Referência da Classe Game

`#include <Game.hpp>`

Diagrama de hierarquia da classe Game:

Membros Públicos

void	**GameStart** (std::string nomedojogo, std::string nomedouser) inicia uma nova partida
bool	**exist** (std::string name) carrega um jogo salvo
void	**LoadGame** (std::string jogo)
std::string	**WhoTurn** ()
unsigned	**GetTurn** ()
void	**IncreaseTurn** ()
void	**SavePlay** (int x, int y, int xp, int yp, std::string nomea, std::string nomedouser) salvar partidas
bool	**Cavalo** (int x, int y, int xp, int yp)
bool	**Torre** (int x, int y, int xp, int yp)
bool	**PeaoP** (int x, int y, int xp, int yp)
bool	**PeaoB** (int x, int y, int xp, int yp)
bool	**King** (int x, int y, int xp, int yp)
bool	**Bispo** (int x, int y, int xp, int yp)
bool	**Queen** (int x, int y, int xp, int yp)
bool	**IsValid** (int x, int y, int xp, int yp)
void	**Move** (int x, int y, int xp, int yp)

- Membros Públicos herdados de Login

Outros membros herdados

- Atributos Protegidos herdados de Login

Documentação das funções

◆ Bispo()

bool Game::Bispo (int x,
int y,
int xp,
int yp
)

```
153                                                 {
154     int difx = xp - x;
155     int dify = yp - y;
156     if(difx != dify && difx != -1*dify){
157         std::cout << "Movimento invalido" << std::endl;
158         std::cout << "[DICA] O bispo apenas se move nas diagonais." << std::endl;
159         return 0;
160     }
161     //diagonal inferior esquerda
162     if(difx > 0 && dify > 0){
163         for(int i = 1; i < xp - x; i++){
164             if(Tab.pieces[xp - i][yp - i].tipo != '-'){
165                 std::cout << "Movimento invalido" << std::endl;
166                 std::cout << "[DICA] O bispo nao eh capaz de pular pecas." <<
    std::endl;
167                 return 0;
168             }
169         }
170         return 1;
171     }
172     if(difx > 0 && dify < 0){
173         for(int i = 1; i < xp - x; i++){
174             if(Tab.pieces[xp - i][y - i].tipo != '-'){
175                 std::cout << "Movimento invalido" << std::endl;
176                 std::cout << "[DICA] O bispo nao eh capaz de pular pecas." <<
    std::endl;
177                 return 0;
178             }
179         }
180         return 1;
181     }
182     if(difx < 0 && dify > 0){
183         for(int i = 1; i < x - xp; i++){
184             if(Tab.pieces[x - i][yp - i].tipo != '-'){
185                 std::cout << "Movimento invalido" << std::endl;
186                 std::cout << "[DICA] O bispo nao eh capaz de pular pecas." <<
    std::endl;
187                 return 0;
188             }
189         }
190         return 1;
191     }
192     if(difx < 0 && dify < 0){
193         for(int i = 1; i < x - xp; i++){
194             if(Tab.pieces[x - i][y - i].tipo != '-'){
195                 std::cout << "Movimento invalido" << std::endl;
196                 std::cout << "[DICA] O bispo nao eh capaz de pular pecas." <<
    std::endl;
197                 return 0;
198             }
199         }
200         return 1;
201     }
202     std::cout << "nem sei oq aconteceu";
203     return 0;
204 }
```

◆ Cavalo()

bool Game::Cavalo (int x,
int y,
int xp,
int yp
)

```
135                                                 {
136     if((xp == x + 1 && yp == y + 2) || (xp == x - 1 && yp == y + 2)){
137         return 1;
138     }
139     if((xp == x + 1 && yp == y - 2) || (xp == x - 1 && yp == y - 2)){
140         return 1;
141     }
142     if((xp == x + 2 && yp == y + 1) ||(xp == x - 2 && yp == y + 1)){
143         return 1;
144     }
145     if((xp == x + 2 && yp == y - 1) || (xp == x - 2 && yp == y - 1)){
146         return 1;
147     }
148     std::cout << "Movimento invalido" << std::endl;
149     std::cout << "[DICA] O cavalo se move apenas em L." << std::endl;
150     return 0;
151 }
```

