INTERNATIONAL JOURNAL OF INTEGRATED ENGINEERING
ISSN: 2229-838X e-ISSN: 2600-7916

IJIE

Vol. 16 No. 3 (2024) 244-256
https://publisher.uthm.edu.my/ojs/index.php/ijie

Shortest Pathfinding in a Standard Rectangular Maze using A* Search Algorithm

Ka Heng Chan[1], Aida Mustapha[1]*, Siaw Chong Lee[1]

[1] *Faculty of Applied Sciences and Technology,*
Universiti Tun Hussein Onn Malaysia, KM 1, Jalan Panchor, 84600 Pagoh, Johor, MALAYSIA

*Corresponding Author: aidam@uthm.edu.my
DOI: https://doi.org/10.30880/ijie.2024.16.03.022

Article Info

Received: 5 July 2024
Accepted: 27 October 2024
Available online: 9 November 2024

Keywords

Shortest path, A*, maze

Abstract

Mazes are defined as a complicated system that consisted of paths or passages that causes confusion. A large-sized maze can be difficult or impossible to be solved by hand due to high time consumption. This paper presents a solution based on the A* algorithm to find the shortest route. A* algorithm is a searching algorithm that is used to find the shortest path between the initial and the final state. Two heuristic approaches are used and compared using the Manhattan and Euclidean distance. In this paper, A* algorithm is used to solve several sizes and several types of rectangular maze and is evaluated based on the average time in solving a maze for one hundred times under various scenarios. Performance evalution includes mazes with wider paths and multiple routes. Limitations arise when dealing with mazes lacking start or end points. The performance comparison favours Manhattan distance due to its efficiency over Euclidean distance, as demonstrated by the time taken to solve different sizes of mazes. Further exploration is recommended, including the evaluation of both heuristics on larger, more complex mazes. Additionally, the study identifies the A* algorithm's need for modification to handle mazes without defined starting or ending points. This research underscores the suitability of the Manhattan distance heuristic and suggests potential extensions of the A* algorithm to dynamic, changing, and multi-agent maze environments.

1. Introduction

Mazes are defined as a complicated system that consists of paths or passages that causes confusion. They originated from labyrinth that possesses a single through route with twists and turns but without any branches. Hence, we can say that labyrinth was not designed to be that challenging [1]. Mazes, however, unlike labyrinth, consist of many branches and one can easily lost in it [1]. In a maze, the pathways and walls are fixed. Nowadays, a maze is designed as a kind of intelligent challenging puzzle that connects a path between a starting point and an ending point [2]. There are various types of complex patterns available in mazes such as a rectangular maze, circular maze and Tubule maze which are shown in Fig. 1, Fig. 2 and Fig. 3, respectively. While small mazes can be easily solved by hand, large-sized mazes can be difficult or impossible to solve by hand because it is too time consuming to do so. Therefore, we intend to seek for an efficient pathfinding algorithm in this project which not only solves the maze, but also finds the shortest route.

This is an open access article under the CC BY-NC-SA 4.0 license.

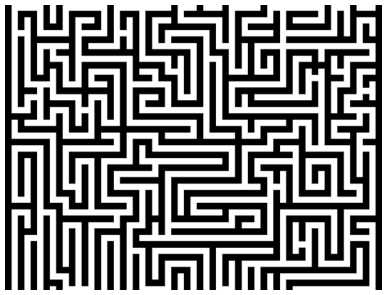


Fig. 1 *Example of a rectangular maze [3]*

Fig. 2 *Example of a circular maze [4]*

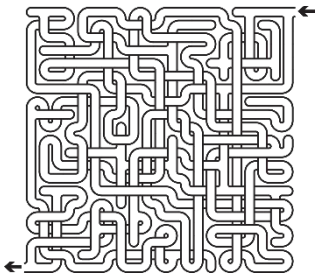


Fig. 3 *Example of a Tubule maze [5]*

Shortest pathfinding is important in real life as complex streets and passages in a map can often be reflected as a type of maze. Pathfinding generally refers to the determination of the route between two points [6]. It can be used to identify the way to move from the source to the destination. The shortest path is an optimization effort of pathfinding. It will give us the low-cost path such as the shortest route or any criteria [7]. In most cases, the path from the source to the destination needs to avoid all obstacles in the lowest cost and short time. To find the shortest path from the source to the destination and prevent obstacles, several pathfinding algorithms such as depth-first search, breadth-first search, Dijkstra's algorithm, best-first search and A* algorithm can be applied [8]. However, the application of inappropriate algorithms can affect the speed of the computing process of pathfinding [9] and we intend to seek out the most efficient solution. Hence, A* algorithm is chosen in our project as it is the most suitable and least time-consuming algorithm when compared to Dijkstra's algorithm and best-first search algorithm [9]. Fig. 4 represents an example of pathfinding between two points.

A* (pronounced as "A-star") algorithm is a searching algorithm that can be used to find solutions for many problems, and it is widely used in the shortest pathfinding between the initial and the final state. It combines information about path cost, which is the exact cost from one point to another point and uses a heuristic approach to effectively compute optimal solutions; unlike Dijkstra's algorithm which only uses path cost. Heuristic approach is a pre-computed technique that is used to find an optimal solution [11]. A* algorithm is the first and only algorithm to use a heuristic function to find the path from the starting point to the ending point [12], making it a preferable algorithm to be utilized in this study as compared to the other algorithms.

Besides, A* algorithm is one of the most used algorithms in solving mazes, board game, puzzle, or video game, as it informs us about how many moves it would require to get the optimal solution. For example, A* algorithm is suitable to solve the maze and 15-puzzle. 15-puzzle is a puzzle that consists of 15 squares numbered from 1 to 15 that are placed in a 4 × 4 box leaving one position out of the 16 as empty. The goal is to reposition the squares from a given arbitrary starting arrangement by sliding them one at a time into the configuration shown in Fig. 5. This algorithm is also widely used for finding the shortest path in labyrinths, navigation, and route-building in real life [13].

