Programming Assignment 1: PathOfDestruction

Dr. Sharma Thankachan

COP 3503 – Fall 2018

Due Wednesday, September 12th, at 11:59pm

1. Deliverable:

A source file named *PathOfDestruction.java* written in Java.

2. Objective:

The objective of this assignment is to get you warmed up for CS2 and for a semester of Java. This assignment requires only basic Java knowledge and an understanding of CS1 concepts like backtracking, and is not meant to take a long time.

3. Description:

The game of American checkers has two types of pieces: regular checkers and king checkers. Regular checkers placed on a checkerboard follow one restriction: all checkers must be on the same color as all other checkers placed. Figures 1 and 2 below are examples of valid checkerboards (where a capital O symbolizes a checker). Figure 3 is invalid because not all of the checkers occupy the same color. For this assignment the color of the checkers does not matter.

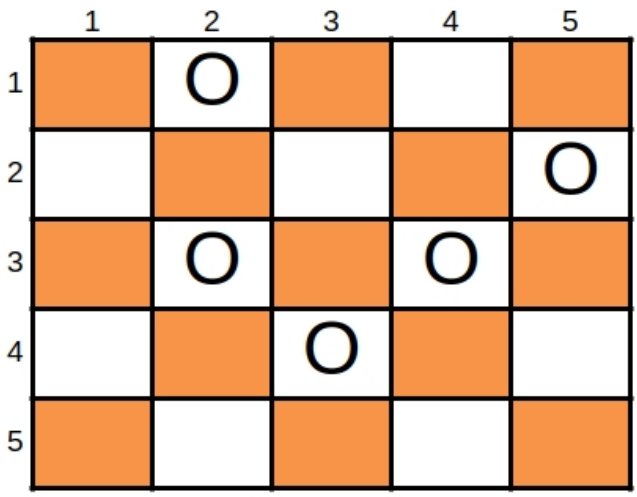

Figure 1 – A valid board.

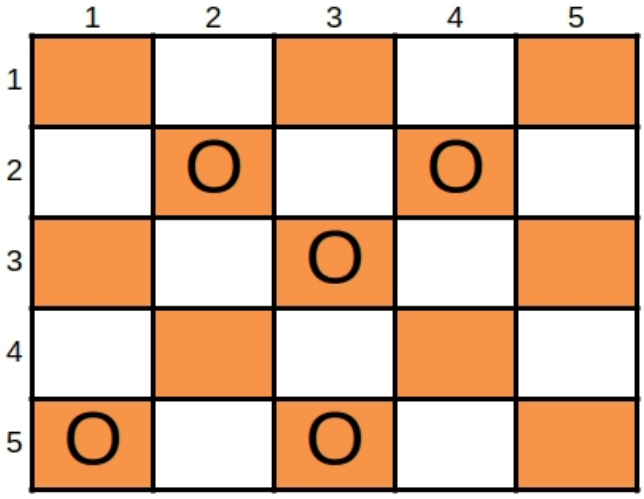

Figure 2 – Another valid board.

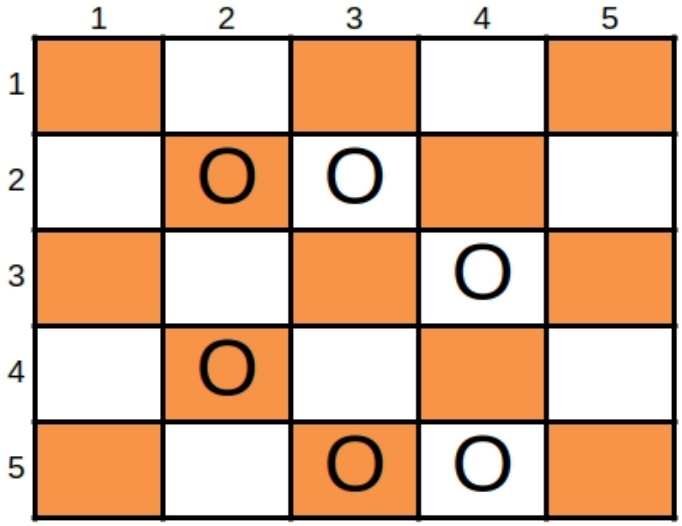

Figure 3 – An invalid board.

