Samuel Bogdan

Sebastian Neumayer

CSCE A211

19 September 2024

**Question:** Given a random start location, think of an algorithm that does not use recursion that would find its way to the exit S. Your algorithm should be able to run on an arbitrary maze, and not be specific to the sample given above. Can your algorithm avoid going in circles? Describe your algorithm

For writing my response to this problem. I am assuming that the maze is a 2d boolean array where true values, are visitable, and false values are “walls.” To even start the maze you are given the starting coordinates, and the dimensions of the array (otherwise you will have to be able to calculate the finish line of the maze). That way, your program starts at the given coordinates, then stops when it encounters a true-statement coordinate on the bounds of the array.

Since I can’t use any recursion, that makes my solution harder (in my minds-eye at least).

Navigation will be written from the perspective of the “maze-participant.” Imagine you are in the maze trying to find you way through.

Using a 2-d vector (A vector of structs could and should be used here):

* track coordinates traveled
* mark tiles that have been traveled (Tertiary Value: not traveled, traveled, traveled now blocked-> dead-end),
* direction traveled
* how many possible directions it can travel from tile

When navigating the maze, the program will see if it can move left, then right, or forward, all possible directions are counted before the program moves tiles. (Up, Down, Straight/Back). The program will chooe directions based on the preference: Left, Straight, Right. To prevent the program from repeatingingly slamming into dead ends, if the only possible direction is the direction it just came from, it will move back one tile, then modify the “dead-end” variable of the vector struct. Preventing the program from going where it just traversed until it encounters a tile space with more than one possible directions to travel. Since the previous direction is now blocked off, it will adopt the other “lane”. This is tricky and would require some implementation to ensure you do not manufacture your own dead-ends, Or go continuously in a circle. To avoid going in a circle, you can check if you have already been on this tile, and if there are any alternate paths that can be followed. Then moving the path that has not been traversed down. This would implementation and testing though, I can’t visualize this process better without testing and being able to see how my program performs.

The end goal of the program is to find the true tile on the outer bounds of the maze array.

**Question:** Think of a way to have a computer program generate a random maze. What are important criteria that a good maze should have? Describe how your algorithm could possibly meet the criteria.

Criteria for a maze. The algorithm should be scalable, and able to be created with almost any arbitrary 2-D array size. I think one of the best checks would be to have a programmed circle within the maze. It can be quite simple, too. Having an outer bounds, then a ring of true-tiles with one entrance into the real portion of the maze. I can see a few untested maze-solver algorithms failing to navigate deeper in the maze and continuously move around in circles. The maze should also have dead-ends of course, it is a maze after all.

Idea of maze generator. I have a harder time expressing how I would create this into words. Let's say I have a 25x25 grid, all tiles pre-intialized to false. Using a random number generator, I will create a “chiseling” program, to randomly navigate the array in directions until reaches bounds of the array. Now I have my 25x25 array with the “true-maze” pathing. Then every X tiles I would use the chiseling program to make an off-shoot dead end. Then with those off-shoot program, every Y tiles I can use the chiseling program to make more offshoots. Making sure to not poke holes in the array boundaries to make more than one finish.

I imagine the random number generation will be along the lines of: x = tiles to chisel, y = direction, z = #offshoots(testing required here). With the chiseling program repeating for a certain number of iterations before the maze is considered complete. Of course, testing would be required here. The last thing the maze should do is chisel out every tile available.

Past this point, I would need to implement the program to really pick a specific direction.

I have another idea of implementation, with having pre-fabricated maze nodes: a section of zig-zags, a section of loops, etc., then having a program randomly connect those into the maze-grid, and ensuring they are connected to each other. This pre-fab idea seems like it could make more “balanced” mazes, but they would be easier to solve. Of course, as you go deeper and deeper with complexity, you can really mix random-number-generation and tile layout, to better balance the maze. At this point though I would need to implement portions to better understand how a maze generator would work.