See the Assessment Guide for information on how to interpret this report.

ASSESSMENT SUMMARY

```
Compilation:  PASSED
API:          PASSED

SpotBugs:     PASSED
PMD:          PASSED
Checkstyle:   FAILED (0 errors, 1 warning)

Correctness:  22/28 tests passed
Memory:       No tests available for autograding.
Timing:       No tests available for autograding.

Aggregate score: 80.71%
[ Compilation: 5%, API: 5%, Style: 0%, Correctness: 90% ]
```

ASSESSMENT DETAILS

```
The following files were submitted:
----------------------------------
 134 Feb  5 19:47 AnnotationType.java
1.4K Feb  5 19:47 Birthday.class
1006 Feb  5 19:47 Birthday.java
147K Feb  5 19:47 COS_126.xml
142K Feb  5 19:47 COS_126.xml.2020.1
 189 Feb  5 19:47 Class.java
 850 Feb  5 19:47 Computer\ Science.iml
1.2K Feb  5 19:47 DiscreteDistribution.class
 582 Feb  5 19:47 DiscreteDistribution.java
 128 Feb  5 19:47 Enum.java
 268 Feb  5 19:47 File\ Header.java
 133 Feb  5 19:47 Interface.java
1.7K Feb  5 19:47 Minesweeper.class
1.5K Feb  5 19:47 Minesweeper.java
4.2K Feb  5 19:47 Project.xml
 739 Feb  5 19:47 ThueMorse.class
 703 Feb  5 19:47 ThueMorse.java
1.1K Feb  5 19:47 checkstyle-idea.xml
 15K Feb  5 19:47 codeInsightSettings.xml
 142 Feb  5 19:47 codeStyleConfig.xml
 384 Feb  5 19:47 compiler.xml
 201 Feb  5 19:47 encodings.xml
 267 Feb  5 19:47 externalDependencies.xml
 290 Feb  5 19:47 file.template.settings.xml
 352 Feb  5 19:47 findbugs-idea.xml
 560 Feb  5 19:47 introcs.xml
 190 Feb  5 19:47 lift.xml
112K Feb  5 19:47 logo.png
 215 Feb  5 19:47 misc.xml
  58 Feb  5 19:47 module-info.java
 273 Feb  5 19:47 modules.xml
 102 Feb  5 19:47 package-info.java
 173 Feb  5 19:47 profiles_settings.xml
 357 Feb  5 19:47 saveactions_settings.xml
5.7K Feb  5 19:47 workspace.xml


********************************************************************************
*  COMPILING
********************************************************************************


% javac DiscreteDistribution.java
*-----------------------------------------------------------

% javac ThueMorse.java
*-----------------------------------------------------------

% javac Birthday.java
*-----------------------------------------------------------

% javac Minesweeper.java
```

```
*-----------------------------------------------------------


================================================================


Checking the APIs of your programs.
*-----------------------------------------------------------
DiscreteDistribution:

ThueMorse:

Birthday:

Minesweeper:

================================================================


********************************************************************************
*  CHECKING STYLE AND COMMON BUG PATTERNS
********************************************************************************


% spotbugs *.class
*-----------------------------------------------------------


================================================================


% pmd .
*-----------------------------------------------------------


================================================================


% checkstyle *.java
*-----------------------------------------------------------

% custom checkstyle checks for DiscreteDistribution.java
*-----------------------------------------------------------

% custom checkstyle checks for ThueMorse.java
*-----------------------------------------------------------

% custom checkstyle checks for Birthday.java
*-----------------------------------------------------------

% custom checkstyle checks for Minesweeper.java
*-----------------------------------------------------------
[WARN] Minesweeper.java:10: Calling 'Math.random()' in more than one place suggests poor design in this program. [Design]
Checkstyle ends with 0 errors and 1 warning.


================================================================


********************************************************************************
*  TESTING CORRECTNESS
********************************************************************************

Testing correctness of DiscreteDistribution
*-----------------------------------------------------------
Running 6 total tests.