◆ exist()

bool Game::exist (std::string name)

carrega um jogo salvo

Parâmetros

verifica se a partida escolhda para carregar existe

```
421                                 {
422     std::fstream filetest;
423     filetest.open(name, std::ios::in);
424     if(filetest){
425         return 1;
426     }
427     return 0;
428 }
```

◆ GameStart()

void Game::GameStart (std::string nomedojogo,
std::string nomedouser
)

inicia uma nova partida

```
510                                                         {
511     //esta função é responsável por executar o jogo, de maneira padrão
512     char x, xp;
513     char y, yp;
514     turn = 0;
515     std::cout << "Bem vindo ao jogo de xadrez" << std::endl
516      << "Pecas brancas em maiusculo e pretas em minusculo" << std::endl
517      << "Para parar o jogo, digite apenas S" << std::endl;
518     Tab.Print();
519     std::cout << "Turno " << GetTurn() << ", vez das "<< WhoTurn() << std::endl;
520     std::cout << "Digite as coordenadas: ";
521     while(game == 1){
522         std::cin >> x;
523         if(x == 'S' || x == 's'){
524             game = 0;
525             break;
526         }
527         std::cin >> y >> xp >> yp;
528         #ifdef _WIN32
529             std::system("cls");
530         #else
531             std::system("clear");
532         #endif
533         //caso a jogada seja inválida, a função responsável avisará o motivo e nada
    ocorrerá no tabuleiro
534         if(IsValid(getchar(y), getchar(x), getchar(yp), getchar(xp))){
535             //caso seja válida, salva a jogada e executa
536             SavePlay(getchar(y), getchar(x), getchar(yp), getchar(xp), nomea,
    nomedouser);
537             Move(getchar(y), getchar(x), getchar(yp), getchar(xp));
538             IncreaseTurn();
539         }
540         Tab.Print();
541         if(game == 0){
542             break;
543         }
544         std::cout << "Turno " << GetTurn() << ", vez das "<< WhoTurn() << std::endl;
545         std::cout << "Digite as coordenadas: ";
546     }
547     std::cout << "O jogo terminou, pressione uma tecla para continuar: ";
548     std::string a;
549     game = 1;
550     Tab.Reset();
551     std::cin >> a;
552 }
```

◆ GetTurn()

unsigned Game::GetTurn ( )

Retorna

o jogador que deverá jogar

```
15                          {
16     return 1 + (turn/2);
17 }
```

IncreaseTurn()

void Game::IncreaseTurn ( )

Retorna

incrementa a quantidade de turnos jogados

```
12                          {
13     turn++;
14 }
```

IsValid()

```
bool Game::IsValid ( int  x,
                     int  y,
                     int  xp,
                     int  yp
                    )
```

```
315                                                 {
316     //regras globais para todas as peças do jogo
317     if(turn%2 == 0){
318         if(Tab.pieces[x][y].cor == 'B'){
319             std::cout << "Turno das brancas" << std::endl;
320             return 0;
321         }
322     }
323     if(turn%2 == 1){
324         if(Tab.pieces[x][y].cor == 'P'){
325             std::cout << "Turno das brancas" << std::endl;
326             return 0;
327         }
328     }
329     if(Tab.pieces[x][y].tipo == '-'){
330         std::cout << "Peca inexistente" << std::endl;
331         return 0;
332     }
333     if(xp > 8 || yp > 8){
334         std::cout << "Posicao alvo inexistente" << std::endl;
335         return 0;
336     }
337     if(x == xp && y == yp){
338         std::cout << "Movimento invalido" << std::endl;
339         return 0;
340     }
341     if(Tab.pieces[x][y].cor == Tab.pieces[xp][yp].cor){
342         std::cout << "Movimento invalido" << std::endl;
343         std::cout << "[DICA] Tente nao atacar seus aliados" << std::endl;
344         return 0;
345     }
346     //regras específicas de peças
347     switch (Tab.pieces[x][y].tipo){
348         case 'p':
349             return PeaoB(x, y, xp, yp);
350             break;
351         case 'P':
352             return PeaoP(x, y, xp, yp);
353             break;
354         case 'T':
355             return Torre(x, y, xp, yp);
356             break;
357         case 't':
358             return Torre(x, y, xp, yp);
359             break;
360         case 'C':
361             return Cavalo(x, y, xp, yp);
362             break;
363         case 'c':
364             return Cavalo(x, y, xp, yp);
365             break;
366         case 'K':
367             return King(x, y, xp, yp);
368             break;
369         case 'k':
370             return King(x, y, xp, yp);
371             break;
```

```
372         case 'B':
373             return Bispo(x, y, xp, yp);
374             break;
375         case 'b':
376             return Bispo(x, y, xp, yp);
377             break;
378         case 'Q':
379             return Queen(x, y, xp, yp);
380             break;
381         case 'q':
382             return Queen(x, y, xp, yp);
383             break;
384     }
385     //caso passe desse ponto, significa que não se trata de uma peça implementada de xadrez e algo deu errado
386     std::cout << "Peca desconhecida" << std::endl;
387     return 0;
388 }
```