King checkers are able to move more freely than normal checkers on a checkerboard. They can move diagonally by one space in any direction (Figure 4), or they can hop over other checkers in any diagonal direction (Figure 5). These hops can only occur if the hopped checker is directly, diagonally adjacent. Whenever a checker is hopped, it is removed from the board (and thus cannot be hopped again). Kings are represented with a capital K.

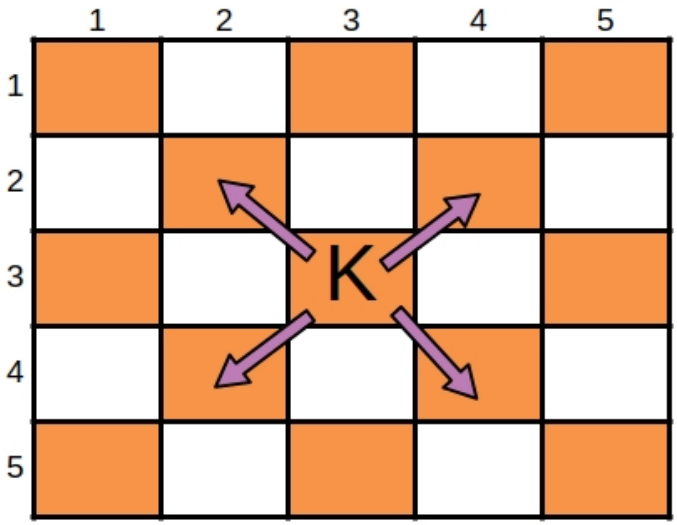

Figure 4 – A king's possible moves.

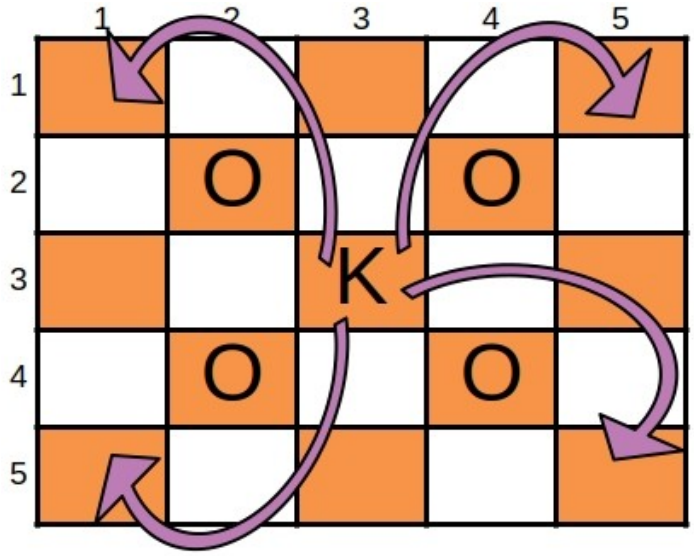

Figure 5 – A king's possible jumps.

When a king hops a checker, it can then attempt to make another hop if one is available from its new position. This results in a "chain" of hops. Any chain of hops which jumps over every checker on the board is a "path of destruction" and this path removes all checkers (except the king) from the board. The next page gives an example of a possible path of destruction on an arbitrary checkerboard, (Figure 6) and an example of a checkerboard where no path of destruction is possible, (Figure 7). The primary goal of this assignment is to check if a path of destruction exists for a variable-sized checkerboard with an arbitrary layout of checkers.

