Maze Solving

by:

Yuki Inoue

Eddie DiLoreto

Maze Solving

Yuki Inoue

Eddie DiLoreto

Abstract:

Inspired by a website discussing a maze solving algorithm that colors wall to solve mazes, we explored the possibilities and the applications of wall-coloring algorithm.

Introduction / Backgrounds:

At one point in our high school career, we were both in the school's Computer Science 2 class. In the class, we learned many things about data structures (ex. stacks, queues), and the teacher mentioned that stacks could be used to solve mazes. This inspired us to look more into this subject matter. And so we looked up various maze solving algorithms on the Internet, and found one interesting attempt to solve mazes. It was to color walls of the maze and by following the path bordering the two color groups, one could solve a maze by following the boundary between two color groups. Although it was a very cool algorithm, it had a major drawback; it can only be used under perfect mazes, and also the starting and the ending cells had to be located at the outermost cells of the mazes. So we decided to expand the applicability of wall-coloring algorithm. Also, while struggling to implement the wall-coloring algorithm, we discovered the idea of pseudo dead ends, which could be used instead of traditional dead ends to make maze solving algorithms more efficient.

The other thing that we looked at was maze synthesis algorithm. For most maze synthesis algorithms, a main factor limiting the largest maze it can create is the computer's memory. As one tries to create bigger mazes, one would have to use more and more components to build that maze, eventually leading up to a memory overflow. A popular way to store maze structure inside of computers is to use JPanels, which act as cells that cover the maze. But JPanels are relatively large in its memory usage. Since the main aim of this experiment is to test what kind of maze solving algorithm can solve maze the fastest, it is important to be able to create larger mazes, so if we use JPanel method, we would only be able to test for mazes up to about 300×300. So that is why we tried to find better alternatives.

Problem:

Is it possible to use wall-coloring algorithm in mazes that are not perfect? Also, is the use of colored walls to check dead ends more efficient than using traditional dead ends?

Hypothesis:

We hypothesize that the wall-coloring algorithm would work most efficiently when the starting/goal cells are outside, and also when there is only one path. Also, when wall-coloring algorithm is used to find the shortest distance between the starting and the ending cells, it would be much more efficient than other such algorithms. So overall, we predict that the wall-coloring algorithm would be more efficient than other maze solving algorithms. Also, the problem that arises when starting/ending cells are inside of mazes could be resolved if Tremaux's algorithm is used as the first part of the wall-coloring algorithm.

Variables:

Independent Variables:

1. Sizes of the Maze
2. Types of the Maze Solving Algorithm

Dependent Variables:

1. How many cells were examined before the goal cell.

Constants:

1. Mazes (algorithms solved the same sets of mazes)
2. Shape of the Maze (square)
3. Shape of the Cells (square)

Materials:

1. Eclipse (Java Programming Platform) 2. Computer

Procedure

We coded a maze that is drawn based on walls rather than JPanels; this is a more efficient way of producing a maze. (The efficiency has been confirmed in one of the experiments)

We made mazes that had a start and finish inside the maze and a start and finish on the edges of the maze. We also made a type of maze that had multiple solution paths. We coded for the two different types of maze solving algorithms (Wall-coloring and Tremaux).

We created a program that would output the desired data for the test and we ran it on maze sizes from 5X5 up to beyond 1000X1000. 20 trials were done for each algorithm, and the size of the maze was increased by 5 every time.

When the maze only had one solution path, we compared the percentage of the cells examined that were actually a part of the solution path. But since it is impossible to find this percentage when a maze has more than one solution path, we just compared the number of cells examined by algorithms to reach the goal.

Then we also applied wall-coloring algorithm in shortest-path seeking algorithms. So we coded two more shortest-path algorithms (water-filling and A* algorithms) and compared the efficiency by finding (number of cells in the shortest path)/(cells examined).

Discussions:

The purpose of this experiment is to come up with and demonstrate how wall-coloring algorithm could be altered, so it could adapt to different conditions. We also assessed the strength of pseudo dead end as a heuristic for maze solving algorithms.

