Generating and Displaying Mazes in Two, Three, and Four Dimensions

Robert Kaufman
rbkaufm2

October 2015

Abstract

Mazes are a sort of puzzle in which the objective is navigate some space from a starting point to an ending location. The challenge arises from the layout of the seemingly random walls which restrict what paths may be taken. The most common examples of mazes are two dimensional but in this project, part of the goal is to extrapolate the concept of a maze into three and four dimensions. The second goal of this project is to experiment with different algorithms to automatically generate the maze with each providing slightly different visual styles and solution complexity.

1 Introduction

While mazes may be designed by hand, it is faster and easier to do so with a computer program, especially if the maze goes beyond 2D. This project will be showing how different algorithms develop a maze each with their own pattern and difficulty as well as how they translate into higher dimensions. Since the majority of the algorithms simply have different methods of selecting a random cell to open to, the translation of each of the four selected algorithms to the higher dimensions will not be too challenging. Another goal is for it to also provide an effective user interface and display system for a user to try and attempt the 2D, 3D and 4D generated mazes. Given the nature of mazes and the importance to see the large picture, this task is the more difficult of the two components of this project. With 3D and even more so with 4D mazes, it becomes difficult if not impossible to visualize the whole maze at once in order to try and see the one correct path to the end.

2 Background: Generation Methods

While there are many different algorithms explored and developed at this point by a variety of individuals through the year, four have been selected for their variety in generation. Each algorithm, though designed for 2D mazes, can and will be modified to work with mazes of higher dimension.

2.1 Recursive Division

The recursive division algorithm[1] tends to produce intersections with many paths connected resulting in an open and simple to solve maze. This algorithm works by randomly splitting the whole space into two parts with a wall and then randomly picks a hole for that wall. The two new sections undergo the same process until the maze is complete.

2.2 Recursive Back-tracker

The recursive back-tracker algorithm[1] generates few dead-ends meaning it contains longer passages. It works by randomly generating a path until it hits a dead end (it is at the edge of the space or another move would connect it to a previous path). Once it does, the algorithm will back track along the path it just made until it encounters a space with another option to move in which it will make another random decision and continue down that path, performing the same backtrack algorithm.

2.3 Prim's Algorithm

Prims algorithm[1] uses a minimum spanning tree algorithm for a graph except each node is chosen at random instead of by weight. In practice, this algorithm works by selecting a random location from the space and creating a list of available neighbors. The method then chooses one of the neighbors at random and extends the path to it, also adding the new location's neighbors to the list of possible new locations. It continues this until the maze is completed.

[1] https://en.wikipedia.org/wiki/Maze_generation_algorithm

2.4 Growing Tree

Finally, the growing tree algorithm[2] works by choosing a random cell and adding it to a list. It continues to randomly grow the path until it reaches a dead end in which by some predetermined method, it will pick another cell from the list to extend from again. If a cell is deemed a dead end, it is removed from the list as well. This continues until the maze is completed. This method is malleable in its generation reliant on the method used to select the new location to extend from. If the newest added location is selected, it acts like a recursive backtracker and if the selection is random, the result maze is similar to that of one generated from Prims algorithm

3 Goals and Methods

3.1 Maze Generation

My goal with the maze generation is successfully implement each of the four above algorithms for 2D, 3D, and 4D.

This will be accomplished by using functions in javascript with the walls of each maze being stored in an appropriately sized 1D matrix representation of the space that the maze is in (be it 2D, 3D, or 4D). The wall array will start out full and then each function will remove walls according to their generation algorithm.

3.2 Displaying the Maze

The goal of the display is to give an intuitive user interface and visual of each dimension of maze with a combination of key stroke or mouse movement to navigate the maze.

The accomplish this goal, the project will be coded using javascript and the three.js and WebGL libraries for it.

Any 2D mazes will be just a simple 2D rendering of a series of lines that will make up the maze. For larger mazes and better visibility, options will be provided to the

[2]http://weblog.jamisbuck.org/2011/2/7/maze-generation-algorithm-recap

user to allow them to zoom in or out of the maze, pan across the maze, and rotate the maze 90 degrees at a time. Some 2D visual representing the current location of the user will be present and four movement keys will be assigned as the control mechanism.

The 3D mazes will be generated and rendered as a partially visible box. The user will have the options to move the camera in all three axis of direction as well as to rotate the camera. A set distance (possibly editable by the user) will be assigned as the focus point in which the plane that is most directly facing the user camera and closest to that distance will receive full opacity while any obstructing part of the maze between that 2D maze plane will be translucent. There will six movement keys assigned for the user to navigate the maze.

The 4D mazes can be one displayed in one of two ways. One way is to display it as a series of separate 3D mazes with jumps between each other present. Another ways is to render it as a hypercube. While neither method makes the solution to the maze particularly simple to examine, the hypercube version may be especially hard to visualize. Like with the other views, adequate controls will be provided to rotate and pan the camera in all relevant axis as well as the move the user-maker through each dimension of the maze.