# Task 3: Path Planning and Computer Vision puzzle

Ashwani Kumar Kamal

***Abstract*— The purpose of this document is to showcase what all steps were taken to solve the puzzle. There were 4 levels to it, each level had some hidden clue about the next one. Starting with some raw images and a very subtle clue to going till the last mp3 treasure. Not to mention that the treasure was absolutely worth it (what do you expect will make someone smile after doing all that work to crack open the treasure :-))**

## I. INTRODUCTION

The puzzle starts off with nothing but an image which seems to be filled up with random noise and a clue mentioning ASCII characters and RGB bits. Solving this part is the most easiest job in the whole puzzle which opens the doors to more clues and a general idea of the big picture. For solving the whole problem, the techniques which I had to implement were- extracting ASCII values from grayed out image, template searching / matching, noise removal, path planning, and then finally getting the mp3 by again extracting ASCIIs.

## II. PROBLEM STATEMENT

The problem statement for this task is the most interesting one as it only comprises of throwing random raw material at the solver and a gentle clue to starting the adventure. Here goes- Your first encounter is with an image called Level1.png. As you might know in a normal RGB scale the colour is represented by a number from 0 to 255 inclusive for each channel. We also know that an ASCII character can be represented as a binary number from 0 to 255 inclusive. So could the thin initial gray scale pixels of the image contain some useful information? Could it help us on what to do with the rest of the pixels in the image? There are a total of 4 files you would need for this task and are provided here. The treasure file is password protected and there are absolutely no clues or directions for what to do with the other images.

## III. RELATED WORK

Studied about how path planning algorithms work- Dijkstra's algorithm, $A*$ algorithm, Breadth first search, Greedy first search, $RRT$ algorithm.

## IV. INITIAL ATTEMPTS

There seemed an absolute lack of information but then again that's how puzzles work in general. The clue given in problem statement about ASCII values and RGB bits appears to be promising. And I also noticed that the first few pixelated lines in Level1.png are grayed out. This, combined with the information that equal amounts of R,G and B always create another shade of gray, I knew what I had to do. I iterated first few characters of the image and noted their RGB bits into a file (after changing them to char data type) and I was astonished- a hidden message and a clue!

## V. FINAL APPROACH

### A. Level 1

I tried iterating all the characters but then it raised an error since some pixel in colored RGB's ASCII was createing a fuss. So I implemented a loop and terminated it when the error occurs (using try and throw an also the fact that grayed out pixels cannot create an error). This was enough to read the whole text and then decide what to do next. The text stated that colored image image startes after the 'colon' character, so I changed the conditions accordingly and saved the hint text into an output text file.

### B. Level 2

The next step was to search the colored part in 'zucky_elon.png' file. At first, the hint created more confusion than it solved, as the dimensions of Level1.png was 177 x 177 however the image to be searched was claimed to be of 200 x 150. Moreover the colored part of image started midway after some 150 characters. It took me time to come upon the fact that maybe the image they are referring to is the matrix pixels transformed to 200 x 150 shape. For confirmation I iterated through the colored part and transformed the matrix values into a blank 200 x 150 image and I was not at all disappointed. The image was of Mark Zuckerburg which can clearly be seen in the 'zucky_elon.png' file. Iterated through the 'zucky_elon.png' and did a basic template search (the same as string search but this time we're comparing pixel values) and drew a rectangle onto the output to show that the search was successful. Noted the x coordinate of the detected location's top left corner for later level.

### C. Level 3

Didn't know how to remove noise but had the noise value- 230. Iterated though some pixels of the maze and checked for 230 in RGB bits, noticed that many pixels (like seriously many) had 230 as the B bit. To get a clear picture, I set the pixel values to NULL (black) where 230 was detected and did FULL (white) otherwise. Saved the image and viewed output. The maze was de-noised and could be recognised cleanly. The next job was to solve it. On first viewing only, one can point out that the maze obstacles appear to be in the shape of some English alphabets. Following steps were implemented in solving it-

- Made a class named $Point()$: will hold the self $(x, y)$, parent $(x, y)$, $distance$ and $visited$ boolean value

- $is\_ok()$ function to check if currentPoint can be explored or not
- $pop\_element\_adj()$ function to get a list of all 'is_ok' adjacent points to the currentPoint
- Fed in the image, made a matrix full of $Point()$'s corresponding to the image $(x, y)$ location
- Added a mouse callback on the image and prompted he user to click at two locations- the start node and end node.
- Started Dijkstra with start in queue, did an easy implementation of priority queue (the node which has the least distance in queue is given index 0 and hence processed first)
- Ran the Dijkstra's algorithm, termination when currentPoint == end
- Backtracked the found path using parent $(x, y)$ on each point staring from end till start