Test 1: check output format
  % java DiscreteDistribution 9 1 1 1 1 1 1
  2 1 3 5 5 1 1 3 3

  % java DiscreteDistribution 8 10 20 30 40 50 60 50 40 30 20 10
  4 6 10 2 6 4 3 3

  % java DiscreteDistribution 7 10 10 10 10 10 50
  6 5 4 2 6 6 4

  % java DiscreteDistribution 6 50 50
  2 2 2 2 2 1

  % java DiscreteDistribution 5 80 20
  1 1 1 1 1

  % java DiscreteDistribution 4 301 176 125 97 79 67 58 51 46
```

```
    7 4 4 9

    % java DiscreteDistribution 3 19 49 60 47 32 18 3 3 1
    3 5 4

    % java DiscreteDistribution 2 9316001 10274874 10109130 10045436 9850199 6704495 5886889
    5 5

    % java DiscreteDistribution 1 8167 1492 2782 4253 12702 2228 2015 6094 6966 153 772     ...
    5

  ==> passed

  Test 2: check that output contains correct number of integers
    * fair die                          [ repeated 1000 times ]
    * sum of two dice                   [ repeated 1000 times ]
    * loaded die                        [ repeated 1000 times ]
    * fair coin                         [ repeated 1000 times ]
    * 80/20 biased coin                 [ repeated 1000 times ]
    * 9 digits in Benford's law         [ repeated 1000 times ]
    * goals in FIFA World Cup 1990-2002 [ repeated 1000 times ]
    * U.S. birthdays by day of week     [ repeated 1000 times ]
    * 26 letters in English language    [ repeated 1000 times ]
  ==> passed

  Test 3: check that output is a sequence of integers between 1 and n
    * fair die                          [ repeated 1000 times ]
    * sum of two dice                   [ repeated 1000 times ]
    * loaded die                        [ repeated 1000 times ]
    * fair coin                         [ repeated 1000 times ]
    * 80/20 biased coin                 [ repeated 1000 times ]
    * 9 digits in Benford's law         [ repeated 1000 times ]
    * goals in FIFA World Cup 1990-2002 [ repeated 1000 times ]
    * U.S. birthdays by day of week     [ repeated 1000 times ]
    * 26 letters in English language    [ repeated 1000 times ]
  ==> passed

  Test 4: check that program produces different results when run twice
    * fair die                          [ repeated 10 times ]
    * sum of two dice                   [ repeated 12 times ]
    * loaded die                        [ repeated 10 times ]
    * fair coin                         [ repeated 20 times ]
    * 80/20 biased coin                 [ repeated 30 times ]
    * 9 digits in Benford's law         [ repeated 10 times ]
    * goals in FIFA World Cup 1990-2002 [ repeated 10 times ]
    * U.S. birthdays by day of week     [ repeated 14 times ]
    * 26 letters in English language    [ repeated 10 times ]
  ==> passed

  Test 5: check randomness
    * fair die                          [ repeated 100000 times ]
      - command-line arguments = "100000 1 1 1 1 1 1"

           value  observed  expected   2*O*ln(O/E)
        -------------------------------------------
               1     19917   16666.7       7096.94
               2     19916   16666.7       7094.59
               3     20169   16666.7       7693.91
               4     20021   16666.7       7342.54
               5     19977   16666.7       7238.50
               6         0   16666.7          0.00
        -------------------------------------------
                    100000  100000.0      36466.48

      G-statistic = 36466.48 (p-value = 0.000000, reject if p-value <= 0.0001)
      Note: a correct solution will fail this test by bad luck 1 time in 10,000.

