Oplossingen:

Randomized depth-first search:  
kiest random een cell naast de current cell en haalt de muur tussen die twee cellen weg, dan markeerd hij de cell waar hij geweest is als checked en voegt ie het toe aan een lijst genaamd the stack voor het backtracken later. Wanneer een cell geen unvisited neighbours heeft backtracked hij tot hij er 1 vind met een unvisited neighbour. Dit doet hij totdat elke cell bezocht is als dat is gebeurt backtracked hij naar het begin en dan weten we zeker dat hij klaar is.

Dit algroritme zorgd ervoor dat het een maze is met weinig junctions, en vooral 1 lange lijn. Voordat hij gaat backtracken

As given above this algorithm involves deep recursion which may cause stack overflow issues on some computer architectures. The algorithm can be rearranged into a loop by storing backtracking information in the maze itself. This also provides a quick way to display a solution, by starting at any given point and backtracking to the beginning.

The depth-first search algorithm of maze generation is frequently implemented using backtracking. This can be described with a following recursive routine:

Given a current cell as a parameter

Mark the current cell as visited

While the current cell has any unvisited neighbour cells

Choose one of the unvisited neighbours

Remove the wall between the current cell and the chosen cell

Invoke the routine recursively for the chosen cell

Randomized prim’s algorithm:

This algorithm is a randomized version of Prim's algorithm.

Start with a grid full of walls.

Pick a cell, mark it as part of the maze. Add the walls of the cell to the wall list.

While there are walls in the list:

Pick a random wall from the list. If only one of the cells that the wall divides is visited, then:

Make the wall a passage and mark the unvisited cell as part of the maze.

Add the neighboring walls of the cell to the wall list.

Remove the wall from the list.

Note that simply running classical Prim's on a graph with random edge weights would create mazes stylistically identical to Kruskal's, because they are both minimal spanning tree algorithms. Instead, this algorithm introduces stylistic variation because the edges closer to the starting point have a lower effective weight.

Uiteindelijk kies ik voor RDFS omdat het een simpel algoritme is waar weinig fout kan gaat om het goede resultaat te krijgen

Sources:

<https://en.wikipedia.org/wiki/Maze_generation_algorithm>