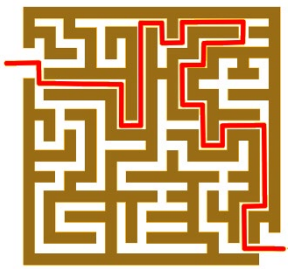


Fig. 4 *Example of pathfinding between two points [10]*

Fig. 5 *Example of 15-puzzle [14]*

Apart from that, A* algorithm is proven to be a better and more efficient algorithm by [9] in games such as maze runner games. Maze runner game is an obstacle arrangement game, that ask player to place the obstacles and block the path of Non-Player Character (NPC) from the start node to the destination node. Then, the NPC will find the shortest path from the start node to the destination node after installing the A* algorithm. Hence, A* is chosen as the algorithm to be used in this project based on the experimental results of other researchers. There are also a lot of applications for the A* algorithm. A* algorithm can be used in solving cell equalizing problem [15]

Int. Journal of Integrated Engineering Vol. 16 No. 3 (2024) p. 244-256

and balancing the energy consumption of wireless sensor network [16]. In path planning, A* algorithm is used in planning the better path for unmanned vehicle [17-25], controlling robot [26-30] and virtual human motion [31].

Although pathfinding may be easy in a small maze, the difficulty to perform pathfinding in a maze increases exponentially as the size of the maze gets larger. The pathfinding in a large maze is too complex that it will be extremely time-consuming to find a solution, and it is almost impossible to find the shortest path if the pathfinding is performed manually. Therefore, an efficient way to perform pathfinding is crucial to our daily lives. In this study, applying A* algorithm to perform pathfinding on mazes of any size will be done to obtain the shortest path of a maze. In this case, we could spend the minimum effort to obtain a solution of the shortest path.

Various types of maze-solving algorithms exist, including Dijkstra's algorithm, Breadth-first search, the backtracking algorithm, and the genetic algorithm. [9] conducted an analysis of pathfinding using the A* algorithm, Dijkstra's algorithm, and the Breadth-first search algorithm within a maze runner game. Their findings indicated that the A* algorithm demonstrated the shortest computation time, the shortest path, and the fewest computed blocks compared to the other two algorithms. Similarly, [13] conducted a comparable study, contrasting the A* algorithm, backtracking algorithm, and genetic algorithm's effectiveness in finding the shortest path in a maze. Their conclusion highlighted the A* algorithm's superior efficiency in maintaining optimal route identification as maze size and obstacle density increased. [32] also evaluated the performance of the A* algorithm, Breadth-first search, Depth-first search, and Iterative Depth-first search algorithms, determining that the A* algorithm outperformed the others.

The main objective of this research is to develop a program base of A* algorithm to compute the shortest path of a rectangular random maze. Moreover, the objective is to test the performance of A* algorithm in several type of mazes such as wider path maze, maze without starting and ending point, blocked ending point and multiple path maze. This research also compares the performance of heuristic approaches that using in A* algorithm, Manhattan distance and Euclidean distance by measuring the completion time for each algorithm.

2. Materials and Methods

2.1 Representation of A Maze and Rules of Maze Solving

To solve a maze in a program, we first translate it into a text file. In the text file, four symbols which are "+", "#", "*S*" and "*E*" which represent the walkable path, non-walkable obstacle, or wall, starting point and ending point respectively. The translation of a maze into the text file is shown in Fig. 6. The pathfinding process starts from point "*S*" to point "*E*". In path finding, at any point *n*, the next step can be one of the eight directions (up, down, left, right or diagonally) as long as there are no walls. The distances of moving one step to up, down, left and right are counted as 1 unit, and distance moving one step diagonally is counted as 1.41 unit ($\sqrt{2}$ unit).

Fig. 6 *Translation of maze in a text file*

Fig. 7 shows the distance for moving one step. Moving one step from the middle point to up, down, left and right is one unit. For top left, top right, bottom left and bottom right, it is $\sqrt{2}$ unit.

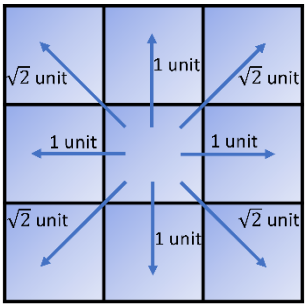


Fig. 7 *The distance for moving one step from the current point*

2.2 A * Algorithm

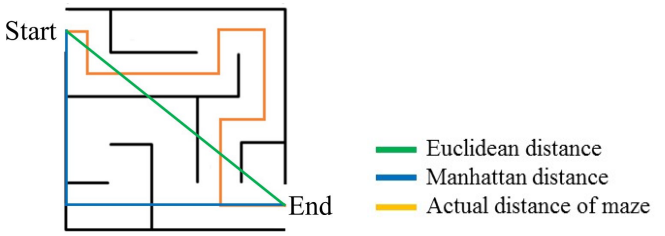
2.2.1 Evaluation Function of A* Algorithm

A* algorithm significantly improved from Dijkstra's algorithm on the computational efficiency aspect by introducing a heuristic approach. Eq. (1) shows the evaluation function of the A* algorithm. In this equation, *n* is a point on the path, *g(n)* is the exact distance from the source point "*S*" to point n, *h(n)* is the heuristic function that estimates the distance from the point n to the desired destination point "*E*" and *f(n)* is the summation of *g(n)* and *h(n)*. Every point to be explored has its *f*, *g*, and *h* value.

$$f(n) = g(n) + h(n) \tag{1}$$

Heuristic approach is a pre-computed technique that estimates the distance of the current point to the ending point in path finding [11]. We can find the distance from the starting point to point *n*, but we cannot exactly know the shortest distance from point *n* to the ending point. Therefore, an estimation is needed and the accuracy of this estimation of the distance is important. If the estimated distance is exactly equal to the real distance, then the best path to solve the maze is found. Thus, a good heuristic function that can accurately estimate the distance may make the path finding algorithm perform better [6]. On the other hand, using a heuristic that overestimates the true distance may result in exploring much fewer points than non-overestimation heuristic approaches [9]. When the heuristic equals to zero, the A* algorithm turns to Dijkstra's algorithm.

To avoid overestimating the heuristic approach of distance from the current point to the ending point, Euclidean distance of these two points is used as the heuristic function. Euclidean distance is a measure of the true straight-line distance between two points in Euclidean space. Therefore, Euclidean distance ensures that the heuristic function will not be overestimated as it is the exact displacement between the current point and desired destination. In this project, we also considered another heuristic, which is the Manhattan distance. Manhattan distance is the distance between two points measured along axes at a right angle. It can be used as an estimated distance that can avoid overestimation of the heuristic approach. Fig. 8 shows Euclidean distance and Manhattan distance between two coordinates in a 2-dimensional space.



$$\text{Euclidean distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$
$$\text{Manhattan distance} = |x_2 - x_1| + |y_2 - y_1|$$

Fig. 8 *Euclidean distance and Manhattan distance between two coordinates*

Equation 2 shows the evaluation function of A* algorithm which uses Euclidean distance as heuristic approach. Equation 3 shows the evaluation function of A* algorithm using the Manhattan distance as heuristic approach.