    * sum of two dice                   [ repeated 100000 times ]
      - command-line arguments = "100000 10 20 30 40 50 60 50 40 30 20 10"

           value  observed  expected   2*O*ln(O/E)
        -------------------------------------------
               1      2758    2777.8        -39.41
               2      5632    5555.6        153.94
               3      8271    8333.3       -124.20
               4     11128   11111.1         33.80
               5     13887   13888.9         -3.78
               6     16564   16666.7       -204.70
               7     13904   13888.9         30.24
               8     11212   11111.1        202.69
               9      8517    8333.3        371.35
              10      5650    5555.6        190.49
              11      2477    2777.8       -567.74
        -------------------------------------------
```

```
                     100000  100000.0         42.67

    G-statistic = 42.67 (p-value = 0.000006, reject if p-value <= 0.0001)
    Note: a correct solution will fail this test by bad luck 1 time in 10,000.

  * loaded die                         [ repeated 100000 times ]
  * fair coin                          [ repeated 100000 times ]
  * 80/20 biased coin                  [ repeated 100000 times ]
    - command-line arguments = "100000 80 20"

            value  observed  expected   2*O*ln(O/E)
        -------------------------------------------
                1     80904   80000.0       1818.18
                2     19096   20000.0      -1766.51
        -------------------------------------------
                     100000  100000.0         51.67

    G-statistic = 51.67 (p-value = 0.000000, reject if p-value <= 0.0001)
    Note: a correct solution will fail this test by bad luck 1 time in 10,000.

  * 9 digits in Benford's law          [ repeated 100000 times ]
  * 26 letters in English language     [ repeated 100000 times ]
  * goals in FIFA World Cup 1990-2002  [ repeated 100000 times ]
    - command-line arguments = "100000 19 49 60 47 32 18 3 3 1"

            value  observed  expected   2*O*ln(O/E)
        -------------------------------------------
                1      8261    8189.7        143.31
                2     21405   21120.7        572.43
                3     25862   25862.1         -0.14
                4     20351   20258.6        185.18
                5     13820   13793.1         53.85
                6      7644    7758.6       -227.54
                7      1346    1293.1        107.93
                8      1311    1293.1         36.04
                9         0     431.0          0.00
        -------------------------------------------
                     100000  100000.0        871.06

    G-statistic = 871.06 (p-value = 0.000000, reject if p-value <= 0.0001)
    Note: a correct solution will fail this test by bad luck 1 time in 10,000.

  * U.S. birthdays by day of week      [ repeated 100000 times ]
==> FAILED

Test 6: check randomness when n = 1
  * a_1 = 1                            [ repeated 100000 times ]
  * a_1 = 100                          [ repeated 100000 times ]
==> passed


DiscreteDistribution Total: 5/6 tests passed!


================================================================
Testing correctness of ThueMorse
*-----------------------------------------------------------
Running 5 total tests.

Test 1: check output format
  % java ThueMorse 2
  + -
  - +

  - line 0 of output in student solution:
    '+ - '
  - student solution (ignoring trailing whitespace) has 3 characters
  - it should have exactly 4 characters
  - it should start with either a '+' or '-' character
  - it should alternate between two space characters and either a '+' or '-' character

  % java ThueMorse 4
  + - - +
  - + + -
  - + + -
  + - - +

  - line 0 of output in student solution:
    '+ - - + '
  - student solution (ignoring trailing whitespace) has 7 characters
  - it should have exactly 10 characters
  - it should start with either a '+' or '-' character
  - it should alternate between two space characters and either a '+' or '-' character
```

```
    % java ThueMorse 8
    + - - + - + + -
    - + + - + - - +
    - + + - + - - +
    + - - + - + + -
    - + + - + - - +
    + - - + - + + -
    + - - + - + + -
    - + + - + - - +