Graph 1. ***Synthesis of Mazes***

According to this graph, it is easy to see that the wall method is far superior than JPanel method is. JPanel method takes a long time to be created, and the time it takes to create mazes increases almost exponentially, unlike the wall method, in which the time increases very slowly. So we have confirmed that the wall method is much faster than the JPanel method.

Graph 2. ***Out, One Path***

This is the graph of maze solving one path, outside start and end cells maze. Since this condition was the condition that the wall-coloring algorithm was created for, we did expect the wall-coloring algorithm to be more efficient than Tremaux's algorithm, and it was the case. With an average of over 96%, wall-coloring algorithm was able to solve mazes much more efficiently than Tremaux's algorithm could. It was interesting to see that while the efficiency of Tremaux's algorithm declined as the size of the maze increased, the efficiency of wall-coloring did not change significantly, indicating that wall-coloring algorithm was independent of maze sizes, or O(1).

Graph 3. ***Out, Multi Path***

This is the graph of multi-paths, start and end cells outside maze. Since we do not know the length of the shortest path of the maze, the comparison was done in terms of how many cells were examined by each algorithm. The trend that we saw in Graph 2 (Tremaux's algorithm decreasing in efficiency as maze size increased) was definitely present in this graph also, except that the efficiency of wall-coloring algorithm declined slowly with the increase in maze size. This decline occurred probably because we had to tweak the algorithm itself so it can solve the maze under the given condition.

Graph 4. ***In, One Path***

This graph is the graph of one-path maze that has starting and ending cells inside of it. As seen in Graph 3, both algorithms declined in their efficiency overtime. This is probably due to the fact that as mazes become larger and larger, the complexity of the maze increases too, making it much harder to solve the maze at a high efficiency.

Graph 5. ***Out, Multi Path***

This was the graph of multi-paths and start and end cells inside of the maze. Since we do not know the length of the shortest path of the maze, the comparison was done in terms of how many cells were examined by each algorithm, just like we did in Graph 3. On average, Wall-coloring algorithm did much better than Tremaux's algorithm again. So we can apply wall-coloring algorithm under start and end cells that are inside of the maze, we combined Tremaux's algorithm and the wall-coloring algorithms. So it can be seen from the graph that some of the efficiency had to be sacrificed for practicality.

Graph 6. ***Shortest Distance Algorithms***

We also decided to look at shortest-path finding algorithms. We used water-filling algorithm and tried to make it more efficient by implementing pseudo dead end heuristic to it. As a result, we were able to make the water-filling algorithm about four times more efficient. When compared to A* algorithm, which is another heuristic, it should be obvious that the pseudo dead end heuristic is very powerful.

Analysis:

The experiments showed that the pseudo dead ends are very strong heuristics that heightens the efficiency of many algorithms. So we analyzed how pseudo dead ends raise the efficiency. The thing that we noticed was that because anything past pseudo dead ends only leads to real dead ends, the program does not have to look any further into that path of the maze. So knowing where pseudo dead ends lessens the number of cells the program have to look at, effectively making the size of the maze smaller. So then we decided to do a follow-up experiment, focusing on how much smaller mazes were made after pseudo dead ends were applied to mazes. We did tests on two types of mazes, one for one-path mazes, and another for multiple path mazes. We hypothesized

that pseudo dead end would be able to shorten the one-path mazes better, because there are less number of right paths.

Discussions on the Follow-up Experiment

As the graph shows, pseudo dead end does shrink the size of the maze, and as we hypothesized, the shrinkage is greater for one-path algorithms. As we explained in the Analysis section, this trend (bigger shrinkages for one-path mazes) is attributed to the fact that there is more number of right paths (paths that lead to the right path) in multiple path mazes. Also, as the size of mazes increased, the shrinkage increased. It is very surprising that a single concept like pseudo dead end can shrink he size of mazes this much. So from this follow-up experiment, we conclude that pseudo dead end increases the efficiency of maze solving algorithms by decreasing the size of mazes significantly.