Int. Journal of Integrated Engineering Vol. 16 No. 3 (2024) p. 244-256

$$f(n) = g(n) + \sqrt{(E_x - n_x)^2 + \left(E_y - n_y\right)^2} \quad (2)$$

$$f(n) = g(n) + |E_x - n_x| + |E_y - n_y| \quad (3)$$

In Eq. (2) and Eq. (3), $E_x$is the x-coordinate of ending point, $E_y$is the y-coordinate of ending point, n_x is the x-coordinate of point n and n_y is the y-coordinate of point n.

2.2.2 Illustration of A* Algorithm

There are four important terms used in the A* algorithm, which are closed list, open list, current point, and parent point. A* algorithm uses two lists to find the shortest distance of the maze which is an open list and a closed list. The closed list contains all the points in the maze that had been totally explored. This means that the closed list contains all the points that we visited. The open list contains all the points in the maze that have not been totally explored yet. This means that the open list contains all the available points around the point we visited. All points on the closed and open list are assigned with their parent point, *g*, *h* and *f* values.

The parent of point *n* is the previous point of point n. Point "*S*", as the starting point of the maze, does not have a parent point. Current point is the latest point that is selected from the open list to explore the maze. A point on the open list that has the lowest *f* value will be chosen as a current point. If there is more than one point with the smallest value of *f*, then A* will choose the first point it found with the lowest *f*. The shortest path from the starting point to the ending point can be found by tracing backward from the destination point to the starting point by using the parent point.

In Fig. 9 to Fig. 17, each point of the maze is illustrated at the left-hand side of the figure. The values of *f*, *g*, *h* value, parent point of each point, open list and closed list will be displayed at the right hand side. The point will be coloured with green and red which represents the point is on the open list and the closed list respectively.

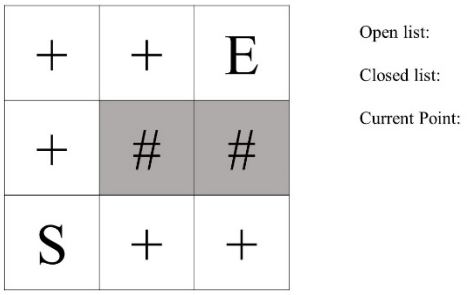


Fig. 9 *Illustration of maze example*

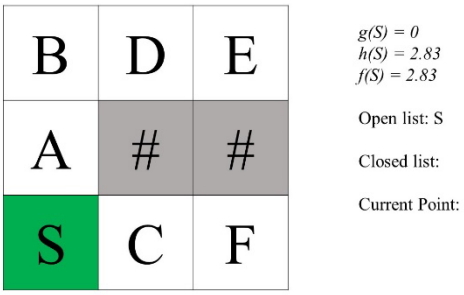


Fig. 10 *Illustration of point "S" is added to the open list and g, h, and f value is displayed*

For pathfinding, the A* algorithm repeatedly explores the nearest location it has seen [6]. In Fig. 10, each available point is assigned with symbol "A", "B", "C", "D" and "F". A* algorithm will first find the starting point which is point "*S*", and then add point "*S*" to the open list and assigns with *g*, *h*, and *f* value. In Fig. 10, point "*S*" is added to the open list and is colored green. The *g*, *h*, and *f* value of point "*S*" is displayed. The *g* value of point "*S*" is 0 since it has not started moving. Euclidean distance from point "*S*" to point "*E*", which is *h* value is 2.83 units, and *f* value is 2.83 units.

Next, the program will find out the point with the smallest *f* value on the open list and set that point as the current point. Fig. 11 represents point "*S*" which is on the open list with the lowest *f* value are switched to closed list and it becomes current point. For each reachable point from the current point, which is the eight points around the current point, if it is on the closed list, we ignore it. If it is not on the closed list and it is not on the open list, we add it to the open list. The *g*, *h*, and *f* value and the parent point of this point will be recorded. If the point is on the open list already, it will check the *g* value of the point to see whether the new *g* value is smaller than its previous *g* value. If so, the parent point will be set to the current point and the *g*, *h* and *f* values will be recalculated.

Fig. 12 illustrates the points around the current point are added to the open list and its *g*, *h* and *f* value of the points is recorded. The *g* value of both point "A" and point "C" is 1 unit. The *h* value of both point "A" and point "C" is 2.23 units. The *f* value for the point "A" and point "C" is 3.23 units.

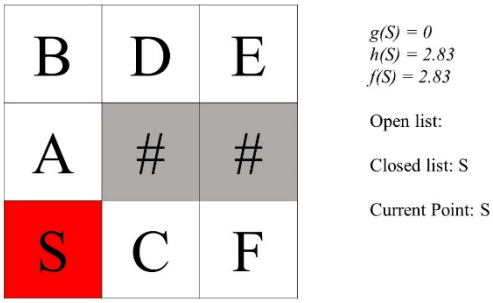


Fig. 11 *Illustration of the point "S" is switched to the closed list*

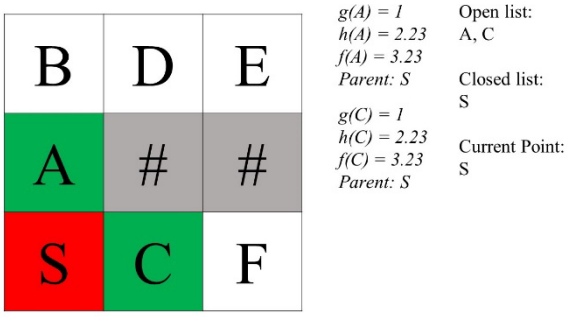


Fig. 12 *Illustration of the reachable points from point "S" are added to the open list with its f, g and h value*

After all of the reachable points are fully recorded, the new current point is set to the point which has the lowest *f* value on the open list. The new current point is switched from the open list to the closed list. In Fig. 13, point "A" and point "C" have the lowest *f* value which is 3.23 units. As we mentioned before, the A* algorithm will choose the first point it detects on the open list. Point "A" is chosen as the current point and switch to the closed list because it is the first point that detects by the A* algorithm. The neighboring points of point "A" are added to the open list if the point is not the element of the open list and closed list.

