File - /Users/moe/Downloads/cs61b/fa20-s1319/hw2/hw2/Percolation.java

```java
package hw2;
import static org.junit.Assert.*;
import edu.princeton.cs.algs4.WeightedQuickUnionUF;

public class Percolation {
    private boolean[][] grid;
    private int openCount;
    private WeightedQuickUnionUF gridUnion;
    private int size;
    private int top;
    private int bottom;
    private WeightedQuickUnionUF everythingButBottom;


    public Percolation(int N) { // create N-by-N grid, with all sites initially blocked
        if (N <= 0) {
            throw new IllegalArgumentException();
        }
        grid = new boolean[N][N];
        for (int x = 0; x < N; x++) {
            for (int y = 0; y < N; y++) {
                grid[x][y] = false;
            }
        }
        openCount = 0;
        gridUnion = new WeightedQuickUnionUF(N * N + 2);
        size = N;
        top = N * N;
        bottom = N * N + 1;
        everythingButBottom = new WeightedQuickUnionUF(bottom);
    }

    private int xyTo1D(int r, int c) { //takes in a x value and y value then returns the grid number
        return r * size + c;
    }

    private void validate(int r, int c) {
        if (r < 0 || c < 0 || r > size - 1 || c > size - 1) {
            throw new IndexOutOfBoundsException();
        }
    }

    public void open(int row, int col) { // open the site (row, col) if it is not open already
        validate(row, col);
        if (!isOpen(row, col)) {
            grid[row][col] = true;
            openCount++;
            if (col < size - 1 && isOpen(row, col + 1)) {
                gridUnion.union(xyTo1D(row, col), xyTo1D(row, col + 1));
                everythingButBottom.union(xyTo1D(row, col), xyTo1D(row, col + 1));
            }
            if (col > 0 && isOpen(row, col - 1)) {
                gridUnion.union(xyTo1D(row, col), xyTo1D(row, col - 1));
```

```
54                  everythingButBottom.union(xyTo1D(row, col), xyTo1D(row, col - 1));
55              }
56              if (row < size - 1 && isOpen(row + 1, col)) {
57                  gridUnion.union(xyTo1D(row + 1, col), xyTo1D(row, col));
58                  everythingButBottom.union(xyTo1D(row + 1, col), xyTo1D(row, col));
59              }
60              if (row > 0 && isOpen(row - 1, col)) {
61                  gridUnion.union(xyTo1D(row - 1, col), xyTo1D(row, col));
62                  everythingButBottom.union(xyTo1D(row - 1, col), xyTo1D(row, col));
63              }
64              if (row == 0) {
65                  gridUnion.union(xyTo1D(row, col), top);
66                  //gridUnion.union(xyTo1D(row, col), 0);
67                  everythingButBottom.union(xyTo1D(row, col), top);
68              }
69              if (row == size - 1) {
70                  gridUnion.union(xyTo1D(row, col), bottom);
71                  //gridUnion.union(xyTo1D(row, col), 0);
72              }
73          }
74      }
75
76      public boolean isOpen(int row, int col) { // is the site (row, col) open?
77          validate(row, col);
78          return grid[row][col];
79      }
80
81      public boolean isFull(int row, int col) { // is the site (row, col) full?
82          validate(row, col);
83          return everythingButBottom.connected(top, xyTo1D(row, col));
84          /*for (int i = 0; i <= size; i++) {
85              if (gridUnion.connected(i, xyTo1D(row, col))) {
86                  return true;
87              }
88          }
89          return false;*/
90      }
91
92      public int numberOfOpenSites() {   // number of open sites
93          return openCount;
94      }
95
96      public boolean percolates() {       // does the system percolate?
97          return gridUnion.connected(top, bottom);
98      }
99
100     public static void main(String[] args) {
101         Percolation table = new Percolation(6);
102         table.numberOfOpenSites();
103         table.xyTo1D(0, 0);
104         table.xyTo1D(0, 1);
105         table.xyTo1D(1, 0);
106         table.xyTo1D(1, 1);
```

```
107         table.xyTo1D(0, 2);
108         table.xyTo1D(2, 0);
109         table.xyTo1D(2, 2);
110         assertFalse(table.isOpen(1, 1));
111         assertFalse(table.isOpen(1, 2));
112         assertFalse(table.isOpen(2, 3));
113         table.open(1, 1);
114         table.open(1, 2);
115         table.open(2, 3);
116         table.open(2, 3);
117         table.open(0, 1);
118         assertTrue(table.isOpen(1, 1));
119         assertTrue(table.isOpen(1, 2));
120         assertTrue(table.isOpen(2, 3));
121         assertTrue(table.isOpen(0, 1));
122         assertTrue(table.isFull(1, 1));
123         assertTrue(table.isFull(1, 2));
124         assertFalse(table.percolates());
125         table.open(2, 2);
126         table.open(3, 2);
127         table.open(4, 2);
128         table.open(5, 2);
129         assertTrue(table.percolates());
130     }
131 }
132
```