    - line 0 of output in student solution:
      '+ - - + - + + - '
    - student solution (ignoring trailing whitespace) has 15 characters
    - it should have exactly 22 characters
    - it should start with either a '+' or '-' character
    - it should alternate between two space characters and either a '+' or '-' character

    % java ThueMorse 16
    + - - + - + + - - + + - + - - +
    - + + - + - - + + - - + - + + -
    - + + - + - - + + - - + - + + -
    + - - + - + + - - + + - + - - +
    - + + - + - - + + - - + - + + -
    + - - + - + + - - + + - + - - +
    + - - + - + + - - + + - + - - +
    - + + - + - - + + - - + - + + -
    - + + - + - - + + - - + - + + -
    + - - + - + + - - + + - + - - +
    + - - + - + + - - + + - + - - +
    - + + - + - - + + - - + - + + -
    + - - + - + + - - + + - + - - +
    - + + - + - - + + - - + - + + -
    - + + - + - - + + - - + - + + -
    + - - + - + + - - + + - + - - +

    - line 0 of output in student solution:
      '+ - - + - + + - - + + - + - - + '
    - student solution (ignoring trailing whitespace) has 31 characters
    - it should have exactly 46 characters
    - it should start with either a '+' or '-' character
    - it should alternate between two space characters and either a '+' or '-' character

  ==> FAILED

  Test 2: check correctness when n is a power of 2
    * java ThueMorse 2
    * java ThueMorse 4
    * java ThueMorse 8
    * java ThueMorse 16
    * java ThueMorse 32
    * java ThueMorse 64
  ==> passed

  Test 3: check correctness when n is not a power of 2
    * java ThueMorse 3
    * java ThueMorse 5
    * java ThueMorse 6
    * java ThueMorse 7
    * java ThueMorse 9
    * java ThueMorse 10
    * java ThueMorse 11
    * java ThueMorse 12
    * java ThueMorse 13
    * java ThueMorse 14
    * java ThueMorse 15
  ==> passed

  Test 4: check corner case
    * java ThueMorse 1
  ==> passed

  Test 5: check random values of n
    * 100 random values of n in [16, 32)
    * 100 random values of n in [32, 64)
    * 50 random values of n in [64, 128)
    * 25 random values of n in [128, 256)
  ==> passed


  ThueMorse Total: 4/5 tests passed!


  ================================================================
  Testing correctness of Birthday
```

```
*-----------------------------------------------------------
Running 6 total tests.

Test 1: check output format
  % java Birthday 365 100000
  1     0       0.0
  2     252     0.00252
  3     549     0.00801
  4     842     0.01643
  5     1038    0.02681
  6     1338    0.040190000000000003
  7     1582    0.056010000000000004
  8     1804    0.07405
  9     2066    0.09471
  10    2223    0.11694
  11    2399    0.14093
  12    2473    0.16566
  13    2754    0.1932
  14    2863    0.22183
  15    3007    0.2519
  16    3035    0.28225
  17    3144    0.31369
  18    3201    0.3457
  19    3260    0.3783
  20    3265    0.41095000000000004
  21    3245    0.4434
  22    3215    0.47555000000000003
  23    3195    0.5075000000000001

  % java Birthday 31 100000
  1     0       0.0
  2     3232    0.03232
  3     6247    0.09479
  4     8904    0.18383
  5     10596   0.28979
  6     11445   0.40424
  7     11473   0.5189699999999999

==> passed

Test 2: check values in first column
  * java Birthday 365 10000
  * java Birthday 31 10000
  * java Birthday 1 1000
  * java Birthday 2 1000
==> passed

Test 3: check that cumulative percentages are monotone nondecreasing
        and table stops when percentage reaches (or exceeds) 50%
  * java Birthday 365 10000 [ repeated 10 times ]
  * java Birthday 31 10000 [ repeated 10 times ]
  * java Birthday 10 5 [ repeated 1000 times ]
  * java Birthday 4 4 [ repeated 1000 times ]
  * java Birthday 2 2 [ repeated 1000 times ]
==> passed

Test 4: check that cumulative percentages are consistent with frequencies
  * java Birthday 365 10000
  * java Birthday 31 10000
==> passed

Test 5: check that each execution of program outputs a different table
  * java Birthday 365 10000 [ repeated twice ]
  * java Birthday 31 10000 [ repeated twice ]
==> passed

Test 6: check randomness of birthdays
  * java Birthday 365 1000000
  * java Birthday 31 1000000
  * java Birthday 7 1000000
  * java Birthday 5 1000000
==> passed


Birthday Total: 6/6 tests passed!