Fig. 14, Fig. 15 and Fig. 16 illustrate the repeated step in finding of the new current point and calculation of *g*, *h*, and *f* values for every reachable point. These steps will be repeated until the ending point is added to the closed list as shown in Fig. 16. These steps will also stop if the open list is empty. The open list is empty means that the ending point of the maze is blocked. If the destination is found, the shortest path from point "*S*" to the point "*E*" can be obtained by tracing backward from point "*E*" to point "*S*" by using their parent point. Next, Fig. 17 illustrates the shortest path from point "*S*" to point "*E*".

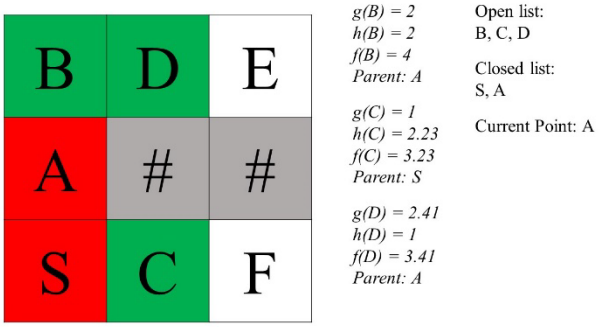


Fig. 13 *Illustration of the lowest f value point on the open list is added to the closed list*

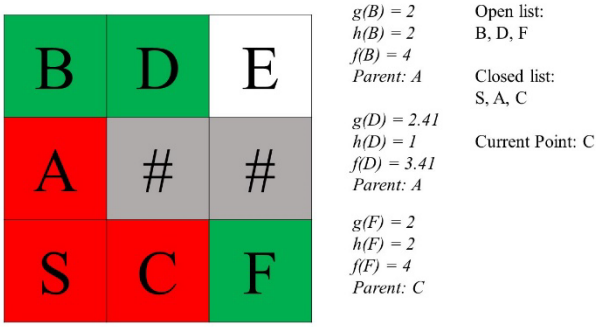


Fig. 14 *Illustration of the repeated step in finding of the new current point and calculation of g, h, and f value for every reachable point*

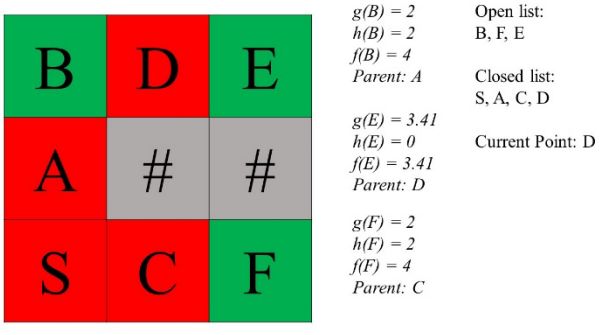


Fig. 15 *Illustration of the repeated step in finding of the new current point and calculation of g, h, and f value for every reachable points (continued)*

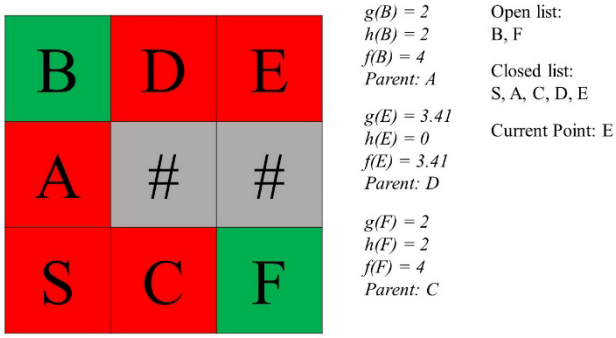


Fig. 16 *Illustration of the repeated step in finding of the new current point and calculation of g, h, and f value for every reachable points (continued)*

Int. Journal of Integrated Engineering Vol. 16 No. 3 (2024) p. 244-256

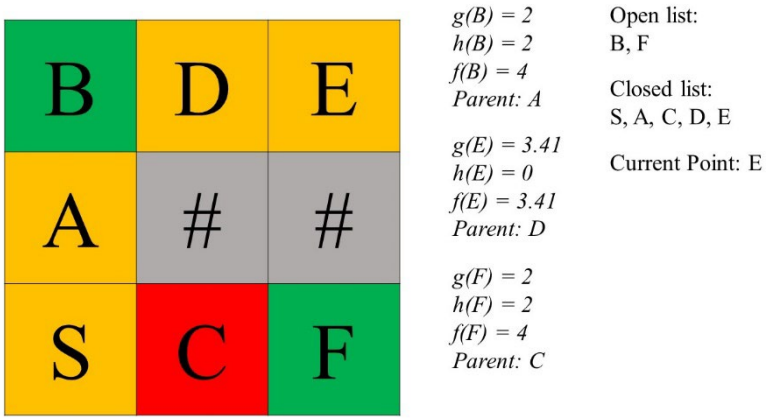


Fig. 17 *Illustration of the shortest path from point "S" to point "E"*

Based on Fig. 17, the path length from point "*S*" to point "*E*" is the *f* value of the point "*E*" which is 3.41 unit. Since the parent point of point "*E*" is point "D", the parent point of point "D" is point "A", the parent point of point "A" is point *"S"*, then the shortest path of the maze is S→A→D→E. All of these steps will be coded in MATLAB. These steps are repeated in solving 5 different size of mazes to determine the capability of A* algorithm. These steps are also used in solving different mazes.

2.3 Flowchart of A* Algorithm in Pathfinding

Fig. 18 illustrates the flow chart of the A* algorithm in pathfinding. MATLAB will start computing the shortest path of mazes by adding the starting point to the open list. Then, it will look for the point which has the lowest *f* value on the open list and adds the point to the closed list and set it as the current point. For every reachable point of the current point, the program will check whether it is on the closed list or not, if so, the point is ignored. If not, the program will check whether it is on the open list or not. If not, add the point to the open list and record the *g*, *h* and *f* values and parent of the point. If so, the program will check whether it has smaller value of *g*. If so, recalculate and change *g*, *h* and *f* values and reset the parent for the point.

After all reachable points have been checked, the program will check whether the ending point is on the closed list or not. If so, the shortest path can be found by tracing backward and the program will stop. If not, the program will check whether the open list is empty or not, if so, no destination found and the program will stop. If the open list is not empty, the program will look for the lowest *f* value point on the open list and repeat until the open list is empty or the destination is found.