The highlighted path shows the claimed password- 'APPLE'. Indeed it was the password, Implemented some more algorithms and compared their performance.
Extending the idea of Dijkstra, I implemented $A*$ algorithm with $Euclidean$ distance as heuristic. Along with these, I also implemented $BFS$ and $GreedyFirstSearch$ and compared results. Did a walk through of the $RRT$ algorithm. Even implemented successfully on simple mazes, however failed on complex mazes. However what I implemented was perhaps the most naive version of the many $RRT$ algorithms which are out there. Naturally without any kind of heuristic estimation the program is bound to take a large chuck of time, not to mention that the algorithm samples in random directions. But I also cannot deny the fact that $RRT$ algorithms are advantageous when complete information is not available about the obstacle space. Also they turn out to be probabilistically complete and asymptotically optimal ($RRT*$ and its siblings). Got a basic outline of Bidirectionalized $RRT*$.

### D. *Level 4: Treasure*

Opening the password protected zip file, we get another file full of shades of gray. Run another RGB - ASCII converter to generate the text but it didn't play on changing extension. Did a bit of research and came to know that all mp3's are stored like this only- binary ASCII coded. So instead of saving the values to text format, I save them in a dat file (with 'wb' mode). Change the extension and much to my surprise: voila-

Rick Rolled

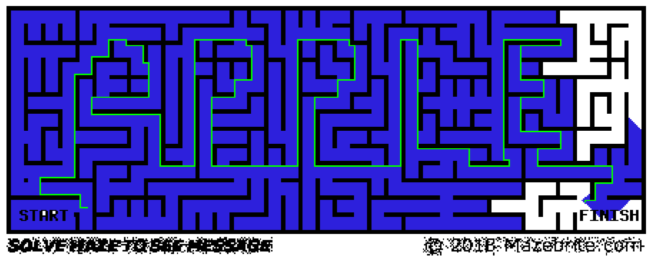
The important lesson that can be drawn is how we can encode mp3 and other media formats onto an image or a dat file and send them wherever applicable.

## VI. Results and Observation

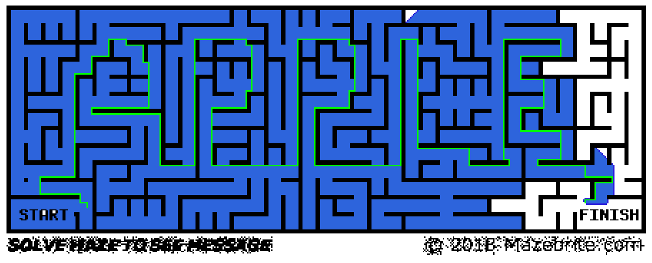
Rick roll the people when they least expect it, now that's what a proper rick roll is lol.

### A. *Comparison table for level 3*

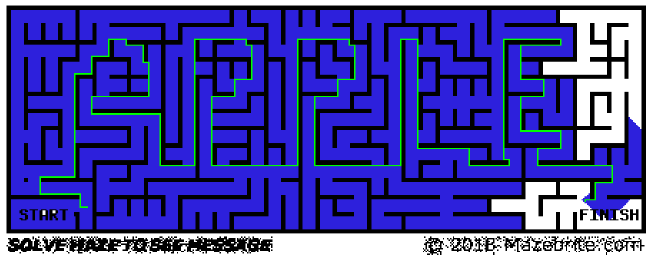
| **Algorithm** | Time (s) |
|---|---|
| Dijkstra | 1.1 |
| $A*$ euclidean | 6.5 |
| $BFS$ | 0.8 |
| $GreedyFirstSearch$ | 14.5 |



