Solving Mazes using Image Processing techniques and Basic Graph Theory

Neeraj Jadhav

September 2021

Abstract: *Mazes are quite common in day-to-day lives. Commuting between two points on a map is a basic example path finding techniques and application of Graph Theory. The difference in the two situations is availability of complete information of the scenario prior to travelling in case we have a map. This can be solved by the popular Watershed algorithm. We limit our discussion to the other situation.*
This paper aims to solve mazes using a randomised graph traversal algorithm keeping track of important checkpoints. It may not be the best possible solution but gives the correct result with any kind of maze.

Keywords: DFS, Thresholding, Dead-ends.

1 Introduction

First of all, the problem faced was to efficient, non-lossy conversion of the image to an array of grid-boundaries. Most of the algorithms need the maze to be in a format where the starting and ending pixels are marked with distinctive colors and path be of width 1 pixel. The images which are generally available may not be in this format. We approximate the image to a near-ideal requirement of the algorithm. Then, although it takes around $2\times$ more time but the time complexity remains unchanged for most of the situations.

2 Approach

2.1 Pre-processing the image

We start by taking Adaptive Threshold on the initial image followed by a series of erosions and dilations for grouping the unevenly thresholded boundaries. This is followed by a set of iterations of Blurring (Gaussian blurring with a kernel of size 3). This smoothens the unevenness to a great extent. Now, there are two approaches to reduce the white space as much as possible to keep the neighborhood as less as possible for higher efficiency. One is to resize the image by a factor obtained by taking the average path-width. Path-width can be calculated by drawing a series of Hough-Lines then taking the distance between two consecutive of these. But, this makes task tedious. For sake of simplicity we take use the other way. That is, we process the image again in a cycle of thresholding, erosion and dilation (the number of erosions are kept higher than dilations for reducing the path width). This is followed by placing a blue border frame in all four directions (ending positions) and a red colored pixel at the center (starting position).

2.2 Traversing

This part is relatively simpler. Whenever, we make a move we mark the previous pixel as green. Also, we reject green pixels from being counted as valid neighbors to prevent infinite loops. The only pixels which will be valid neighbors would be white or blue. And since the algorithm is just a depth first traversal, the only things we needed to take care of were dead ends and checkpoints.

- **Dead End:** If a pixel has no white or blue neighbor, it is a dead end
- **Checkpoints:** If we were able to make more than 1 valid move from a point, it is a checkpoint.

Whenever we reach a dead end, our strategy was to backtrack the checkpoints and start traversal in a new direction henceforth. Since we keep marking the traversed pixels with green, it automatically prevents retracing same path again. Finally, we store the valid path as a list and mark it at the end.