3. Results and Discussion

3.1 Path Finding in Several Size of Maze

A program base of A* algorithm in MATLAB is developed. Fig. 19 shows the command window of MATLAB after solving a 10 × 10 maze. Point "*S*" represents the starting point of the maze, point "*E*" represents the ending point of the maze, point "#" represents the wall and point "+" represents the available path. The command window shows the maze at the beginning. Then the solution of the maze is shown at the bottom of the maze. The path of the maze will replace point "+" to point "O" to show the shortest path of the maze. The shortest path of the maze is presented below the solution. The connection of every coordinate will guide to the shortest path of the maze. The length of the path and the time used to solve the maze are also calculated and shown in the command window. This program is also used to solve other four random mazes with size 20 × 20, 30 × 30, 40 × 40 and 50 × 50.

3.2 Path Finding in Several Type of Maze

There is some abnormal maze in our real life, such as maze with a wider path, no starting point or ending point, blocked ending point. There are some mazes with multiple solutions. Therefore, the A* algorithm is used to test whether these mazes can be solved or not.

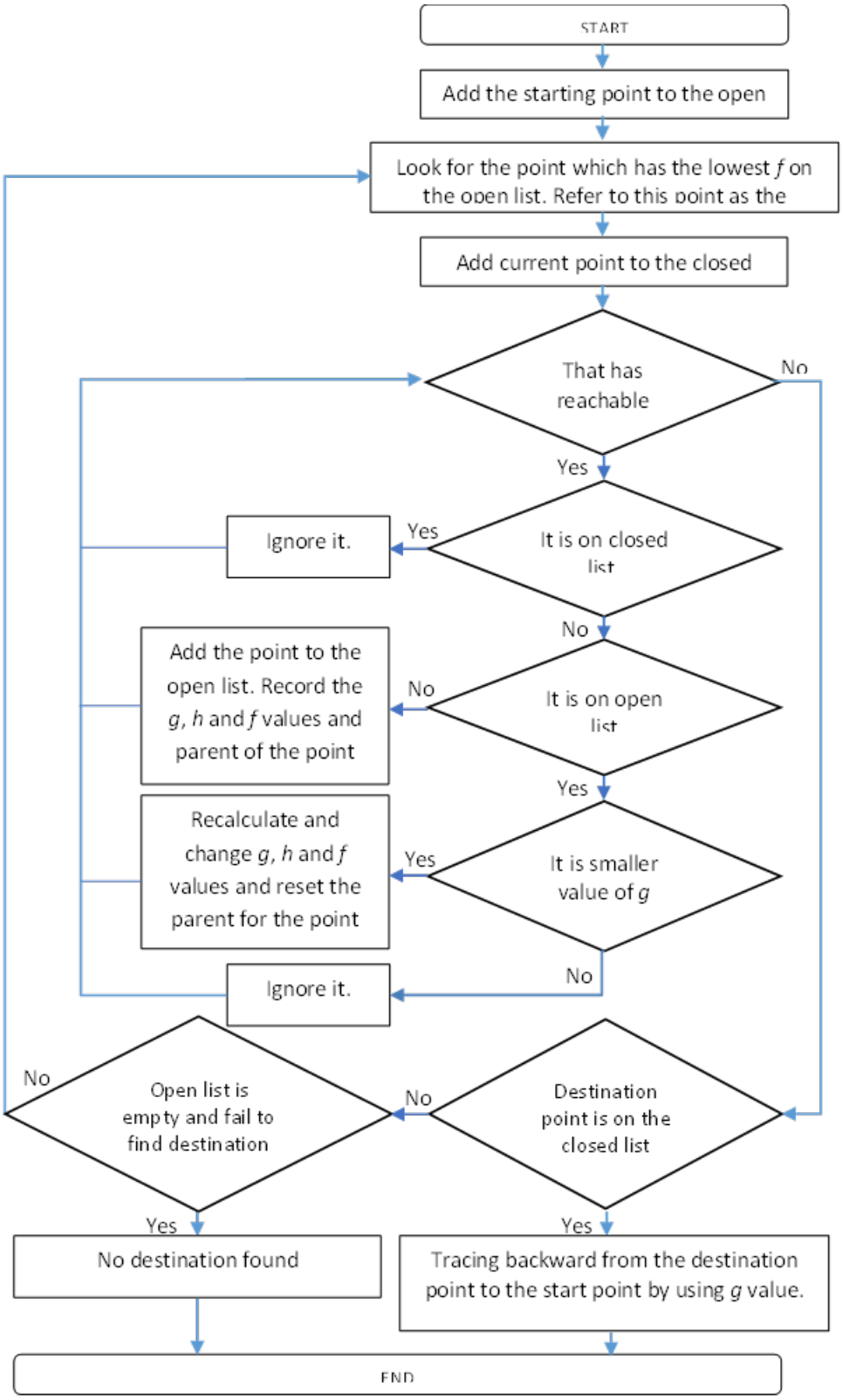


Fig. 18 *Flowchart of A* algorithm in pathfinding*

Int. Journal of Integrated Engineering Vol. 16 No. 3 (2024) p. 244-256

3.2.1 Wider Path Maze

A* algorithm also able to find the shortest path of a wider path maze as in Fig. 20 and 21.

```
Maze:
##########
S+#+++++##
#+###+#+##
#+++++#+##
########+##
#++++#++##
#+###+#+###
#++#+#+###
##+#+#+#+#
#++#+++++E

EXIT found


Solution:
##########
S+#+++O+##
#O###O#O##
#+OOO+#O##
########O##
#++++#+O##
#+##+#O###
#++#+#O###
##+#+#O#+#
#++#+++OOE

Path:
2 1 > 3 2 > 4 3 > 4 4 > 4 5 > 3 6 > 2 7 > 3 8 > 4 8
> 5 8 > 6 8 > 7 7 > 8 7 > 9 7 > 10 8 > 10 9 > 10 10
Path Length = 18.80
Time Used = 0.0579s
```

