Dynamic Weight-Based Pathfinding for Maze Generation Using Randomized Recursive Algorithms

Mutian Lin

Academie Ste Cecile International School, Windsor, Canada

2027m_lin@stececile.ca

Abstract. These days, Artificial Intelligence (AI) has become more and more popular. Pathfinding AI is one of them. There are many ways to train AI, one of them is letting them solve the maze, that is what this paper is talking about. The main logic is the dynamic weight of the road. Assume there is a robot that connects the start and the end. Wherever it passes, is the path. On each position, the algorithm will generate a number based on random numbers and the distance towards the end. After that, the robot will go to the next road that has the highest weight of road. Then continue generating the weight of the next point. Finally, the author of the paper found the maximum number of movements down needs to be higher otherwise the robot will go close to the edge of the maze. Overall, this study demonstrates how simple dynamic weighting rules can guide effective pathfinding in mazes while revealing the importance of careful parameter tuning.

Keywords: Generating maze, random number, generation algorithms.

1. Introduction

Maze is the most attractive problem in centuries, from the mythical Labyrinth of Crete to modern computational puzzles. However, generating a maze needs more creativity, high level math and computer science understanding.

Many researchers made develop many algorithms to generate a maze, generally there are three paths to generating a maze - Breadth First Search (BFS), Depth First Search (DFS) algorithm, prim. Such as in 2010, a research group led by Md. Sazzad Mahmud discussed three variant DFS algorithms. They imagined a general square filled with grids. Then a supposed robot breaks walls between to generate a maze [1]. Also, in 2023, Agnesia discussed using the prim algorithm to generate a maze. They generate a basic grid as the "floor" of the maze, and then use the prim algorithm to decide where is the path and where is the wall [2].

Beside these common algorithms, D Ashlock et al. and his group tried an unusual kind of algorithm. D Ashlock et al. and his group talked about another algorithm that used evolution. D Ashlock They proposed several new ways to express a maze. The first way represents each point in binary, and the second is represented by a rainbow. D Ashlock and his group also designed a fitness function based on the checkpoints. This method will give guidance to any requirements of the maze generating, such as length, size, degree of branches etc. [3]. The group of Paul Hyunjin Kim et al. discussed an algorithm based on the center [4]. This algorithm will generate a topological route that meets the requirement, and scratch the path into a maze. Also, Vijaya discussed the balance between generating engaging content and maintaining gameplay diversity in BFS, DFS, Prim's algorithm [5].

The purpose of this paper is to discuss the value in another kind of common algorithm - recursive description. The basic logic is to determine the start and generate the path to connect them. An imaginary pointer will start from the start point. Next, the pointer will go to the next point which has the Weight of Road (short for WoR). There are many ways to calculate the WoR, such as if the pointer is far from the end, the program will add more WoR that is towards the end. If the pointer is close to the end, the program will add more WoR that is opposite to the end. Therefore, in this paper, the author is going to change the WoR by adding more values and functions.

2. Method

2.1.Maze Design

In this method, mazes are abstracted to the connection relation between points. If there was a road between two neighbors' points, there is a road between. Otherwise, if there is no road, it represents there is a wall. Based on this basic idea, the whole maze has been stored in a custom 'points' created by class. The "point" type includes two integer variables and five Boolean variables. The two integer variables' names were "x" and "y". They were used to represent the precision of a point in the whole maze. The five Boolean variables were "a"," b", "c", "d" and "r" separately. The "a"" b"" c"" d" were used to represent the connection between the point itself to the points surrounding it. For example, if up to the point was a wall, then "a" was equal to false. However, if there is a road, then "a" was equal to true. The relation between the letter and the direction was clockwise and follows the Alphabet. Which means "a" represents the upward direction, "b" represents left direction, "c" represents down direction and "d" represents the right direction. The "r"used to represent the point visited situation. All the "r" s in the maze were all false initially. If the point has been included into the road, then "r" will be equal to true and there will be no new path to include this point. This variable ensures this maze must be a perfect maze (the maze has no circle).

The size of the maze was 30*30, the start and the end were automatically settled at the upper left corner and lower right corner separately, which was position (1,1) and (30,30). All of the maze was stored in a 2-dimensional point array, which can record the road situation of each point in each direction. To give redundancy in the algorithm, the actual 2D array is 32*32. The reason why is to avoid checking out of the edge (such as "maze [3] [-1].x"). And the visualization of the maze is created by AI Grok3.

2.2.Recursive Algorithm

All the logic of this algorithm is based on an abstract tool - a robot (or a pointer/needle). This algorithm had several functions to calculate the current distance between the current point and the end point and a big recursion where the robot and the WoR part. The robot started from the start point. On each point, the robot will calculate the direction of the next step. It only allows more vertically or horizontally. The robot will keep going until he touches the end point. Wherever the robot passed is the path. Similar to the Markov chain, the next point of the robot going to not depends on anything else but the WoR. In this algorithm, the WoR depends on the distance between the current point and the end point. If the distance between two points is ⅓ times the distance between the start points and the end point, the algorithm will make the robot move towards the end point. If the point is smaller than this point, the algorithm will make the robot move backwards to the end.

Besides, to ensure the maze is not the same each time generating, the WoR also has a random number factor involved. At each point, the algorithm is going to generate four random numbers based on time seeds. The system will generate a random number between the current numbers, and add with the WoR adjusting and get the total WoR. The direction that has the highest WoR will become the next step of the robot and included into the path.