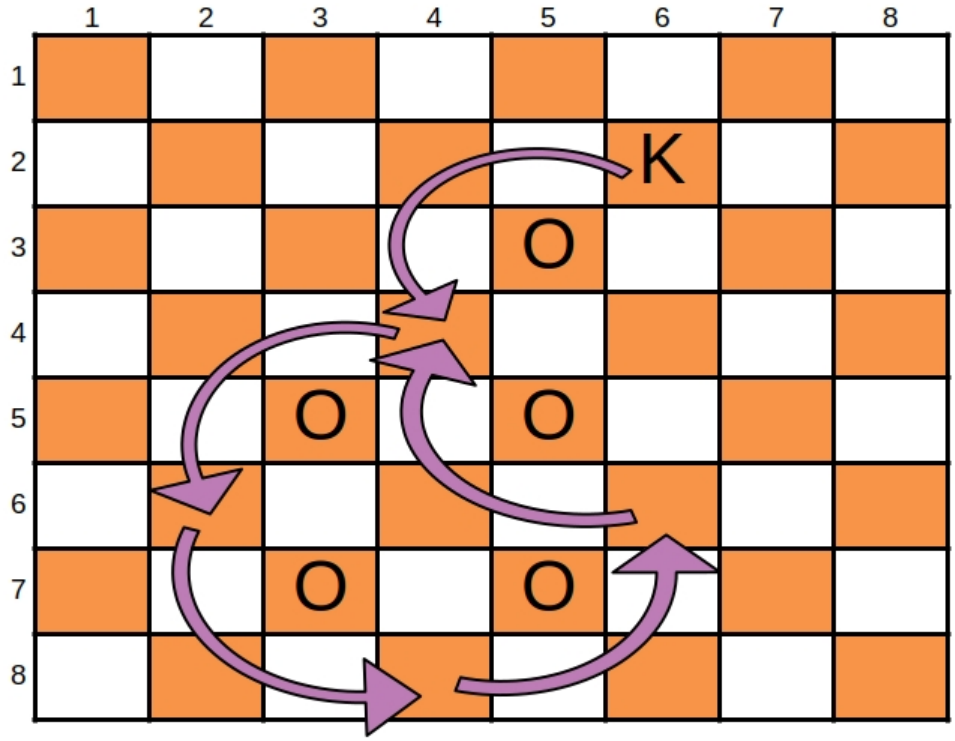

Figure 6 – When a king begins at position 6, 2 he can complete a path of destruction that removes all pieces from the board.

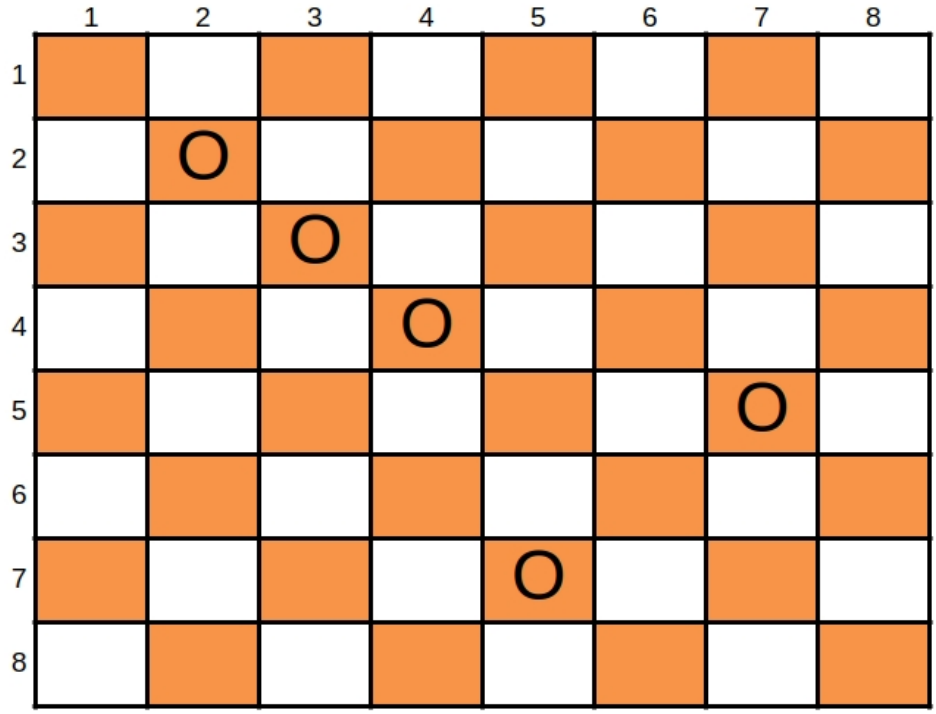

Figure 7 – No start position exists on this board where a king could begin and ultimately complete a path of destruction.

