83                  if (validarJogada(x, y, jogador2)) {
84                      temMovimentoParaJogador2 = true;
85                  }
86              }
87          }
88      }
89
90      // Se nenhum jogador tem movimento válido, conta as peças no tabuleiro
91      if (!temMovimentoParaJogador1 && !temMovimentoParaJogador2) {
92          int contagemJogador1 = 0;
93          int contagemJogador2 = 0;
94
95          for (int x = 0; x < 8; x++) {
96              for (int y = 0; y < 8; y++) {
97                  char peca = tabuleiro.obterPeca(x, y);
98                  if (peca == jogador1->minhaPeca()) {
99                      contagemJogador1++;
100                 } else if (peca == jogador2->minhaPeca()) {
101                     contagemJogador2++;
102                 }
103             }
104         }
105
106         // Determina o resultado do jogo
107         if (contagemJogador1 > contagemJogador2) {
108             return 1; // Jogador 1 vence
109         } else if (contagemJogador2 > contagemJogador1) {
110             return 1; // Jogador 2 vence
111         } else {
112             return -1; // Empate
113         }
114     }
115
116     return 0; // O jogo continua
117 }
```

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

- /Users/iangodoi/Desktop/TP-jogosTabuleiro-cpp-desenvolvimento/include/**Reversi.hpp**
- /Users/iangodoi/Desktop/TP-jogosTabuleiro-cpp-desenvolvimento/src/**Reversi.cpp**
