A3.6 Maze Generation

Wouter Slegers — Uppsala University — 950825-1957
Report — December 22, 2020 — wjslegers@gmail.com

Introduction

I have elected to do A3.6 and make a maze generation algorithm. To make writing code as easy as possible I chose Python, as I find it quick and easy to set things up. I made a 'Maze' class and print it using the 'print()' function of Python to simplify the creating of the graphical part of the program. You can find all the Python files in the zip-folder.

1. Path Through Maze Algorithm

The idea of the algorithm is to generate a random relatively dense maze. For example, for each wall there is a 70% chance it becomes a wall and 30% it is left open. Then we use a search algorithm that saves all the nodes it can reach from the starting node in a list called 'searched'. Every one of those nodes that is a dead end it also saves in a list called $'dead_ends'$, a dead end we define as a node with three or more walls around it. If we come across the end node we stop the algorithm. Otherwise we go through the dead ends until we find one that has a wall that leads to a previously unexplored node i.e. the neighbouring node is not in 'searched'. We remove the wall and the algorithm repeats itself, but starting from the node that we just connected to and we continue to use and add to the lists 'searched' and $'dead_ends'$ so that we don't have to do the same work many times. We, of course, remove the old dead end that we opened up from the list $'dead_ends'$.

Notes on the algorithm. The choice of dead end to check first for

Notes: choice of dead end, neighbour TODO: *rondlists*

1.1. Why it works.

2. Self-avoiding random walk

2.1. Why it works. ———————— **Features** ————————