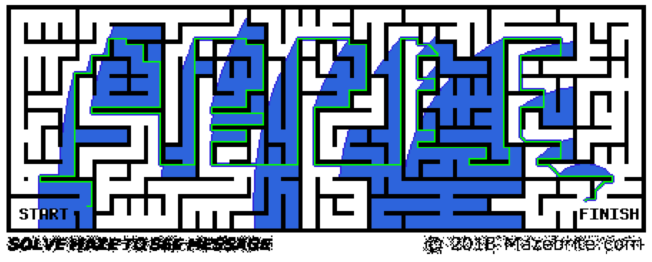
Dijkstra's algorithm



$A*$ algorithm

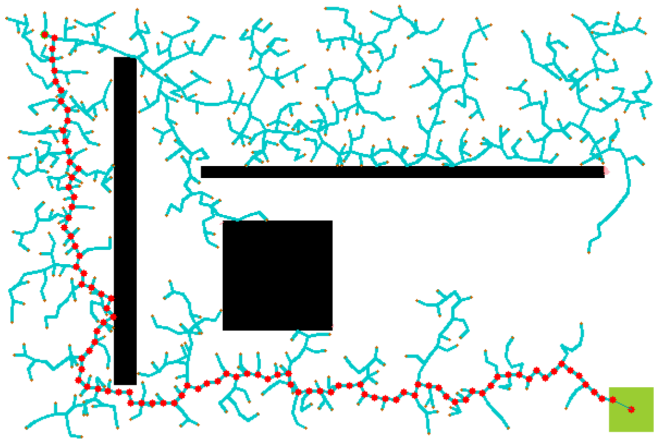


$BFS$ algorithm

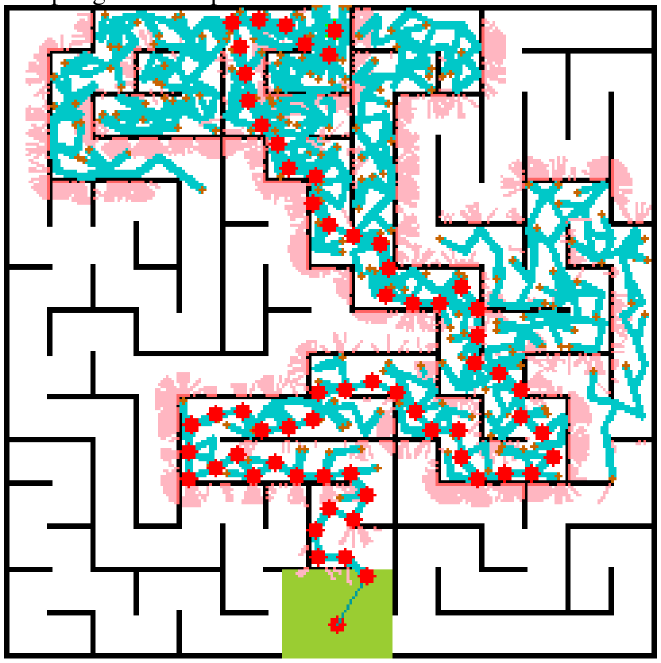


Greedy First Search algorithm

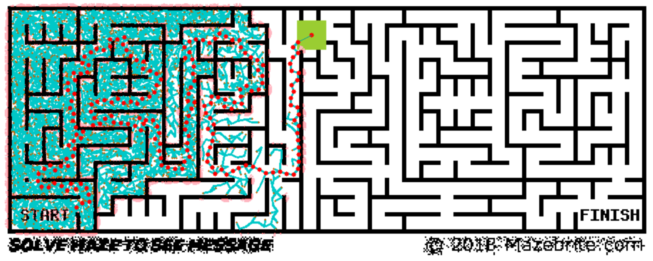
## B. Work on RRT Algorithm



Sampling on a simple maze



Sampling on a maze which I found on a maze generator



Part sampling on given maze (tried to do for half)
Full sampling on the given maze results in memory overflow. I suppose the program should run fine on large sample space (and not the one with small convoluted pockets) but then again it would be very slow without any heuristic.

## VII. Future Work

I really liked the idea of encoding data in images so maybe work on understanding how file transfers work in general

## Conclusion

The puzzle spanned 4 elaborate levels which all required some amount of familiarity with Computer Vision and Path Planning. The end result was not anticipated but had a satisfactory feel.

## References

[1] Amit Patel "Implementation of A*", Red Blob Games, 2014
[2] Rachit Belwariar, "A* Search Algorithm", GeeksforGeeks, 2021
[3] Zaid Tahir, Ahmed Hussain Qureshi, Raheel Nawaz, Yasar Ayaz, "Potentially Guided Bidirectionalized RRT* for Fast Optimal Path Planning in Cluttered Environments", ReasearchGate, 2018
[4] Tim Chin "Robotic Path Planning: RRT and RRT*", Medium, 2019