================================================================
Testing correctness of Minesweeper
*-----------------------------------------------------------
Running 11 total tests.

Test 1: check output format
  % java Minesweeper 9 9 10
```

```
    1  2  2  1  0  0  0  0  0
    1  *  *  1  0  0  0  0  0
    1  2  2  1  0  0  0  0  0
    1  1  1  0  1  1  1  1  1
    1  *  1  0  1  *  2  2  *
    1  1  1  0  2  3  *  2  1
    1  1  1  1  2  *  2  1  0
    *  1  1  *  2  2  2  1  0
    1  1  1  1  1  1  *  1  0

    % java Minesweeper 16 16 40
    *  3  *  *  2  2  *  *  1  0  0  0  0  1  *  *
    2  *  4  4  3  *  3  2  1  0  0  0  0  2  3  3
    1  2  *  2  *  2  1  0  0  0  0  1  1  3  *  2
    0  1  1  2  1  1  1  1  1  0  0  1  *  3  *  2
    0  0  0  0  1  1  2  *  1  0  0  1  1  2  1  1
    0  0  0  0  1  *  2  1  1  0  0  0  1  1  1  0
    0  0  0  0  1  1  1  0  0  0  0  0  1  *  1  0
    0  0  0  1  1  1  0  0  0  0  0  0  1  1  1  0
    0  0  0  1  *  2  2  2  2  2  2  1  0  0  0  0
    1  1  2  2  2  2  *  *  2  *  *  1  0  0  0  0
    1  *  2  *  1  1  2  2  2  2  2  1  0  0  0  0
    1  1  2  1  1  0  0  0  0  1  1  1  0  0  0  0
    1  1  1  0  0  1  1  1  1  2  *  1  0  0  0  0
    1  *  1  0  0  1  *  2  2  *  3  2  2  1  1  0
    1  1  1  0  0  1  2  *  3  3  4  *  3  *  2  0
    0  0  0  0  0  0  1  1  2  *  *  2  3  *  2  0

    % java Minesweeper 16 30 82
    *  1  0  0  0  0  0  0  0  0  1  1  1  1  *  2  2  2  1  0  0  0  0  1  1  1  1  1  1  0
    2  2  0  0  0  1  2  2  2  1  2  *  2  2  2  3  *  *  1  0  0  0  0  2  *  2  1  *  2  1
    *  2  1  1  0  1  *  *  2  *  3  2  2  *  1  2  *  3  1  1  2  2  1  2  *  3  2  1  2  *
    1  2  *  1  0  1  2  3  3  4  *  2  1  2  2  2  1  1  0  1  *  *  1  1  2  *  1  0  1  1
    0  1  2  3  2  1  0  1  *  3  *  2  0  1  *  1  0  0  0  1  2  2  1  0  2  3  3  1  0  0
    0  0  1  *  *  1  0  2  2  3  1  1  0  2  2  2  0  1  2  2  1  0  1  1  2  *  *  1  0  0
    1  1  1  2  3  2  2  2  *  1  0  0  0  1  *  2  1  1  *  *  1  0  1  *  2  2  2  1  0  0
    *  1  1  1  2  *  2  *  2  1  0  0  0  1  2  *  1  1  2  2  1  0  1  1  2  1  1  1  1  1
    1  1  2  *  3  1  2  1  2  1  2  1  1  1  3  4  3  2  1  1  0  0  0  0  1  *  3  3  *  2
    0  1  3  *  3  2  2  1  2  *  3  *  1  1  *  *  *  2  *  1  0  0  0  0  1  2  *  *  3  *
    0  2  *  3  2  *  *  3  3  *  3  2  2  2  2  4  3  3  1  1  0  0  0  0  0  1  2  2  2  1
    0  2  *  2  1  3  *  *  3  2  1  1  *  1  0  1  *  3  2  1  0  0  1  1  1  1  1  1  0  0
    0  1  1  2  1  2  2  3  *  1  0  1  2  3  2  2  2  *  *  1  0  0  1  *  1  1  *  1  0  0
    1  1  1  1  *  1  0  1  1  1  0  0  1  *  *  1  1  2  2  1  0  0  1  1  1  2  2  2  0  0
    1  *  1  1  2  2  1  0  1  1  1  0  2  3  4  2  1  0  0  0  1  1  1  0  0  2  *  2  1  1
    1  1  1  0  1  *  1  0  1  *  1  0  1  *  2  *  1  0  0  0  1  *  1  0  0  2  *  2  1  *

    % java Minesweeper 4 8 0
    0  0  0  0  0  0  0  0
    0  0  0  0  0  0  0  0
    0  0  0  0  0  0  0  0
    0  0  0  0  0  0  0  0

    % java Minesweeper 8 4 32
    *  *  3  1
    *  *  *  3
    3  *  *  *
    1  4  *  *
    2  4  *  4
    *  *  5  *
    *  6  *  *
    2  *  *  *

    % java Minesweeper 1 20 10
    0  1  *  *  1  1  *  1  0  0  1  *  *  *  *  1  0  1  *  *

