Trabalho MC322

Etapa 01

Resta Um

O **Resta Um** (*Peg Solitaire* em inglês) é um jogo antigo cujo primeiro registro é de 1687 (https://en.wikipedia.org/wiki/Peg_solitaire). Ele é constituído de um tabuleiro em formato de cruz com todas as casas preenchidas por pedras, exceto a do meio.



Por Denis A. C. Conrado - Author, Attribution, https://commons.wikimedia.org/w/index.php?curid=1969740



By Photo by Gnsin edited by WolfgangW. - Photo by Gnsin GFDL, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=198914

Trata-se um jogo para se jogar sozinho. Em cada movimento do jogo, cada peça pode "comer" outra pulando-a (no mesmo estilo do jogo de damas). Tal como na dama, o pulo só

pode acontecer se do outro lado houver um espaço vago; a peça comida desaparece. O único movimento permitido é esse de "comer" e ele só pode ser realizado nas quatro direções: cima, baixo, direita e esquerda; não são permitidos movimentos diagonais.

O desafio final é só deixar uma única pedra no tabuleiro. O risco é que alguma pedra fique sem nenhum vizinho pois, nesse caso, ela não poderá se mover.

Abaixo segue uma animação demonstrando o funcionamento do jogo.



By Joho345 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=70948834

O seu desafio aqui é escrever um conjunto de classes que simule o funcionamento do **Resta Um**. Deve haver pelo menos uma classe cujo objeto represente o tabuleiro e outra cujos objetos representem as peças do tabuleiro.

Os movimentos a serem jogados no tabuleiro serão recebidos a partir da chamada de um método de um objeto, cuja classe já está codificada (veja detalhamento abaixo). O estado inicial do tabuleiro e seu novo estado depois de cada movimento deve ser mostrado no console na forma de caracteres, como ilustrado a seguir.

Tabuleiro:

O tabuleiro deverá ser printado da forma mostrada acima. As posições do tabuleiro serão descritas utilizando as colunas de 'a' até 'g' e as linhas de 1 até 7, onde uma posição é dada combinando coluna com linha, por exemplo, c4 consiste na coluna c e na linha 4.

Entrada do Programa:

A entrada do programa será um arquivo `.csv` contendo todos os comandos a serem executados pelo jogo. Cada comando consistirá de uma posição inicial (posição da peça a ser movida) e uma posição final (posição para onde a peça selecionada será movida). No arquivo `.csv` os comandos serão separados por vírgulas, ou seja, cada comando, contendo posição inicial e final, será separado por vírgula do próximo. Está sendo disponibilizado uma classe (CSVReader) para ler esse arquivo .csv e retornar a entrada pronta num vetor de String, onde cada posição desse vetor consiste da posição inicial e da posição final separadas por ":".

Exemplo de uma posição do vetor contendo as entradas: **f4:d4**. Nesse exemplo, a posição inicial é a coluna **f** e a linha **4** e a posição final é a coluna **d** e a linha **4**.

Classe CSVReader e como usá-la:

```
public class CSVReader {
        private String dataSource;
        private String[] commands;
        public CSVReader() {
                this.commands = null;
                this.dataSource = null;
        }
        public String getDataSource() {
                return dataSource;
        }
        public void setDataSource(String dataSource) {
                 this.dataSource = dataSource;
                 if (dataSource == null) {
                commands = null;
        } else
                readCSV();
        }
         public String[] requestCommands() {
                return commands;
        }
        private void readCSV() {
                try {
                         BufferedReader file = new BufferedReader(new FileReader(dataSource));
                         String line = file.readLine();
                         if (line != null) {
                         commands = line.split(",");
                         line = file.readLine();
```

```
}
file.close();
} catch (IOException erro) {
    erro.printStackTrace();
}
}
```

Para utilizar a classe CSVReader, deve-se fazer uma instância dela e passar o caminho do arquivo .csv de entrada no método setDataSource():

```
CSVReader csv = new CSVReader();
csv.setDataSource("../../src/db/arq001.csv");
```

Agora, para obter o vetor String com a entrada do programa (arquivo .csv), basta chamar o método requestCommands():

String commands[] = csv.requestCommands();

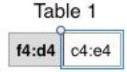
Pronto, agora você tem disponível um vetor onde cada posição dele contém o comando a ser executado.

Saída do Programa:

A saída do programa deve ser impressa na saída padrão STDOUT e deve imprimir a posição inicial (source) e final (target) da peça que vai ser movimentada na rodada, bem como o estado do tabuleiro após a movimentação. Antes da primeira movimentação imprima o estado inicial do tabuleiro, como demonstrado abaixo:

```
Tabuleiro inicial:
7
     PPP
6
     PPP
5 P P P P P P
4 P P P - P P P
3 P P P P P P
2
     PPP
     PPP
 abcdefq
Source: f4
Target: d4
     PPP
     PPP
5 P P P P P P
4 P P P P - -
3 P P P P P P
     PPP
     PPP
 abcdefg
Source: c4
Target: e4
     PPP
     PPP
5 P P P P P P
4 P P - - P - P
3 P P P P P P
     PPP
     PPP
 abcdefg
```

A saída de exemplo mostrada na imagem acima foi resultado do seguinte '.csv':



Então, o vetor de string dado pelo método requesCommands() descrito acima fica da seguinte forma: {"f4:d4", "c4:e4"}.

Entrega

A entrega deve ser realizada via Github. Vocês devem submeter o código-fonte no seu Github e enviar, pela atividade do Classroom, o link do seu repositório no Github. A imagem abaixo mostra como enviar um link na atividade:

- 1) Entre na Atividade;
- 2) Clique em "Adicionar ou criar";
- 3) Link

