Self-Avoiding Random Walk

Introduction to Programming in Java by Robert Sedgewick and Kevin Wayne provides on page 109 the following program:

```
public class SelfAvoidingWalk
{
    public static void main(String[] args)
    {    // Do T random self-avoiding walks in an N-by-N lattice
        int N = Integer.parseInt(args[0]);
        int T = Integer.parseInt(args[1]);
        int deadEnds = 0;
        for (int t = 0; t < T; t++)
        {
            boolean[][] a = new boolean[N][N];
            int x = N/2, y = N/2;
            while (x > 0 && x < N-1 && y > 0 && y < N-1)
            { // Check for dead end and make a random move.
                a[x][y] = true;
                if (a[x-1][y] && a[x+1][y] && a[x][y-1] && a[x][y+1])
                { deadEnds++; break; }
                double r = Math.random();
                if      (r < 0.25) { if (!a[x+1][y]) x++; }
                else if (r < 0.50) { if (!a[x-1][y]) x-; }
                else if (r < 0.75) { if (!a[x][y+1]) y++; }
                else if (r < 1.00) { if (!a[x][y-1]) y-; }
            }
        }
        System.out.println(100*deadEnds/T + "% dead ends");
    }
}
```

Do one of the following exercises. Everyone should (ideally) select a different exercise.

Exercise 1. (Ex. 1.4.18 and 1.4.19 in the book.) Modify `SelfAvoidingWalk` to calculate and print the average length of the paths and the average area of the smallest axis-oriented rectangle that encloses the path. Keep separate the average lengths of escape paths and dead-end paths.

Exercise 2. Extend `SelfAvoidingWalk` to print, on the standard output, cell visit counts, using "ASCII art": if any cell was visited by maximally K random walks, divide the range 0..K into several intervals and represent them by characters of increasing "density", for example `.`, `o`, `x`, `#` (and a space for unvisited cell). Remember not to count a single cell visit multiple times. Try to avoid using conditional statements when computing the character code (e.g.., `o`, `#`) from the number of visits, use indexing into an array of codes instead.

Exercise 3. (Ex. 1.4.31 in the book.) Self-avoiding walk length. Suppose that there is no limit on the size of the grid. Run experiments to estimate the average walk length. (Rather than using a large fixed lattice size, increase the size when it turns out not to be sufficient.)

Exercise 4. (Ex. 1.4.32 in the book.) Three-dimensional self-avoiding walks. Run experiments to verify that the dead-end probability is 0 for a three-dimensional self-avoiding walk and to compute the average walk length for various values of N.

Exercise 5. (Ex. 1.4.33 in the book.) Random walkers. Suppose that N random walkers, starting in the center of an N-by-N grid, move one step at a time, choosing to go left, right, up, or down with equal probability at each step. (They are not "self-avoiding".) Write a program to help formulate and test a hypothesis about the number of steps taken before all cells are touched.