  ==> passed

  Test 2: check that counts are consistent with mines (varying k)
    * m =  4, n =  8, k random [1000 trials]
    * m =  8, n =  4, k random [1000 trials]
    * m =  5, n = 40, k random [1000 trials]
    * m =  7, n = 30, k random [1000 trials]
    * m = 10, n = 10, k random [1000 trials]
  ==> passed

  Test 3: check that counts are consistent with mines (fixed k)
    * k =  1, m and n random   [1000 trials]
    * k = 10, m and n random   [1000 trials]
    * k = 20, m and n random   [1000 trials]
    * k = 50, m and n random   [1000 trials]
    * k = 80, m and n random   [1000 trials]
    * k = 90, m and n random   [1000 trials]
    * k = 99, m and n random   [1000 trials]
  ==> passed
```

```
Test 4: check that counts are consistent with mines (corner cases)
  * m =  5, n = 10, k =  0
  * m = 10, n =  5, k =  0
  * m =  5, n = 10, k = 50
  * m = 10, n =  5, k = 50
  * k =  0, m and n random   [1000 trials]
  * k =  1, m and n random   [1000 trials]
==> passed

Test 5: check that program produces different results each time
  * m =  4, n =  8, k = 16 [2 trials]
  * m =  8, n =  4, k = 26 [2 trials]
  * m =  1, n = 20, k = 16 [2 trials]
  * m = 20, n =  1, k = 10 [2 trials]
==> passed

Test 6: check number of mines, with k varying
  * m =  4, n =  8, k random [1000 trials]
    - m = 4, n = 8, k = 14
    - student  number of mines = 11
    - required number of mines = 14

    - failed on trial 1 of 1000

  * m =  8, n =  4, k random [1000 trials]
    - m = 8, n = 4, k = 16
    - student  number of mines = 12
    - required number of mines = 16

    - failed on trial 2 of 1000

  * m =  5, n = 40, k random [1000 trials]
    - m = 5, n = 40, k = 69
    - student  number of mines = 60
    - required number of mines = 69

    - failed on trial 1 of 1000

  * m =  7, n = 30, k random [1000 trials]
    - m = 7, n = 30, k = 184
    - student  number of mines = 118
    - required number of mines = 184

    - failed on trial 1 of 1000

  * m = 10, n = 10, k random [1000 trials]
    - m = 10, n = 10, k = 74
    - student  number of mines = 49
    - required number of mines = 74

    - failed on trial 2 of 1000

==> FAILED

Test 7: check number of mines, with k fixed
  * k =  5, m and n random   [1000 trials]
    - m = 3, n = 23, k = 5
    - student  number of mines = 4
    - required number of mines = 5

    - failed on trial 8 of 1000

  * k = 10, m and n random   [1000 trials]
    - m = 12, n = 7, k = 10
    - student  number of mines = 9
    - required number of mines = 10

    - failed on trial 1 of 1000

  * k = 50, m and n random   [1000 trials]
    - m = 8, n = 29, k = 50
    - student  number of mines = 46
    - required number of mines = 50

    - failed on trial 1 of 1000

  * k = 99, m and n random   [1000 trials]
    - m = 21, n = 21, k = 99
    - student  number of mines = 89
    - required number of mines = 99

    - failed on trial 1 of 1000
```

```
==> FAILED

Test 8: check number of mines for corner cases
  * m =  5, n = 20, k =  0
  * m = 20, n =  5, k =  0
  * m =  5, n = 10, k = 50
    - m = 5, n = 10, k = 50
    - student  number of mines = 34
    - required number of mines = 50

  * m = 10, n =  5, k = 50
    - m = 10, n = 5, k = 50
    - student  number of mines = 30