Fig. 19 *Solution of 10 × 10 maze*

```
Maze:
#++++++++++++++++++++++++++++###
E++++++++++++++++++++++++++++###
###++++++++++++++++++++++#++###
#############################++###
###++++++++++++++#######++##++###
######+################++##++###
######+######++++++++++##++###
###++++++++#######+#######++++#
###+######++#######+#########+++#
###+########++#####+#########+++#
###++++#######++###+########+++##
##############+###+#######+++###
######++++++++++##+######+++####
######++++++++++##++++++++#####
#############+++##++++++++#######
#############+++######++######+#
#############+++###++++++++++#
#++++++++++++++++##########+++#
#+##############++##########+++#
#++++++++######++##########+++#
###############++##########+++#
###++++++++####++##########+++#
###+######+####++##########+++#
###+######+####++##########+++#
###+######+####++##########+++#
###+##++++++++++++++++++++###++#
###+#########################++##
###++++++++++++++++++++++++++###
S++++++++++++++++++++++++++####
#################################
```

Fig. 20 *Excerpt of solution of 30 × 30 maze with wider path*

```
Path:
29 1 > 29 2 > 29 3 > 28 4 > 28 5 > 28 6 > 28 7 > 28 8 > 28 9
> 28 10 > 28 11 > 28 12 > 28 13 > 28 14 > 28 15 > 28 16 > 28 17 > 28 18
> 28 19 > 28 20 > 28 21 > 28 22 > 28 23 > 28 24 > 28 25 > 28 26 > 27 27
> 26 28 > 25 27 > 24 27 > 23 27 > 22 27 > 21 27 > 20 27 > 19 27 > 18 27
> 17 26 > 17 25 > 17 24 > 17 23 > 16 22 > 15 22 > 14 23 > 13 24 > 12 25
> 11 26 > 10 27 > 9 27 > 8 26 > 7 26 > 6 26 > 5 26 > 4 26 > 3 26
> 2 25 > 2 24 > 2 23 > 2 22 > 2 21 > 2 20 > 2 19 > 2 18 > 2 17
> 2 16 > 2 15 > 2 14 > 2 13 > 2 12 > 2 11 > 2 10 > 2 9 > 2 8
> 2 7 > 2 6 > 2 5 > 2 4 > 2 3 > 2 2 > 2 1
Path Length = 83.20
Time Used = 0.3007s
```

Fig. 21 *Solution of 30 × 30 maze with wider path (continued)*

3.2.2 Maze without Starting Point or Ending Point

However, A* algorithm cannot solve the maze if the maze has no starting point or ending point. If maze without a starting point or ending point is inserted, the program does not show the solution of the maze. It shows a message to inform user that there is no starting point or ending point. Fig. 22 and Fig. 23 shows the program inform user there are no starting point or ending point in the maze.

3.2.3 Maze with Blocked Ending Point

If there is no path from the starting point to the ending point or the ending point is blocked by obstacles, the program will show a message to inform user that there is no path from the starting point to the ending point. Fig. 24 shows the program informs user that there is no path from the starting point to the ending point.

3.2.4 Maze with Multiple Path

In some of the mazes, there is more than one path from the beginning point to the ending point. A* algorithm able to find the shortest path from all the available paths. Fig. 25 shows that the A* algorithm chooses the shortest path although there are other paths is available.

Fig. 22 *Command window of 30 × 30 maze with no starting point*

Fig. 23 *Command window of 30 × 30 maze with no ending point*

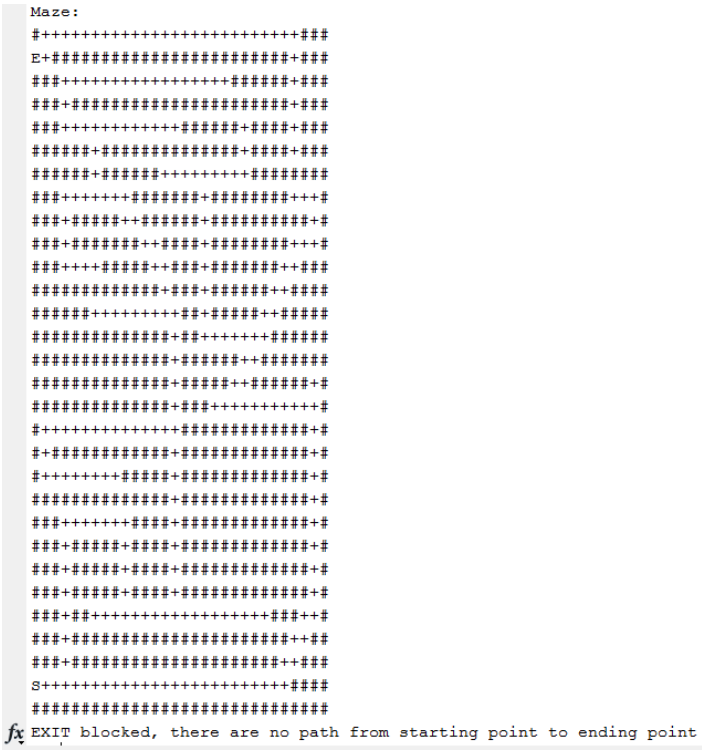


Fig. 24 *Command window of 30 × 30 maze with blocked ending point*

Fig. 25 *Solution of 10 × 10 maze with multiple path*

Int. Journal of Integrated Engineering Vol. 16 No. 3 (2024) p. 244-256

3.3 Comparison in Performance of Heuristic Approach

There are two heuristic approaches that can be used in the A* algorithm to estimate the distance between the current point to the ending point, which are Manhattan distance and Euclidean distance. For each of the heuristic approach applied in the A* algorithm, we record the time taken to find the solution and repeat the experiment for 100 times. Table 1 shows the average time used to solve a maze with several size for one hundred times using Manhattan distance and Euclidean distance in A* algorithm.

Table 1 *Average time used to solve a maze with several size for one hundred times using Manhattan distance and Euclidean algorithm in A* algorithm*

Size	10 × 10	20 × 20	30 × 30	40 × 40	50 × 50
Manhattan Distance	0.0106s	0.0250s	0.0474s	0.0714s	0.1026s
Euclidean Distance	0.0144s	0.0259s	0.0507s	0.0731s	0.1033s

The average time in solving maze of both of the heuristic approaches is calculated to identify the better heuristic approach. The Manhattan distance in the A* algorithm demonstrates a time efficiency advantage over the Euclidean distance, consuming 35.849%, 3.6%, 6.962%, 2.381%, and 0.6822% less time for mazes of sizes 10 × 10, 20 × 20, 30 × 30, 40 × 40, and 50 × 50, respectively. The current point that chosen by A* algorithm will be the same for both Euclidean distance and Manhattan distance. So, we conclude that the difference in the time is solely depends on the operation count.