4. Requirements:

This assignment must be named *PathOfDestruction.java* and contain a public class named *PathOfDestruction*. Within this public class you must write these 2 methods:

```
public static boolean isValidBoard(String boardFile) throws IOException
```

This method will return true if the checkerboard described in the file *boardFile* has a valid layout of checkers. Recall that all checkers on a checkerboard must be on the same "color" as all other checkers on that board. This can be determined mathematically, so to write this function properly you'll need to discover that relationship. Refer to the input specifications for more detail about the potential contents of *boardFile.* Return false if the checkerboard is invalid.

NOTE: It is highly recommended to draw out some checkerboards to try and deduce the validity relationship on paper before proceeding.

```
public static boolean hasPathOfDestruction(String boardFile) throws IOException
```

This method will return true if the checkerboard described in the file *boardFile* contains a path of destruction, otherwise return false.

NOTE: Boards containing no checkers have paths of destruction, since a king placed on such a board can follow a path that results in all of the checkers on the board being removed (by just staying put). This is a type of vacuous truth.

NOTE: Boards that contain an invalid layout of checkers cannot possibly have a path of destruction.

5. Input Specification:

This is a sample input file that represents the checkerboard in Figure 6:

```
8 8 5
5 3
5 5
3 5
3 7
5 7
```

The first line of every checkerboard file will contain 3 integers.

- The first is the maximum *x* coordinate that a board may have. All checkers must have *x* coordinates inside of the range [1, maximum *x*] on a valid board. The maximum *x* coordinate will always be greater than 0.
- The second is the maximum *y* coordinate. All checkers must have *y* coordinates inside of the range [1, maximum *y*] on a valid board. The maximum *y* coordinate will always be greater than 0.
- The third is the number of checkers on this board. This number will always be accurate and will be greater than or equal to 0.

The second line, and every line thereafter contain the *x* and *y* integer coordinates of a single checker, respectively. Checkers are guaranteed to be unique, but their coordinates are not guaranteed to be valid coordinates. For example, the valid checker (1, 7) could be found inside the sample given above, as well as the invalid checkers (400, 13) and (0, -22). **It is your responsibility to check the coordinates of each checker as you parse the file to ensure they are within bounds for that given board.**

6. Testing:

To test, you will want to run the provided test script in a Linux environment. Your assignment will be graded on Eustis, and if your assignment does not work on Eustis then it **does not work.** For the script to function, you will need to set up your directory (folder) to look like this:

```
TestSuite/
        PathOfDestruction.java
        PathOfDestructionTester.java
        Test.sh
        Inputs/
                Input1.txt
                Input2.txt
                ...
```

To run the test script, use this command:

```
$ bash Test.sh
```

NOTE: The "$" character is not part of the command. It is a symbol frequently used to indicate the beginning of a terminal command.

7. Grading Criteria:

The grading criteria for this assignment is as follows:

- ***50%*** *hasPathOfDestruction()* unit tests
- ***50%*** *isValidBoard()* unit tests

8. Final Notes:

Comments will not be graded generally, however I reserve the right to penalize assignments that include no comments, or whose comments are horrible. In the real world, comments are extremely useful and expected from developers, so get used to writing them. I also reserve the right to penalize assignments with atrocious/inconsistent style (or lack thereof). This does not mean that you will lose points for using one particular style over another—it only means that you need to apply your style consistently.

You are free to discuss concepts and high level ideas about this assignment with your peers, however code sharing and code review is strictly prohibited.

Examples of **acceptable** peer questions about this assignment look like this:

- What do you think is the best approach to implement *isValidBoard()*?
- I used an array in *hasPathOfDestruction()...* do you think a set would be better?
- Should I solve this problem using the backtracking method?

Examples of **unacceptable** peer questions about this assignment look like this:

- Can you review my code for *isValidBoard()*?
- Here's my *hasPathOfDestruction()* code. How could I change it from an array implementation to a set implementation?
- My backtracking solution has a bug. Can you look and tell me how to fix it?

You MUST do all of the following:

- Include your name and NID in a comment at the top of your submission.
- Ensure that your submission compiles and runs on Eustis.
- Name your submission *PathOfDestruction.java.*
- Submit the assignment before the deadline.
- Write the assignment in Java.

If you do not follow the guidelines above you will receive a 0.