This main logic is common for both the main path (the path from (1,1) to (30,30)) and the branches. But the difference is in generating the branches. The algorithm will generate the main path at the very first step. Then on the main path, it will randomly take some points to create branches. Then on the branches, there will be a random number to calculate should this branch have a branch or not. If it is, then generate it. If not, then just keep it there.

3. Results and Discussion

3.1.The performance of the algorithm

Fig. 1, Fig. 2 and Fig. 3 are some of the pictures that show the maze. To better display the main path of the maze, it delineated most of the branches and only left the main path and several branches. Because the point type has “a”,” b”,” c”,” d” in order to represent the four directions. Therefore, the change of the maxim WoR will be represented by {a, b, c, d}.



Fig. 1 Generated maze-1 (Picture credit: Original).



Fig. 2 Generated maze-2 (Picture credit: Original).

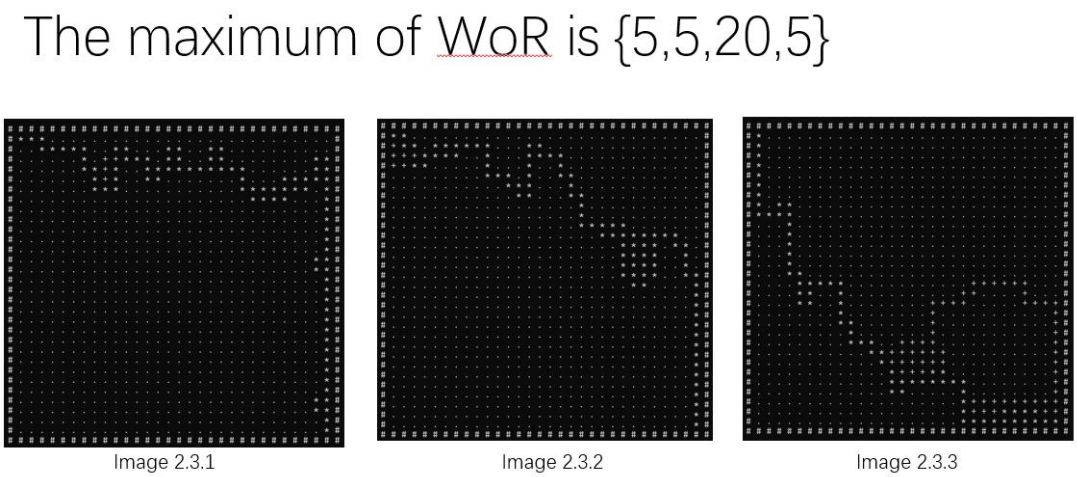


Fig. 3 Generated maze-3 (Picture credit: Original).

The main path with all average random numbers and without any adjustments

3.2.Results description

There is only one result of all average numbers and without adjustments. The reason is the author run the code about 10 times and only got one available answer. Otherwise, the robot will only stay in a small area and never go to the end point. Therefore, the result always has 1000 steps total in a 30*30 maze. It shows that the adjustment is necessary for a maze, otherwise it can't even touch the end.

The difference between two main paths is that the WoR in picture 2 have the same maxim value 5, which the author called it “average”. However, under the average situation, the robot will never go down until he touches the edge and goes down along the wall. However, if increasing the maximum WoR value on moving down direction from 5 to 20, the result will be more non-predictable, like what the images show.

3.3.Discussion

Obviously, this algorithm has some weird beheaver. When generating the main path, the robot is more likely to go left rather than going down. On the other hand, the probability of moving left is much more than moving left. Therefore, to make the main path more natural, the only reason is to increase the maximum value of WoR on moving down direction. The reason this is un-random may be the problem of generating random numbers. For text about 200 times, in a four array of random numbers, the third number has about 60% to be bigger than the second number.

Besides this random number issue, there are also many other things that need to be improved. Until now, the path is not actually random. The path always goes roughly to the lower right and lacks turning. Therefore, to make the path not too boring, it is possible to set some turning points. Such as recording all the turning points. Once there are many turns towards this direction, the algorithm will adjust to turn in the opposite direction. Therefore, there will be more turning points, they will improve the playability of the maze. Besides, the branch generating algorithm also needs optimization. It is possible to set the end of the branch and need some direction to the start, but not only randomly.

4. Conclusion

This paper discussed the weight of roads that affect the main path. The result is - when the end point is at the right-down of the start point, the maxim of the random number that represents moving down needs to be higher than the other three directions. The reason is the comparison for this direction is one of the last numbers that has been compared. On another way to say, if the other

direction has a higher weight of road, the robot will go in the other direction but not down direction. In the future, further study will fix this problem by randomly choosing directions when there is the same weight of reading.

References

[1] Mahmud S, Sarker U, Islam M, et al. A Greedy Approach in Path Selection for DFS Based Maze-Map Discovery Algorithm for an Autonomous Robot. 2012 15th International Conference on Computer and Information Technology (ICCIT), 2012: 546-551.

[2] Istiono AW, Nusantara M, Boulevard S, et al. Wall Pattern Detection with Prim's Algorithm to Create Perfect Random Maze. Journal of Theoretical and Applied Information Technology, [no date], 101(9): 1832-1840.

[3] Ashlock D, Lee C, McGuinness C. Search-Based Procedural Generation of Maze-Like Levels. IEEE Transactions on Computational Intelligence and AI in Games, 2011, 3(3): 260-270.

[4] Kim PH. Design-Centric Maze Generation. Proceedings of the 14th International Conference on the Foundations of Digital Games, 2019: 1-7.

[5] Vijaya J. Analysis of Maze Generation Algorithms. 2024 5th International Conference on Innovative Trends in Information Technology (ICITIIT), 2024: 1-6.