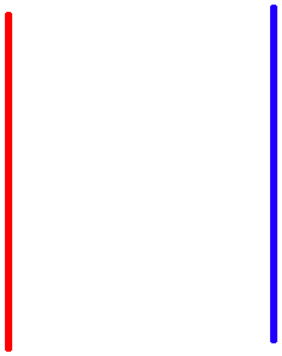
Conclusion:

As we expected, wall-coloring algorithm proved itself more efficiently than the Tremaux's algorithm in all 4 conditions. Since the only difference between Tremaux's algorithm and wall-coloring algorithm was their dead end recognition, this result also lets us conclude that the wall-coloring technique is a more efficient dead end recognition system than the traditional dead ends are. But the results of the experiment also presented wall-coloring algorithm's weakness- for wall-coloring algorithm to work at its full potential, a maze must be a perfect maze (any cells must be able to be reached by any other cells with one path), which is only true when we are testing "outside, one solution path." And any alterations made on the algorithm to fit the condition that deviates from the "outside, one solution path" would lower the algorithm's efficiency.

For shortest path algorithms, we can also conclude that wall-coloring algorithm was the best option to solve mazes with maximum efficiency. From this lab, we were also able to conclude that when compared to other heuristics such as the A* algorithm, wall coloring was a much stronger heuristic.

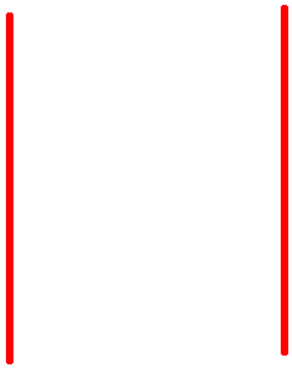
Appendix A: Analysis of Possible Wall Orientation Patterns for Two or More Walls

--> May lead to the right path, for this square may or may not be a boundary between two color walls.



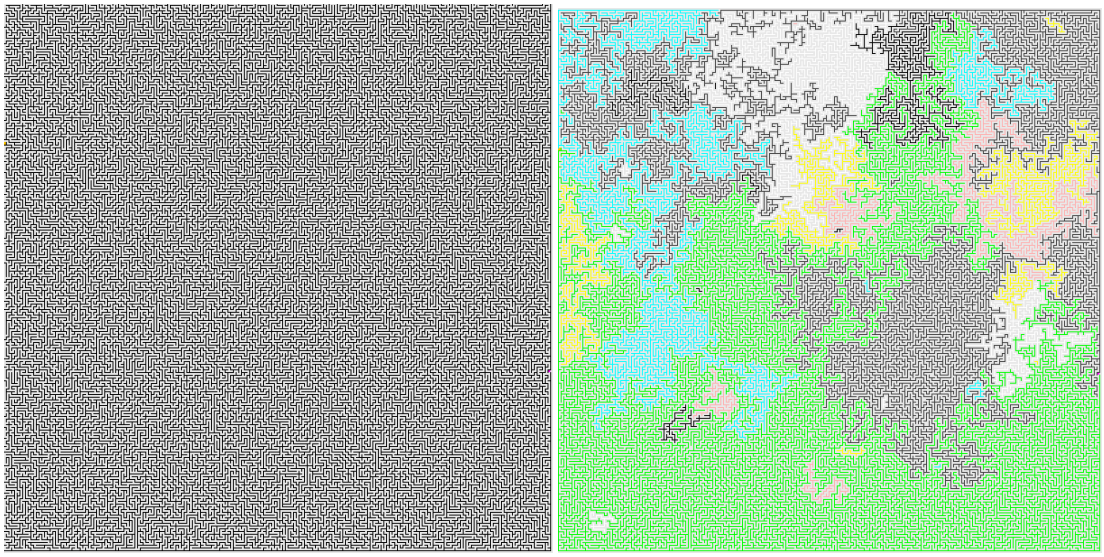
--> Clearly indicates that the square is on the boundary between two color groups. So it leads to the right path.

--> Traditional dead end. Three walls around the square would prevent the program from going anywhere.



