15 PUZZLE

Francisco Ribeiro, Matheus Bissacot, Sérgio Coelho

1 Descrição

Este é o "trabalho para a semana 2" feito por Francisco Ribeiro (up202104797), Matheus Bissacot (up202106708) e Sérgio Coelho (up202107951) para a cadeira de Inteligência Artificial do curso Ciência dos computadores, da faculdade de Ciências da Universidade do Porto.

A estrutura de dados utilizada para representar uma configuração de jogo foi uma lista de inteiros, por conta da importância da ordem ser essencial para a resolução do jogo.

A linguagem utilizada foi Java.

1.1 Objetivo

- Implementar um algoritmo para saber se um dado puzzle 15 é resolvível ou não.
- Implementação dos operadores que transformam uma configuração de jogo na sua sucessora.

1.2 Requisitos

Java 8 ou superior.

2 Execução

Para executar o programa, basta correr o seguinte comando:

```
$ java Puzzle
```

Introduzir a configuração inicial e final e apenas se for realizável o jogo será possivel interagir com os tabuleiros.

3 Referências

Link do vídeo que ajuda a entender melhor como conseguimos ver se um jogo é fazivél.

4 Código

Todo o código se encontra neste link, aqui estará todo completo com alguns extras apenas para ser mais bonito visualmente, mas nao estará no relatório para ser mais legível.

Implementação de uma classe para representar qualquer configuração de jogo com algumas funções auxiliares, como se é realizável chegar de um tabuleiro inicial para qualquer final "isSolvable", assim como também aplicar movimentos ('u' cima, 'd' baixo, 'r' direita, 'l' esquerda) a uma configuração (itable) certificando-se que são legais, "movements".

```
class TABULEIRO {
    int[] initialTable; //tabela inicial
    int[] finalTable;   //tabela final
    int[] itable;       //tabela interativa

    TABULEIRO(){
        initialTable = new int[16];
        finalTable = new int[16];
        itable=new int[16];
    }

    void read(Scanner in){
        for(int i = 0; i < 16; i++) initialTable[i] = in
   .nextInt();
```

```
14          for(int j = 0; j < 16; j++) finalTable[j] = in.
    nextInt();
15          itable= Auxiliares.copyList(initialTable);
16      }
17
18
19      //limites
20      public boolean limites(int pos, char direcao) {
21          switch(direcao) {
22              case 'u': //Up
23                  if(pos >= 0 && pos < 4) {
24                      return false;
25                  }
26                  break;
27              case 'd': //Down
28                  if(pos >= 12 && pos < 15) {
29                      return false;
30                  }
31                  break;
32              case 'l': //Left
33                  if(pos == 0 || pos == 4 || pos == 8 ||
    pos == 12) {
34                      return false;
35                  }
36                  break;
37              case 'r': //Right
38                  if(pos == 3 || pos == 7 || pos == 11 ||
    pos == 15) {
39                      return false;
40                  }
41                  break;
42          }
43
44          return true;
45      }
46
47
48      //modificar com os movimentos, "u" para cima, "d"
    para baixo, "l" para esquerda e "r" para direita
49      public boolean movements(char direction) {
```

```
50          int pos = Auxiliares.findIndex(itable, 0);
51          int temp = itable[pos];
52
53          switch(direction) {
54              case 'u': //Up
55                  if(limites(pos, 'u')) {
56                      itable[pos] = itable[pos - 4];
57                      itable[pos - 4] = temp;
58                      return true;
59                  }
60              break;
61              case 'd': //Down
62                  if(limites(pos, 'd')) {
63                      itable[pos] = itable[pos + 4];
64                      itable[pos + 4] = temp;
65                      return true;
66                  }
67              break;
68              case 'l': //Left
69                  if(limites(pos, 'l')) {
70                      itable[pos] = itable[pos - 1];
71                      itable[pos - 1] = temp;
72                      return true;
73                  }
74              break;
75              case 'r': //Right
76                  if(limites(pos, 'r')) {
77                      itable[pos] = itable[pos + 1];
78                      itable[pos + 1] = temp;
79                      return true;
80                  }
81              break;
82          }
83
84          return false;
85      }
86
87
88      //imprimir o jogo atual
89      public String toString() {
```

```
90          String res = "";
91          for(int i = 0; i < 16; i++) {
92              if(i%4 == 0) {
93                  if (i==0) res = "\n";
94                  else res += "|\n";
95                  res += "+---------------+\n";
96              }
97              if (itable[i]<10){
98                  res += "|"+ itable[i] + "   ";
99              }
100             else{
101                 res += "|"+itable[i] + "  ";
102             }
103         }
104         res+="|"+"\n+---------------+";
105         res += "\n";
106         return res;
107     }
108
109
110     int SaltosBranco(){
111         int pos = Auxiliares.findIndex(initialTable, 0);
112         int pos_final = Auxiliares.findIndex(finalTable,
    0);
113         //as linhas sao os grupos (0-3)(4-7)(8-11)
    (12-15)
114         int dist_vertical = (pos_final/4) - (pos/4);
115         int dist_horizontal = (pos_final%4) - (pos%4);
116         //distancia tem de ser positiva
117         if(dist_vertical < 0) {
118             dist_vertical = dist_vertical * -1;
119         }
120         if(dist_horizontal < 0) {
121             dist_horizontal = dist_horizontal * -1;
122         }
123         int distancia_final = dist_vertical +
    dist_horizontal;
124         return distancia_final;
125     }
126
```

```
127     //Verificar se um puzzle e realizavel
128     public boolean isSolvable() {
129         int[] temp = Auxiliares.copyList(initialTable);
130         int inv = 0; //Permutations
131         int distancia_brancos = SaltosBranco(); //
    Distancia dos brancos
132 
133         for(int i = 0; i < 16; i++) {
134             if(temp[i] != finalTable[i]) {
135                 for(int j = Auxiliares.findIndex(temp,
    finalTable[i]); j > i; j--) {
136                     int temporario = temp[j - 1];
137                     temp[j - 1] = temp[j];
138                     temp[j] = temporario;
139                     inv++;
140                 }
141             }
142         }
143 
144         return (distancia_brancos%2 == 0) == (inv%2 ==
    0);
145     }
146 
147 }
```