◆ King()

bool Game::King (int x,
int y,
int xp,
int yp
)

```
206                                             {
207     bool cond1 = 0;
208     if(xp == x + 1 || xp == x - 1 || xp == x){
209         cond1 = 1;
210     }
211     bool cond2 = 0;
212     if(yp == y + 1 || yp == y - 1 || yp == y){
213         cond2 = 1;
214     }
215     if(cond1 && cond2){
216         return 1;
217     }
218     std::cout << "Movimento invalido" << std::endl;
219     std::cout << "[DICA] O rei se move apenas 1 casa em todas as direcoes." << std::endl;
220     return 0;
221 }
```

◆ LoadGame()

void Game::LoadGame (std::string jogo)

Retorna

carrega o jogo escolhido existente

```
554                                     {
555     std::fstream f;
556     f.open(jogo, std::ios::in);
557     int x, y, xp, yp;
558     Hist temp;
559     while(f >> x >> y >> xp >> yp){
560         Move(x, y, xp, yp);
561         IncreaseTurn();
562         temp.x = x;
563         temp.y = y;
564         temp.xp = xp;
565         temp.yp = yp;
566         History.push_back(temp);
567     }
568 //esta função está relacionada com o banco de dados,
569 //onde é possível capturar as jogadas realizadas e continuar de onde tenha parado
570 //Nota que não é preciso validar jogadas, uma vez que apenas jogadas válidas são
    salvas
571 //Para continuar o progresso, basta usar a função antes de iniciar com GameStart()
572 }
```

◆ Move()

void Game::Move (int x,
int y,
int xp,
int yp
)

```
390                                                 {
391     //como já há uma função para validar movimentos, esta apenas executa a ação
392     Tab.pieces[xp][yp].tipo = Tab.pieces[x][y].tipo;
393     Tab.pieces[xp][yp].cor = Tab.pieces[x][y].cor;
394     Tab.pieces[x][y].tipo = '-';
395     Tab.pieces[x][y].cor = '-';
396     //checando o Xeque-Mate
397     bool checB = 0;
398     bool checP = 0;
399     for(int i = 0; i < 8; i++){
400         for(int j = 0; j < 8; j++){
401             if(Tab.pieces[i][j].tipo == 'k'){
402                 checB = 1;
403             }
404             if(Tab.pieces[i][j].tipo == 'K'){
405                 checP = 1;
406             }
407         }
408     }
409     //Anunciando a vitória caso haja xeque
410     if(!checB){
411         std::cout << "Pretos ganharam!" << std::endl;
412         game = 0;
413     }
414     if(!checP){
415         std::cout << "Brancos ganharam!" << std::endl;
416         game = 0;
417     }
418 
419 }
```

◆ PeaoB()

bool Game::PeaoB (int x,
int y,
int xp,
int yp
)

```
31                                                  {
32      bool frente = 0;
33      if(turn == 1) {
34          if(xp == x + 2){
35              frente = 1;
36          }
37      }
38      if(xp == x + 1){
39          frente = 1;
40      }
41      bool cond1 = 0;
42      if(y == yp && Tab.pieces[xp][yp].tipo == '-'){
43          cond1 = 1;
44      }
45      bool cond2 = 0;
46      if((yp == y + 1 || yp == y - 1) && Tab.pieces[xp][yp].tipo != '-'){
47          cond2 = 1;
48      }
49      if(frente && (cond1 || cond2)){
50          return 1;
51      }
52      std::cout << "Movimento invalido" << std::endl;
53      std::cout << "[DICA] O peao se move 1 casa pra frente e captura nas diagonais
    frontais." << std::endl;
54      return 0;
55  }
```

