UNBC

Laboratory Assignment

Computer Science 101—Winter 2022

Backgammon Moves (revised)

Due Date:

This assignment is due Friday, March 04 *by the beginning of lecture*.

Please submit via `learn.unbc.ca`. See page 4 for more instructions on what to submit.

Working with Others

You may, but are not required, work with your teammates and use project-based code to complete this lab.

You **must** state in code comments who you worked with, even if you worked alone.

Backgammon Moves

Your goal in this assignment is to design a program that can compute all of the legal moves available in a particular situation in a game of backgammon.

More specifically, input to the program will be via System.in and will consist of a sequence of lines, each line representing a game situation as described in § **??**.

For each game situation, your program should compute all of the possible legal move combinations. Your program should print this set of move combinations on System.out.

Your program should quit when it reaches end-of-file on System.in.

Input Format

Each input line contains a board configuration and a dice roll separated by a semicolon.

The board configuration is represented by a sequence of 26 integers separated by commas.

- The first number is ≤ 0. Its absolute value is the number of the opponent's pieces on the bar.

- The second number is ≥ 0. Its value is the number of your pieces on the bar.
- The next 24 numbers represent the pieces on the board points, in a sequence by decreasing point number.
  - A **0** represents an empty pooint.
  - A negative integer represents a point occupied by the opponent's pieces, with the absolute value being the total number of pieces.
  - A positive integer represents a point occupied by your pieces, with the value being the total number of pieces.

The dice roll is represented by pair of integers, each between 1 and 6, separated by a comma.

Sample Input Format

Here is a sample input file:

```
0,0, 2,0,0,0,0,-5, 0,-3,0,0,0,5, -5,0,0,0,3,0, 5,0,0,0,0,-2; 6,1
0,0, 2,0,0,0,0,-5, 0,-3,0,0,0,5, -5,0,0,0,3,0, 5,0,0,0,0,-2; 5,5
```

Output Format

The overall output format for a single board situation is a sequence of legal plays separated by semi-colons and terminated by a period and new line.

If no plays are possible, the response should be a period followed by a new line.

Each play normally consists of two (or four in the case of doubles) moves separated by commas, although there may be one (or two or three in the case of doubles) because there are not enough possible legal moves. A move, in turn, consists of a source point number, followed by a hyphen, followed by a destination point number. A move from the bar has a source point number of 25; a bearing off move has a destination point number of 0. Two moves of the same piece may be abbreviated, for instance, `24-18,18-13` may be abbreviated as `24-18-13`. If the destination point has an opponent's blot, the move should be immediately followed by an `x`.

Distinct moves

It is often possible to reach the same outcome by two distinct sequence of moves. For instance, a 3-1 roll might be played as `8-5,5-4` or `8-7,7-4`. As long as there are no blots hit, these moves accomplish the same thing. In this case, it is your choice as to whether to report one or two plays.

Plays that involve moving pieces from the bar must begin with the moves from the bar.

Sample Output Format

Here is sample output:

```
1 24-18,24-23;24-18,8-7;24-18,6-5;13-7,24-23;13-7,8-7;13-7-6;
2 13-7,6-5;8-2,24-23;8-2,8-7;8-2,6-5.
3 13-8,13-8,13-8,13-8;13-8,13-8,13-8-3;13-8,13-8-3,8-3;13-8-3,8-3,8-3.
```

Lines 1–2 are actually one long line.

Further Specifications

There is now a sample input file `data.txt` on the blackboard site. This should produce output somewhat like the sample output file `out.txt` also now available on the blackboard site. (Note that the output format allows verious different ways to format the same correct answer.)

Command-line execution

If you run `java` from the command line, you should be able to execute something like

```
  java lab4.Test < data.txt > my-out.txt
```

to capture the output from your program in a file `my-out.txt`. Depending where you put the text files you may need to use longer names for the files. The precise name to use in place of `lab4.Test` depends on how you have your packages arranged.

IDE execution

It is impossible to describe exactly how to redirect input and output for all possible IDEs and configurations.

Hand-in materials

⇒ Submit:

- all of your `.java` files, preferably as a `.jar` file, but as a `.zip` file is also acceptable;
- a text file `my-out.txt` containing the output produced when the input is as in `data.txt`.
- a plain text (`*.txt`) file describing how you produced `my-out.txt`, and if desired, a list of known problems remaining (that is, cases, that your code does not yet handle correctly).