Para ser possível correr o código foi preciso implementar alguns métodos auxiliares, o nome dos mesmos são auto-explicativos.

```
1 class Auxiliares{
2     public static int findIndex(int[] table, int number)
      {
3         int res = -1;
4         for(int i = 0; i < table.length; i++) {
5             if(table[i] == number) {
6                 res = i;
7                 break;
8             }
9         }
```

```
10
11          return res;
12      }
13      //metodo para copiar duas listas de inteiros
14      public static int[] copyList(int[] list) {
15          int[] res = new int[list.length];
16          for(int i = 0; i < list.length; i++) {
17              res[i] = list[i];
18          }
19
20          return res;
21      }
22
23      //compara duas listas de inteiros
24      public static boolean compare(int[] list1, int[]
    list2) {
25          boolean res = true;
26          for(int i = 0; i < list1.length; i++) {
27              if(list1[i] != list2[i]) {
28                  res = false;
29                  break;
30              }
31          }
32
33          return res;
34      }
35
36  }
```

Por fim, para ser possível testar criamos uma classe com o método main em que cria um jogo e assim lê os tabuleiros dizendo se é realizável ou não. Se for, é possível interagir com o tabuleiro para se tentar chegar á configuração final desejada.

```
1  import java.util.Scanner;
2
3
4  class Puzzle{
```

```
5     static TABULEIRO teste= new TABULEIRO();
6
7     public static void main(String[] args) {
8         //Input Lines
9         Scanner in = new Scanner(System.in);
10        teste.read(in);
11
12        if (teste.isSolvable()) {
13            System.out.println("Tem solucao");
14            interact(in);
15        } else {
16            System.out.println("Nao tem solucao");
17            System.out.println("A sair...");
18        }
19    }
20
21    //interagir com o puzzle
22    public static void interact(Scanner in){
23        System.out.println(teste);
24        in.nextLine();
25
26        //enquanto o carater nao for "q" ou "Q" continua
     a pedir input
27        //o input e um carater de cada vez separado por
    mudanca de linha
28        //se a table.itable for igual a finaltable, o
    puzzle esta resolvido e sai do loop
29        while(in.hasNext()){
30            char input = in.next().charAt(0);
31            if(input == 'q' || input == 'Q') {
32                System.out.println("A sair... :(");
33                break;
34            }
35            if(teste.movements(input)) {
36                System.out.println(teste);
37            }
38            else{
39                System.out.println("Movimento invalido,
    tente novamente");
40            }
```

```
41
42              if(Auxiliares.compare(teste.itable, teste.
    finalTable)) {
43                  System.out.println("Puzzle Resolvido");
44                  break;
45              }
46          }
47      }
48  }
```

Possível teste:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0
1 2 3 4 5 6 7 8 9 10 11 12 0 13 14 15
l
l
d
l
```