Robert Kaufman
10/04/2015

MATH 198 Final Project Pre-Proposal

General Concept:

For my project, I would like to randomly generate and display a solvable, multi-dimensional maze. There would be options for at the very least, a two-dimensional, three-dimensional, and four-dimensional maze that would be then be rendered in such a way as to allow a user to easily gather information to attempt to solve the maze.

Generation Algorithms:

The generation of the maze will have a set of options including the generation algorithm as well as parameters for the algorithm. Since different algorithms have certain biases in its generation, this allows for slightly different styles of mazes to be created. Some algorithms that are common to 2D maze generation that will be implemented and extended to the 3D and 4D mazes are the recursive division algorithm, recursive backtracker algorithm, Prim's algorithm, and a modifiable growing tree algorithm.

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.

The recursive backtracker 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

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

backtrack once that branch hits a dead end. This continues until the path reaches back the beginning which represents that all spaces have reached a dead end.

Prim's 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.

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 Prim's algorithm.

Graphical Display:

The project will be coded using javascript and the three.js and WebGL libraries for it. The graphical display of the maze is going to be primary application of math in the transformations of 3D and 4D to a function 2D display.

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 user to allow them to zoom in or out of the maze, pan across the maze, and rotate the maze 90 degrees at a

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

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.