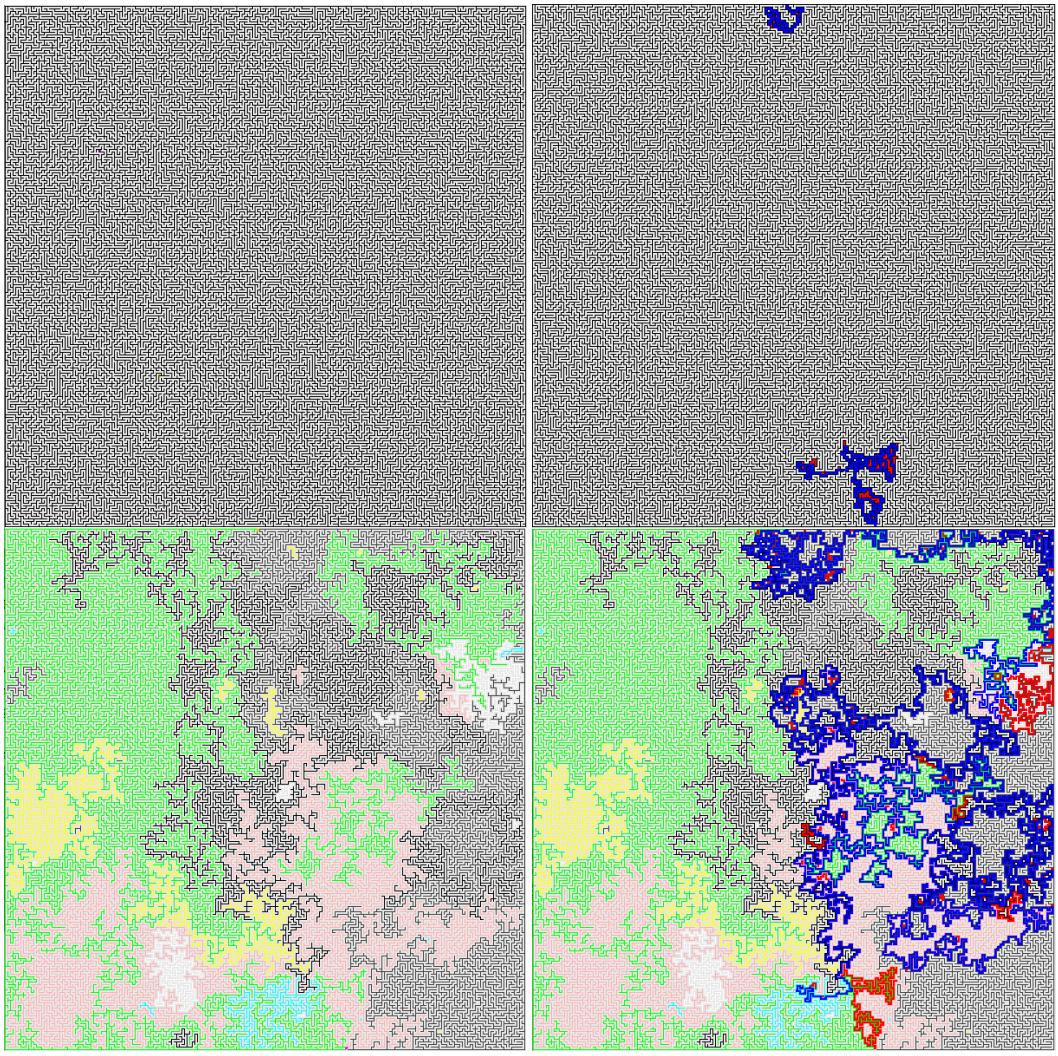
--> Pseudo dead end. This wall pattern indicates that the square will not lead to the right path.

Appendix B: Discussions on "Color Islands":



When walls are colored in Wall-Coloring Algorithm, walls adjacent to each other are colored with the same color. So little "islands" of colored walls form. Because walls making up each island are connected to each other, it is impossible to go through each wall "islands." So as long as the beginning and the ending cells are not in that color group, the program must go around the "islands." What pseudo dead ends do is it stops the program from going inside of islands.

Appendix C: Integrating Tremaux and Wall Coloring:



When starting and the ending cells are inside of a maze, an integration of Tremaux and Wall-Coloring algorithm must happen. First, the algorithm does Tremaux Algorithm from the starting and the ending cells until it reaches the edge of the maze. Then it removes the outward walls of the edge cells, and does the wall-color from those two edge cells. To finish it up, it connects the path that leads from the starting cell to the edge cell #1, the edge cell #1 to the edge cell #2, and the edge cells #2 to the end cell.