Knight's Tour – exercise 2

This is an example application for Verification and Validation course at UPM.

This is an example application for Verification and Validation course at UPM. The application finds solutions for Knight's tour problems.

The team members are:

- Martin Lipták
- Gábor Nagy
- Andreea Oprişan

We recommend reading the documentation files on GitHub: http://github.com/martinliptak/knights-tour/ or in "md" format accessible in the attached "zip" file.

Exercise 2

Injecting faults in the code.

The injected faults are signed in the code by comments and a short description. Find their list and full description below.

List of the injected faults:

1. Fault: ERROR-1 causes that the program is not testing whether the given arguments are integer values or not.

   The given argument should be tested one by one by the following code: (file *Main.java*, line: *58, 86, 112, (149)*)

```
public static boolean isInteger(String s) {
      try{
            Integer.parseInt(s);
      }
      catch(NumberFormatException e){
            return false;
      }

      return true;
}
```

For getting this fault the places where the program should call this method are put in comment. For ex:

```
/*
if ( !isInteger(args[2]) ){
            System.out.println("The third argument is not
an integer!\nPlease folow the instuctions and use an
integer value!\n");
            return false;
} */
```

For activating the fault run the program with not integer arguments. For ex:

```
bash $ java Main 5.4
```

2. Fault: ERROR-2 causes that the size of the table, given by the first argument, is not tested. With this fault is allowed to put lower values than 5 or higher values than 7. Lower values cause strange behavior and not correct outputs. Higher values cause very long waiting time for the result.

   The first argument should be tested by the following code:
   (file *Main.java*, line: *69*)

```
if (sizeOfTable < 5 || sizeOfTable > 7){
                    System.out.println("The first argument
is not valid!\nPlease folow the instuctions and use a number
from 5 to 7!\n");
                    return false;
}
```

   For getting this fault the code above is put in comment.
   For activating this fault use any integer for the first argument what is not from the interval <5,7> .

   *hint: for killing the process in the case of long waiting period caused by higher number than 5 use the command-line shortcut “ctrl+c” (mac: “cmd+c”).

3. Fault: ERROR-3 causes that the user can not choose the first and the last position for starting on the axis X. That means that the argument for axis X can be only from interval <2, “size of table“ - 1>.

   The fault is caused by a bad condition:
   (file *Main.java*, line: *97*)

```
if ( x <= 1 || x >= sizeOfTable)
```

   The right code would be:

```
if ( x < 1 || x > sizeOfTable)
```

The fault is showing up by using number 1 or the maximum number according to size of table for the second argument. For ex:

```
bash $ java Main 5 1 2
bash $ java Main 5 5 2
```

4. Fault: ERROR-4 causes that the value (integer value) of the third argument is not tested at all. The condition contains a logical fault what causes that it will never detect any wrong value of the argument.

   The fault is caused by the condition:
   (file *Main.java*, line: *122*)

   ```
   if ( y < 1 && y > sizeOfTable)
   ```

   The right code would be:

   ```
   if ( y < 1 || y > sizeOfTable)
   ```

   The fault is showing up by using any kind of integer value for the third argument. For ex:

   ```
   bash $ java Main 5 2 0
   bash $ java Main 5 2 7
   ```

5. Fault: ERROR-5 is breaking the logic of the knight's steps. With this fault 4 more steps are allowed and the most of the solutions are not correct.

   The fault is caused by adding 4 more possible steps into the logic:
   (file *State.java*, line: *47*)

   ```
   pushNewPositionIfValid(states, this.x - 2, this.y + 2);
   pushNewPositionIfValid(states, this.x - 2, this.y - 2);
   pushNewPositionIfValid(states, this.x + 1, this.y - 1);
   pushNewPositionIfValid(states, this.x + 1, this.y + 1);
   ```

   The code shouldn't contain these lines at all.
   The result of the fault is mostly visible at every solution of the program.
   For example using steps x+1 and y+1:

   ```
   bash $ java Main 5 2 2
   Size 5x5
   Solving field from start-position 2 2
   10 14 12 17  4
   13 11 16  3  6
   15  9  2  5 18
   22  1  7 24 20
    8 23 21 19 25

   Execution time: 131ms
   ```

6. Fault: ERROR-6 causes that the number of given arguments are not properly tested. Instead of allowing only one or three arguments, the condition is catching only bigger numbers than 3.

   The fault is caused by the condition:
   (file *Main.java*, line: *43*)

   ```
   if (args.length > 3)
   ```

   The right code would be:

   ```
   if (args.length != 1 && args.length != 3)
   ```

   The fault is showing up by using 2 arguments or by not using any argument:

   ```
   bash $ java Main 5 2
   bash $ java Main
   ```

7. Fault: ERROR-7 causes that the given arguments are not properly tested whether they are numeric values. The testing condition is testing only the ASCII value of the first letter of the given number.

   The fault is caused by the condition:
   (file *Main.java*, line: *140*)

   ```
   if ( ((int) s.charAt(0)) >= 48 && ((int) s.charAt(0)) <=
  57) {
              return true;
     }
   return false;
   ```

   The right code would be:

   ```
   return s.matches("[-+]?\\d*\\.?\\d+");
   ```

   For getting this fault put a two letter argument in place of any of the arguments. The argument should start with a number and continue with a non-numeric value. For ex:

   ```
   bash $ java Main 5a
   bash $ java Main 5 2@ 3
   ```

8. Fault: ERROR-8 causes that program can take a very long time when run with certain inputs.

   Checking of maximum iteration count is commented out.
   (file *Main.java*, line: *19*)

```
/*
        steps++;
        if (steps > 100000000)
                return null; // limit exceeded, no solution found
*/
```

   For getting this fault use a big number for the size of table (argument 1):

```
bash $ java Main 20
```