4. Conclusion

The program of A* algorithm is developed and used to test by using normal mazes with size 10 × 10, 20 × 20, 30 × 30, 40 × 40 and 50 × 50. This algorithm finds the shortest distance of the mazes successfully. The performance of the A* algorithm is also tested by using several types of mazes. The A* algorithm program solves the maze with a wider path and maze with multiple paths and the shortest route of these two types of mazes are determined. On the other hand, for the maze without a starting point and an ending point, this A* algorithm is not able solve them. A* algorithm cannot find the path without source and destination. For the performance of the A* algorithm program in solving a maze in blocked ending point, the A* algorithm able to identify that the maze has no path from starting point to ending point.

In the comparison of the heuristic approach used in this work, which is Euclidean distance and Manhattan distance, both distances that using in A* algorithm able to find the best solution of the maze. In this research, Manhattan distance gives a better result than Euclidean distance in time used. Operation count in the Euclidean distance is more than the Manhattan distance cause the time consuming in calculation. Although Euclidean distance gives the minimum distance between two points, but it takes a longer time in calculation.

The time spent on both heuristic approaches is very close in solving the maze. We cannot judge which heuristic approach is the best because the size and the number of mazes may not big enough to let us conclude it. Hence, we recommend that the comparison of both heuristic approaches should be done by using more mazes and a larger size of the maze. It is advisable to consider mazes of substantial scale, such as dimensions of at least 10,000 × 10,000, characterized by high complexity and a multitude of branching paths. Moreover, the weakness of A* algorithm is that it cannot start solving the maze if the maze has no starting point or ending point. Therefore, A* algorithm needs to be modified to deal with the mazes that have no starting point and ending point. In conclusion, the Manhattan distance is a better heuristic approach in A* algorithm compared to Euclidean distance.

The A* search algorithm offers numerous avenues for future exploration. One key direction involves the development of novel heuristic approaches, aiming to reduce the A* algorithm's dependence on specific starting and ending points. Techniques like multi-point exploration involve starting from random points in the maze rather than the designated starting point. This exploration method can run the A* algorithm to identify the starting and ending points in an unknown maze structure. This technique can be used to solve the maze with unknown starting and ending point. These new heuristics would enable the A* algorithm to handle not only in static mazes but also in dynamic, changing and 3D maze environments. Integrating information-theoretic metrics, such as entropy, allows for the evaluation of moves in a way that maximizes the gain of information about the maze's structure. Using the entropy of the maze path can help manage dynamic and changing maze environments. Moreover, there's potential for extending the A* algorithm to accommodate multi-agent scenarios, thereby enhancing pathfinding efficiency.

Acknowledgement

This research was funded by Universiti Tun Hussein Onn Malaysia under the Research Enhancement Graduate Grant (RE-GG) Scheme (Vot Q078).

Conflict of Interest

Authors declare that there is no conflict of interests regarding the publication of the paper.

Author Contribution

The authors confirm contribution to the paper as follows: ***study conception and design:*** *Chan;* ***data collection:*** *Chan;* ***analysis and interpretation of results:*** *Chan;* ***draft manuscript preparation:*** *Chan; assisted in analysis and editing manuscripts: Mustapha, Lee. All authors reviewed the results and approved the final version of the manuscript.*

References

[1] Pandian, J. Arun, R. Karthick, and B. Karthikeyan. "Maze solving robot using freeduino and LSRB algorithm." *International Journal of Modern Engineering Research* 56 (2012): 92-100.
[2] Murata, Yoshitaka, and Yoshihiro Mitani. "A fast and shorter path finding method for maze images by image processing techniques and graph theory." *Journal of Image and Graphics* 2, no. 1 (2014): 89-93. https://doi.org/10.12720/joig.2.1.89-93.
[3] OpenClipart-Vectors. Download Labyrinth Maze Meander Royalty-Free Vector Graphic. October 22, 2013. Pixabay. https://pixabay.com/vectors/labyrinth-maze-meander-orientation-155972/.
[4] Pheee. Download Maze Puzzle Riddle Royalty-Free Vector Graphic. August 3, 2016. Pixabay. https://pixabay.com/vectors/maze-puzzle-riddle-quiz-labyrinth-1560761/.
[5] DreamDigitalArtist. Download Maze Labyrinth Puzzle Royalty-Free Vector Graphic. September 30, 2021. Pixabay. https://pixabay.com/vectors/maze-labyrinth-puzzle-game-6664727/.
[6] Cui, Xiao, and Hao Shi. "Direction oriented pathfinding in video games." *International Journal of Artificial Intelligence & Applications* 2, no. 4 (2011): 1. https://doi.org/10.5121/ijaia.2011.2401.
[7] Sidhu, Harinder Kaur. "Performance Evaluation of Pathfinding Algorithms." PhD diss., University of Windsor (Canada), 2020.
[8] Barnouti, Nawaf Hazim, Sinan Sameer Mahmood Al-Dabbagh, and Mustafa Abdul Sahib Naser. "Pathfinding in strategy games and maze solving using A* search algorithm." *Journal of Computer and Communications* 4, no. 11 (2016): 15. https://doi.org/10.4236/jcc.2016.411002.
[9] Permana, Silvester Handy, Ketut Yogha Bintoro, Budi Arifitama, and Ade Syahputra. "Comparative analysis of pathfinding algorithms A*, Dijkstra, and BFS on maze runner game." IJISTECH (*International J. Inf. Syst. Technol.*, vol. 1, no. 2, p. 1 (2018). https://doi.org/10.30645/ijistech.v1i2.7.
[10] Mohamed Hassan. Download Problem Solving Maze Labyrinth Royalty-Free Vector Graphic. May 21, 2022. Pixabay. https://pixabay.com/vectors/problem-solving-maze-labyrinth-7210303/.
[11] Mathew, Geethu Elizebeth. "Direction based heuristic for pathfinding in video games." *Procedia Computer Science* 47 (2015): 262-271. https://doi.org/10.1016/j.procs.2015.03.206.
[12] Algfoor, Zeyad Abd, Mohd Shahrizal Sunar, and Hoshang Kolivand. "A comprehensive study on pathfinding techniques for robotics and video games." *International Journal of Computer Games Technology* 2015 (2015): 7-7. https://doi.org/10.1155/2015/736138.
[13] Karova, Milena, Ivaylo Penev, and Neli Kalcheva. "Comparative analysis of algorithms to search for the shortest path in a maze." In *2016 IEEE international black sea conference on communications and networking* (BlackSeaCom), pp. 1-4. IEEE, 2016. https://doi.org/10.1109/blackseacom.2016.7901597.
[14] OpenClipart-Vectors. Download Numbers Game Puzzle Royalty-Free Vector Graphic. October 6, 2013. Pixabay. https://pixabay.com/vectors/numbers-game-puzzle-jigsaw-146484/.
[15] [Dong, Guangzhong, Fangfang Yang, Kwok-Leung Tsui, and Changfu Zou. "Active balancing of lithium-ion batteries using graph theory and A-star search algorithm." *IEEE Transactions on Industrial Informatics* 17, no. 4 (2020): 2587-2599. https://doi.org/10.1109/tii.2020.2997828.
[16] AlShawi, Imad S., Lianshan Yan, Wei Pan, and Bin Luo. "Lifetime enhancement in wireless sensor networks using fuzzy approach and A-star algorithm." *IEEE Sensors journal* 12, no. 10 (2012): 3010-3018. https://doi.org/10.1049/cp.2012.0611.
[17] Tang, Gang, Congqiang Tang, Christophe Claramunt, Xiong Hu, and Peipei Zhou. "Geometric A-star algorithm: An improved A-star algorithm for AGV path planning in a port environment." *IEEE access* 9 (2021): 59196-59210. https://doi.org/10.1109/access.2021.3070054.