    - required number of mines = 50

  * k =  0, m and n random   [1000 trials]
  * k =  1, m and n random   [1000 trials]
==> FAILED

Test 9: check that mines are uniformly random
  * m = 1, n = 2, k = 1 [repeated 15000 times]
  * m = 1, n = 3, k = 1 [repeated 15000 times]
  * m = 2, n = 2, k = 2 [repeated 15000 times]
          value  observed  expected   2*O*ln(O/E)
        -------------------------------------------
            1-1      6670    7500.0      -1564.56
            1-2      6417    7500.0      -2001.49
            2-1      6544    7500.0      -1784.61
            2-2      6615    7500.0      -1661.20
        -------------------------------------------
                    26246   30000.0      -7011.86

    G-statistic = -7011.86 (p-value = 0.000000, reject if p-value <= 0.0001)
    Note: a correct solution will fail this test by bad luck 1 time in 10,000.

  * m = 2, n = 4, k = 3 [repeated 15000 times]
          value  observed  expected   2*O*ln(O/E)
        -------------------------------------------
            1-1      4901    5625.0      -1350.54
            1-2      4820    5625.0      -1488.87
            1-3      5009    5625.0      -1161.93
            1-4      4983    5625.0      -1207.77
            2-1      4983    5625.0      -1207.77
            2-2      4994    5625.0      -1188.41
            2-3      4984    5625.0      -1206.01
            2-4      4904    5625.0      -1345.36
        -------------------------------------------
                    39578   45000.0     -10156.66

    G-statistic = -10156.66 (p-value = 0.000000, reject if p-value <= 0.0001)
    Note: a correct solution will fail this test by bad luck 1 time in 10,000.

  * m = 3, n = 3, k = 6 [repeated 15000 times]
          value  observed  expected   2*O*ln(O/E)
        -------------------------------------------
            1-1      7611   10000.0      -4155.46
            1-2      7701   10000.0      -4023.54
            1-3      7575   10000.0      -4207.64
            2-1      7567   10000.0      -4219.18
            2-2      7603   10000.0      -4167.09
            2-3      7592   10000.0      -4183.04
            3-1      7627   10000.0      -4132.16
            3-2      7395   10000.0      -4463.34
            3-3      7684   10000.0      -4048.62
        -------------------------------------------
                    68355   90000.0     -37600.07

    G-statistic = -37600.07 (p-value = 0.000000, reject if p-value <= 0.0001)
    Note: a correct solution will fail this test by bad luck 1 time in 10,000.

==> FAILED

Test 10: check statistical independence of mines within an m-by-n grid
  * m = 500, n = 500, k = 125000
  * m = 500, n = 500, k = 25000
  * m = 500, n = 500, k = 225000
  * m = 100, n = 900, k = 27000
  * m = 900, n = 100, k = 63000
==> passed

Test 11: check statistical independence of mines between m-by-n grids
  * m = 1, n = 2, k = 1 [repeated 50000 times]
  * m = 1, n = 3, k = 1 [repeated 50000 times]
  * m = 2, n = 2, k = 2 [repeated 50000 times]
```

```
    * m = 2, n = 4, k = 3 [repeated 50000 times]
    * m = 3, n = 3, k = 8 [repeated 50000 times]
  ==> passed


  Minesweeper Total: 7/11 tests passed!


  ================================================================
```