◆ PeaoP()

bool Game::PeaoP (int x,
int y,
int xp,
int yp
)

```
56                                                  {
57      bool frente = 0;
58      if(turn == 1) {
59          if(xp == x - 2){
60              frente = 1;
61          }
62      }
63      if(xp == x - 1){
64          frente = 1;
65      }
66      bool cond1 = 0;
67      if(y == yp && Tab.pieces[xp][yp].tipo == '-'){
68          cond1 = 1;
69      }
70      bool cond2 = 0;
71      if((yp == y + 1 || yp == y - 1) && Tab.pieces[xp][yp].tipo != '-'){
72          cond2 = 1;
73      }
74      if(frente && (cond1 || cond2)){
75          return 1;
76      }
77      std::cout << "Movimento invalido" << std::endl;
78      std::cout << "[DICA] O peao se move 1 casa pra frente e captura nas diagonais
    frontais." << std::endl;
79      return 0;
80  }
```

◆ Queen()

bool Game::Queen (int x,
int y,
int xp,
int yp
)

```
223                                                 {
224     int difx = xp - x;
225     int dify = yp - y;
226     if(difx == dify || difx == -1*dify){
227         if(difx > 0 && dify > 0){
228             for(int i = 1; i < xp - x; i++){
229                 if(Tab.pieces[xp - i][yp - i].tipo != '-'){
230                     std::cout << "Movimento invalido" << std::endl;
231                     std::cout << "[DICA] A rainha nao eh capaz de pular pecas." <<
    std::endl;
232                     return 0;
233                 }
234             }
235             return 1;
236         }
237         if(difx > 0 && dify < 0){
238             for(int i = 1; i < xp - x; i++){
239                 if(Tab.pieces[xp - i][y - i].tipo != '-'){
240                     std::cout << "Movimento invalido" << std::endl;
241                     std::cout << "[DICA] A rainha nao eh capaz de pular pecas." <<
    std::endl;
242                     return 0;
243                 }
244             }
245             return 1;
246         }
247         if(difx < 0 && dify > 0){
248             for(int i = 1; i < x - xp; i++){
249                 if(Tab.pieces[x - i][yp - i].tipo != '-'){
250                     std::cout << "Movimento invalido" << std::endl;
251                     std::cout << "[DICA] A rainha nao eh capaz de pular pecas." <<
    std::endl;
252                     return 0;
253                 }
254             }
255             return 1;
256         }
257         if(difx < 0 && dify < 0){
258             for(int i = 1; i < x - xp; i++){
259                 if(Tab.pieces[x - i][y - i].tipo != '-'){
260                     std::cout << "Movimento invalido" << std::endl;
261                     std::cout << "[DICA] A rainha nao eh capaz de pular pecas." <<
    std::endl;
262                     return 0;
263                 }
264             }
265             return 1;
266         }
267     }
268     if((x != xp && yp == y) || (x == xp && y != yp)){
269         if((x != xp && yp == y) && x > xp){
270             for(int i = 1; i < x - xp; i++){
271                 if(Tab.pieces[xp - i][y].tipo != '-'){
272                     std::cout << "Movimento invalido" << std::endl;
273                     std::cout << "[DICA] A rainha nao eh capaz de pular pecas." <<
    std::endl;
274                     return 0;
```

```
275                 }
276             }
277             return 1;
278         }
279         if((x != xp && yp == y) && xp > x){
280             for(int i = 1; i < xp - x; i++){
281                 if(Tab.pieces[x - i][y].tipo != '-'){
282                     std::cout << "Movimento invalido" << std::endl;
283                     std::cout << "[DICA] A rainha nao eh capaz de pular pecas." <<
    std::endl;
284                     return 0;
285                 }
286             }
287             return 1;
288         }
289         if((x == xp && y != yp) && yp > y){
290             for(int i = 1; i < yp - y; i++){
291                 if(Tab.pieces[x][y - i].tipo != '-'){
292                     std::cout << "Movimento invalido" << std::endl;
293                     std::cout << "[DICA] A rainha nao eh capaz de pular pecas." <<
    std::endl;
294                     return 0;
295                 }
296             }
297             return 1;
298         }
299         if((x == xp && y != yp) && y > yp){
300             for(int i = 1; i < y - yp; i++){
301                 if(Tab.pieces[x][yp - i].tipo != '-'){
302                     std::cout << "Movimento invalido" << std::endl;
303                     std::cout << "[DICA] A rainha nao eh capaz de pular pecas." <<
    std::endl;
304                     return 0;
305                 }
306             }
307             return 1;
308         }
309     }
310     std::cout << "Movimento invalido" << std::endl;
311     std::cout << "[DICA] A rainha se move nas laterais e diagonais livremente." <<
    std::endl;
312     return 0;
313 }
```

