The Maze Game

Harshil Solanki

Contents

1 Lorem

See [1]

2 Maze Generation Algorithms

2.1 Krsukal's Algorithm

[Source]
Kruskal's algorithm is a method for producing a minimal spanning tree from a weighted graph. It works something like this:

1. Throw all of the edges in the graph into a big burlap sack. (Or, you know, a set or something.)
2. Pull out the edge with the lowest weight. If the edge connects two disjoint trees, join the trees. Otherwise, throw that edge away.
3. Repeat until there are no more edges left.

The *Randomized Kruskal's algorithm* just changes the second step, so that instead of pulling out the edge with the lowest weight, you remove an edge from the bag at random. Making that change, the algorithm now produces a fairly convincing maze.

2.2 Prism's Algorithm

The standard version of the algorithm works something like this:

1. Choose an arbitrary vertex from G (the graph), and add it to some (initially empty) set V.
2. Choose the edge with the smallest weight from G, that connects a vertex in V with another vertex not in V.
3. Add that edge to the minimal spanning tree, and the edge's other vertex to V.
4. Repeat steps 2 and 3 until V includes every vertex in G.

And the result is a minimal spanning tree of G. For maze generation, in the second step, we select a random edge instead of edge with lowest weight.

2.3 Recursive Backtracing

Here's the mile-high view of recursive backtracking:

1. Choose a starting point in the field.

2. Randomly choose a wall at that point and carve a passage through to the adjacent cell, but only if the adjacent cell has not been visited yet. This becomes the new current cell.

3. If all adjacent cells have been visited, back up to the last cell that has uncarved walls and repeat.

4. The algorithm ends when the process has backed all the way up to the starting point.

Seems simple enough.

2.4 Aldous-Broder Algorithm

Aldous and Broder were researching these uniform spanning trees, and independently arrived at the following algorithm:

1. Choose a vertex. Any vertex.

2. Choose a connected neighbor of the vertex and travel to it. If the neighbor has not yet been visited, add the traveled edge to the spanning tree.

3. Repeat step 2 until all vertexes have been visited.

Note: this algorithm is notable in that it selects from all possible spanning trees (i.e. mazes) of a given graph (i.e. field) with equal probability. The other algorithms shown don't have this property.

2.5 Wilson's Algorithm

Note: a spanning tree is a tree that connects all the vertices of a graph. A uniform spanning tree (UST) is any one of the possible spanning trees of a graph, selected randomly and with equal probability. The algorithm goes something like this:

1. Choose any vertex at random and add it to the UST.

2. Select any vertex that is not already in the UST and perform a random walk until you encounter a vertex that is in the UST.

3. Add the vertices and edges touched in the random walk to the UST.

4. Repeat 2 and 3 until all vertices have been added to the UST.

So, it's still doing the random walk, but this algorithm converges much more rapidly than Aldous-Broder.

2.6 Hunt and Kill Algorithm

Toda's algorithm is the "hunt-and-kill algorithm". Sounds violent, doesn't it? It's actually quite tame. In a nutshell, it works like this:

1. Choose a starting location.
2. Perform a random walk, carving passages to unvisited neighbors, until the current cell has no unvisited neighbors.
3. Enter "hunt" mode, where you scan the grid looking for an unvisited cell that is adjacent to a visited cell. If found, carve a passage between the two and let the formerly unvisited cell be the new starting location.
4. Repeat steps 2 and 3 until the hunt mode scans the entire grid and finds no unvisited cells.

2.7 Growing Tree Algorithm

A slick algorithm. Here's how it works:

1. Let C be a list of cells, initially empty. Add one cell to C, at random.
2. Choose a cell from C, and carve a passage to any unvisited neighbor of that cell, adding that neighbor to C as well. If there are no unvisited neighbors, remove the cell from C.
3. Repeat step 2 until C is empty.

Pretty straight-forward, really. But the fun lies in how you choose the cells from C, in step 2. If you always choose the newest cell (the one most recently added), you'll get the recursive backtracker. If you always choose a cell at random, you get Prim's. It's remarkably fun to experiment with other ways to choose cells from C.

2.8 Eller's Algorithm

It does this by building the maze one row at a time, using sets to keep track of which columns are ultimately connected. But it never needs to look at more than a single row, and when it finishes, it always produces a perfect maze.

Like the recursive backtracking algorithm, here's the "mile-high" overview of Eller's algorithm:

1. Initialize the cells of the first row to each exist in their own set.
2. Now, randomly join adjacent cells, but only if they are not in the same set. When joining adjacent cells, merge the cells of both sets into a single set, indicating that all cells in both sets are now connected (there is a path that connects any two cells in the set).

3. For each set, randomly create vertical connections downward to the next row. Each remaining set must have at least one vertical connection. The cells in the next row thus connected must share the set of the cell above them.
4. Flesh out the next row by putting any remaining cells into their own sets.
5. Repeat until the last row is reached.
6. For the last row, join all adjacent cells that do not share a set, and omit the vertical connections, and you're done!

My Algorithm

Taking inspiration from the previous algorithms, I'm going to implement my algorithm in this way:

1. Make a random path from start-point to the end-point.
2. Choose two arbitrary points in this path and block the initial path we made between them and generate another random path.
3. Repeat the previous step according to the complexity of maze.
4. Use Hunt and Kill algorithm to verify no cell is univisited.

References

[1] Eric Matthes. *Python Crash Course*. William Pollock, 2016.