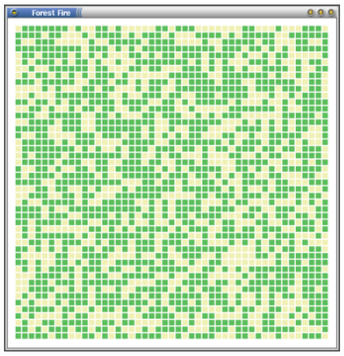
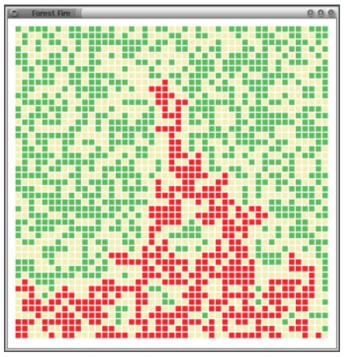
Forest Fire

You live in a beautiful mountain town called Onett in Colorado. Here is a map of one section of the forest just to the south of the little town, showing the trees in green and open spaces in tan:

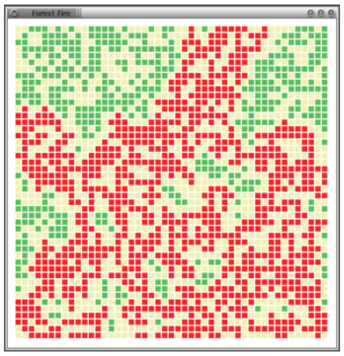


Occasionally in the summer, lightning strikes in the mountains start forest fires. The fire spreads from tree to tree; if a fire anywhere along the south side of this section of trees would make it to the northern border (where the town is), Onett will be in trouble.

Some forest configurations are too sparse, and the fire will not make it to the town:



Other forest configurations do provide a path for the fire to reach the northern edge of the forest:



If this is the case, the fire-fighters would like to know where to put their resources, so a program that maps the spread of the fire would be useful.

The Assignment. Given a "forest" with a 60% probability of having a tree in any location, write a program to determine if the fire will have a path from the bottom edge to the top edge. Fires can only go from tree to tree horizontally or vertically. If there is a path, display it in red, including all trees that would be destroyed.

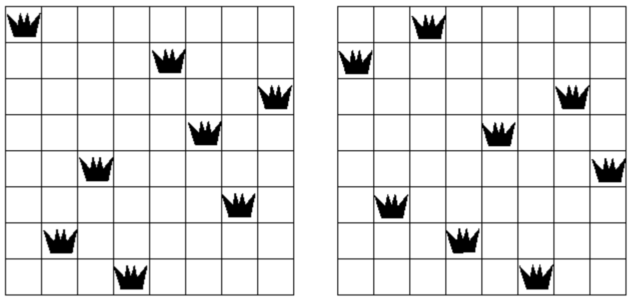
Use recursion to set adjacent trees on fire, starting with every tree on the southern border. When the recursion is complete, display the fire map. If a tree on the northern border is reported as on fire, print a message akin to "Onett is in trouble!", otherwise print "Onett is safe."

You are given the following files, which follow the Model-View-Controller (MVC) pattern. Most of your code will go in the **FireModel** class, though you may need to modify the **FireCell** class as well.

Fire.java	Controller (runner), contains a `main` method
FireCell.java	Individual location in a grid; could be dirt, a tree, or a burning tree
FireModel.java	Fire's path model class, contains the bulk of the logic
FireView.java	Contains a graphical view; displays fire map.

(Advanced) Eight Queens

The eight queens problem attempts to place eight chess queens on an 8×8 board such that none of them attack one another (no two are in the same row, column, or diagonal). A couple solutions for an 8x8 board:



Hints:

- An abstraction of the board which handles much of the logic will simplify the recursive code.
- You can output the results to the console, but the program is more interesting with graphics. Build your own GUI (or use Greenfoot). If you get stuck, here is an example. Your Board class would need to be consistent with what the GUI expects.

(Advanced) Knight's Tour

A ***knight's tour*** is a series of moves by a knight on a chess board such that the knight visits every square on the board once (moving per the rules of chess). Wikipedia has a nice visualization of a knight's tour here.

Write a program that will print a knight's tour for an `n` x `n` chess board. Your program should have the following:

- `int[][] board` – the chess board (all positions should initially start at 0)
- `int size` – size of the chess board, i.e. `n` x `n`
- `void solve()` – method that will print the knight's tour, starting at index location [0, 0]
- an overridden `toString` method for printing the tour

An example of a knight's tour for an 8 x 8 board, starting at [0, 0]:

```
1       60      39      34      31      18      9       64
38      35      32      61      10      63      30      17
59      2       37      40      33      28      19      8
36      49      42      27      62      11      16      29
43      58      3       50      41      24      7       20
48      51      46      55      26      21      12      15
57      44      53      4       23      14      25      6
52      47      56      45      54      5       22      13
```

Once you have it working, modify your code such that you can find a solution starting from any [row, col]. Example of a knight's tour of a 5 x 5 board, starting at [0, 2]:

```
25      14      1       8       19
4       9       18      13      2
15      24      3       20      7
10      5       22      17      12
23      16      11      6       21 //note that not all starting positions have solutions
```