[18] Tseng, Fan Hsun, Tsung Ta Liang, Cho Hsuan Lee, Li Der Chou, and Han Chieh Chao. "A star search algorithm for civil UAV path planning with 3G communication." In *2014 Tenth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pp. 942-945. IEEE, 2014. https://doi.org/10.1109/iih-msp.2014.236.

[19] Erke, Shang, Dai Bin, Nie Yiming, Zhu Qi, Xiao Liang, and Zhao Dawei. "An improved A-Star based path planning algorithm for autonomous land vehicles." *International Journal of Advanced Robotic Systems* 17, no. 5 (2020): 1729881420962263. https://doi.org/10.1177/1729881420962263.

[20] He, Z., Liu, C., Chu, X., Negenborn, R. R., & Wu, Q. (2022). Dynamic anti-collision A-star algorithm for multi-ship encounter situations. In Applied Ocean Research (Vol. 118, p. 102995). Elsevier BV. https://doi.org/10.1016/j.apor.2021.102995

[21] Hong, Z., Sun, P., Tong, X., Pan, H., Zhou, R., Zhang, Y., Han, Y., Wang, J., Yang, S., & Xu, L. (2021). Improved A-Star Algorithm for Long-Distance Off-Road Path Planning Using Terrain Data Map. In ISPRS International Journal of Geo-Information (Vol. 10, Issue 11, p. 785). MDPI AG. https://doi.org/10.3390/ijgi10110785

[22] Zhang, J., Wu, J., Shen, X., & Li, Y. (2021). Autonomous land vehicle path planning algorithm based on improved heuristic function of A-Star. In International Journal of Advanced Robotic Systems (Vol. 18, Issue 5, p. 172988142110427). SAGE Publications. https://doi.org/10.1177/17298814211042730

[23] Guo, B., Kuang, Z., Guan, J., Hu, M., Rao, L., & Sun, X. (2022). An Improved A-Star Algorithm for Complete Coverage Path Planning of Unmanned Ships. In International Journal of Pattern Recognition and Artificial Intelligence (Vol. 36, Issue 03). World Scientific Pub Co Pte Ltd. https://doi.org/10.1142/s0218001422590091

[24] Zhang, D., Chen, C., & Zhang, G. (2024). AGV Path Planning Based on Improved A-star Algorithm. In 2024 IEEE 7th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC). 2024 IEEE 7th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC). IEEE. https://doi.org/10.1109/iaeac59436.2024.10503919

[25] Li, J., Liao, C., Zhang, W., Fu, H., & Fu, S. (2022). UAV Path Planning Model Based on R5DOS Model Improved A-Star Algorithm. In Applied Sciences (Vol. 12, Issue 22, p. 11338). MDPI AG. https://doi.org/10.3390/app122211338

[26] Duchoň, František, Andrej Babinec, Martin Kajan, Peter Beňo, Martin Florek, Tomáš Fico, and Ladislav Jurišica. "Path planning with modified a star algorithm for a mobile robot." *Procedia engineering* 96 (2014): 59-69. https://doi.org/10.1016/j.proeng.2014.12.098.

[27] Kusuma, Mario, and Carmadi Machbub. "Humanoid robot path planning and rerouting using A-Star search algorithm." In *2019 IEEE International Conference on Signals and Systems* (ICSigSys), pp. 110-115. IEEE, 2019. https://doi.org/10.1109/icsigsys.2019.8811093.

[28] Ju, C., Luo, Q., & Yan, X. (2020). Path Planning Using an Improved A-star Algorithm. In 2020 11th International Conference on Prognostics and System Health Management (PHM-2020 Jinan). 2020 11th International Conference on Prognostics and System Health Management (PHM-2020 Jinan). IEEE. https://doi.org/10.1109/phm-jinan48558.2020.00012

[29] Martins, O. O., Adekunle, A. A., Olaniyan, O. M., & Bolaji, B. O. (2022). An Improved multi-objective a-star algorithm for path planning in a large workspace: Design, Implementation, and Evaluation. In Scientific African (Vol. 15, p. e01068). Elsevier BV. https://doi.org/10.1016/j.sciaf.2021.e01068

[30] XiangRong, T., Yukun, Z., & XinXin, J. (2021). Improved A-star algorithm for robot path planning in static environment. In Journal of Physics: Conference Series (Vol. 1792, Issue 1, p. 012067). IOP Publishing. https://doi.org/10.1088/1742-6596/1792/1/012067

[31] Yao, Junfeng, Chao Lin, Xiaobiao Xie, Andy JuAn Wang, and Chih-Cheng Hung. "Path planning for virtual human motion using improved A* star algorithm." In *2010 Seventh international conference on information technology: new generations*, pp. 1154-1158. IEEE, 2010. https://doi.org/10.1109/itng.2010.53.

[32] Kumar, Navin, Sandeep Kaur. "A review of various maze solving algorithms based on graph theory." *IJSRD* 6, no. 12 (2019): 431-434.