Maze Jam

Maze Generation Overview

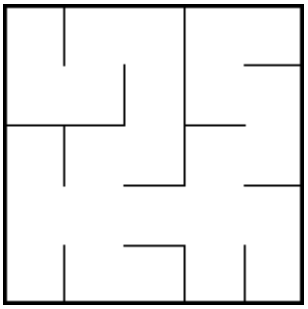
The goal is to produce a two-dimensional rectangular maze of arbitrary size. The actual dimensions of the maze should be accepted parametrically (ex: inputs to a function, command-line arguments, et cetera). The maze itself should be output to JSON as an array of arrays of integers, where each inner array represents one row of the maze and the number in each cell is a bitmask indicating the number of passages leading out of the cell (Note: the bitmask pattern is: North = 1, South = 2, East = 4, West = 8). So, for instance, a (pseudo) function call like

```
buildMaze(5,5)
```

might yield an output of

```
[
  [ 2, 14, 10, 14,  8 ],
  [ 5,  9, 11, 13, 11 ],
  [ 3, 15,  9, 15,  9 ],
  [ 7, 15, 13, 15, 11 ],
  [ 1, 13,  9,  9,  9 ]
]
```

which might be visualized as



In order to simplify this task, a simple web page has been provided, at `http://server:port/index.html` which will allow you to visualize any generated maze. Additionally, there is a service, at `http://server:port/mazes/render`, which will render an image of any generated maze. It accepts POST requests with the maze (in the JSON format described previously) and an integer cell size supplied in the request body. For example, requesting

```
POST http://server:port/mazes/render
```

with a body of

```
maze=[[2,14,10,14,8],[5,9,11,13,11],[3,15,9,15,9],[7,15,13,15,11],[1,13,9,9,9
]]&cellSize=30
```

Returns the image seen previously. The reverse side of this document describes two common algorithms for maze generation. Please feel free to implement one (or both) of these and/or any other algorithm you'd like. Please note, one of the two algorithms is a "wall-builder" instead of a "passage-carver". As the described format is for a "carved" maze, your output may have to be translated. The web page mentioned previously provides a command for this translation. However the rendering service does not. (Hint: the logic for conversion is very simple. You may want to port it to which ever language you are using during the jam.) As a bonus, you may also code up one, or more, maze solvers to apply to your generated output.

Maze Generation Algorithms

Growing Tree

This passage-carving algorithm maintains a list of "carved" cells as it traverses the maze. The general approach is:

1. choose a cell from this list
2. select one of its uncarved neighbors
3. carve that neighbor
4. add the newly carved cell to the list
5. repeat from step 1

If a cell has no uncarved neighbors (at step 2), it is removed from the list and another cell is chosen return to step 1). The maze is finished when there are no cells left in the list. To start the process, simply pick a cell at random from the (initial) block of uncarved cells (i.e. the maze). This algorithm can simulate several other algorithms and/or produce very interesting results by varying how the next cell is chosen from the list (in step 1). Some strategies include: most-recently added, oldest, or random. One can even blend strategies. (Hint: you may want to accept choice of strategy as an input parameter.)

Recursive Division

This wall-building algorithm starts with the maze as an empty, but bounded, area. It then proceeds as follows:

1. Pick a location at random to build a wall running either the height (vertically) or width (horizontally) of the bounded area (maze)
2. Place an opening randomly within the new wall
3. Recursively repeat from step 1 on each of the two subdivisions created by building the wall (e.g. step 1, but now the bounded area is a subset of the overall maze)
4. Halt the process when an (arbitrary) number of subdivisions has been reached

The orientation of the wall should be biased towards the proportions of a given (subdivision) area. In other words, an area where the height is twice the width should be divided horizontally more often than vertically (and vice-versa for inverse ratios). Please note, this algorithm "builds walls", but the output format should describe passages. Therefore, an extra conversion step may be warranted. (Hint: have a peek at the `builtToCarved(built)` JavaScript function in the `index.js` file in the provided utilities.)