◆ SavePlay()

```
void Game::SavePlay ( int         x,
                      int         y,
                      int         xp,
                      int         yp,
                      std::string nomea,
                      std::string nomedouser
                    )
```

salvar partidas

Retorna

salva a partida

```
430
    {
431     // Armazena no vetor a jogada para que seja salva posteriormente
432     Hist temp = {x, y, xp, yp};
433     History.push_back(temp);
434
435     std::string userFolderPath = "user/" + nomedouser;
436
437     // Verifica se o diretório do usuário existe e cria se não existir
438     if (!std::filesystem::exists(userFolderPath)) {
439         std::filesystem::create_directory(userFolderPath);
440     }
441
442     std::string userFilePath = userFolderPath + "/" + nomea + ".txt";
443     // Abre o arquivo do usuário e grava as ações do jogo
444     std::fstream arquivo(userFilePath, std::ios::out | std::ios::app);
445
446     arquivo << x << " " << y << " " << xp << " " << yp << std::endl;
447
448     arquivo.close();
449 }
```

◆ Torre()

bool Game::Torre (int x,
int y,
int xp,
int yp
)

```
 82                                                {
 83     bool cond1 = 0;
 84     if(x != xp && yp == y){
 85         cond1 = 1;
 86     }
 87     bool cond2 = 0;
 88     if(x == xp && y != yp){
 89         cond2 = 1;
 90     }
 91     if(!(cond1 || cond2)){
 92         std::cout << "Movimento invalido" << std::endl;
 93         std::cout << "[DICA] A torre se move apenas nas laterais." << std::endl;
 94         return 0;
 95     }
 96     if(cond1 && x > xp){
 97         for(int i = 1; i < x - xp; i++){
 98             if(Tab.pieces[xp - i][y].tipo != '-'){
 99                 std::cout << "Movimento invalido" << std::endl;
100                 std::cout << "[DICA] A torre nao eh capaz de pular pecas." <<
    std::endl;
101                 return 0;
102             }
103         }
104     }
105     if(cond1 && xp > x){
106         for(int i = 1; i < xp - x; i++){
107             if(Tab.pieces[x - i][y].tipo != '-'){
108                 std::cout << "Movimento invalido" << std::endl;
109                 std::cout << "[DICA] A torre nao eh capaz de pular pecas." <<
    std::endl;
110                 return 0;
111             }
112         }
113     }
114     if(cond2 && yp > y){
115         for(int i = 1; i < yp - y; i++){
116             if(Tab.pieces[x][y - i].tipo != '-'){
117                 std::cout << "Movimento invalido" << std::endl;
118                 std::cout << "[DICA] A torre nao eh capaz de pular pecas." <<
    std::endl;
119                 return 0;
120             }
121         }
122     }
123     if(cond2 && y > yp){
124         for(int i = 1; i < y - yp; i++){
125             if(Tab.pieces[x][yp - i].tipo != '-'){
126                 std::cout << "Movimento invalido" << std::endl;
127                 std::cout << "[DICA] A torre nao eh capaz de pular pecas." <<
    std::endl;
128                 return 0;
129             }
130         }
131     }
132     return 1;
133 }
```

◆ WhoTurn()

std::string Game::WhoTurn ( )

Parâmetros

verifica de quem eh o turno

```
19                          {
20      if(turn %2 == 0){
21          return "brancas";
22      }
23      if(turn %2 != 0){
24          return "pretas";
25      }
26      return "[erro]";
27  }
```

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

- C:/Users/guilh/Downloads/Trabalho_pds2/**Game.hpp**
- C:/Users/guilh/Downloads/Trabalho_pds2/**Game.cpp**