International Journal of Social Sciences and Scientific Studies(2022)

A Comprehensive and Comparative study of DFS, BFS, and A* Search Algorithms in a solving the Maze Transversal problem

Journal homepage: ijssass.com/index.php/ijssass

A COMPREHENSIVE AND COMPARATIVE STUDY OF DFS, BFS, AND A* SEARCH ALGORITHMS IN A SOLVING THE MAZE TRANSVERSAL PROBLEM☆

Iloh Princess Chinemerem [a,*]

[a] *Teesside university UK*

Received 20 March 2022;
Accepted 10 April 2022
Available online 30 April 2022

A R T I C L E I N F O

Keywords:

Pathfinding

Maze

Maze Solving

Artificial Intelligence

Search Algorithm

DFS

BFS, A*

A B S T R A C T

A search algorithm addresses the challenge of determining the shortest path from the start to the goal while avoiding all possible obstacles. In the quest to design realistic Artificial Intelligence in gaming, we use these algorithms to determine the movement of the agents. The search algorithms for finding the shortest path were implemented using a Maze transversal problem. An agent/player in a maze transversal problem needs a search algorithm to get to its destination and in the least time possible. This algorithm assists an agent/player to travel from the start node to the goal node. The implementation of inappropriate algorithms can alter the length of the computer process for determining the shortest path and the agent/player will have to wait longer as the execution process will take more time. In the Maze transversal problem, pathfinding algorithms, Depth first Search (DFS), Breadth First Search (BFS) and A star (A*) were used for the comparison. The comparison procedure was carried out by running the different algorithms in three (3) mazes with the same dimensions but different obstacles and monitoring the execution time and path length. The findings of this study suggest that the A* search algorithm should be used in the Maze transversal problem as it finds the shortest path to the goal in the shortest possible time and length.

1. INTRODUCTION

Artificial Intelligence (AI) involves developing computer programs that can perform tasks that is ordinarily done by a human being. It is an area of computer science that allows us to develop intelligent machines that can act, think, and make decisions like humans. Artificial intelligence covers 4 main categories: Thinking Humanly, acting humanly, thinking rationally, and acting rationally. AI algorithms are capable of tackling learning problems, perception, solving a problem, understanding a language and logical thinking and gaming. AI has different branches Machine Learning (Supervised, Unsupervised, Reinforcement Learning, and Deep learning), Natural Language Processing [NLP] (which involves content extraction, classification, machine translation, question answering and text generation), Expert systems, vision (Image recognition, Machine vision), Speech (speech to text, text to speech), planning and Robotics. (Ziyad Mohammed, 2019)

Artificial intelligence (AI) is now widely used in video games. Some games have begun to include AI into their gameplay. Action games, adventure games, action-adventure games, RPGs, simulation games, strategy games, sports games, and idle games are all available. AI is gaining popularity in a variety of game genres. (Handy Permana *et al.*, 2018)

A maze game is a complex and perplexing network of passages, and to succeed in a maze game you need to know the best path as the time is taken into consideration, as such the game needs the fastest route, hence a search algorithm. Agents in a maze game perform search algorithm in the background in order to reach the goal. A rectangular grid of cells makes up a basic maze. The Role of Elements in Maze Game:

- Cell: In this game, a cell is the most fundamental unit. It specifies the smallest amount of space that an element should occupy. Free cells and occupied cells are the two types of cells we have.
- Free Cell: A list of cells where the agent is allowed to move is called FreeCell. The agent can begin in the initial condition and is permitted to wander on free cells, with the primary aim of completing the labyrinth being to reach the destination.
- Obstacle (Occupied Cells): An obstacle is a cell or a list of cells that has been occupied. It will not be able to pass through. It implies that we are unable to navigate around obstructions.
- Board: It's made with Pyamaze, and each element in the array represents a cell, therefore, our Maze has 25 by 25 cells (Suchit Purohit, Suraj Singh and Palak Doshi, 2019) **.**

The goal of this study is to find the best path for a Maze Runner game.

1.1 Search Algorithms

Search algorithm is believed to be at the heart of a variety of applications and has a wide range of applications in everyday life. For many years, pathfinding in computer games has been a research topic. It's the most well-known but also a complex Artificial Intelligence (AI) challenge. Pathfinding algorithms have gained popularity in a variety of applications over the years, including Mapping, video games, robotics, and metabolic pathways. It deals with the problem of finding the shortest route between two points while avoiding obstacles if any, Traffic routing, Maze navigation, and robot path planning are examples of such problems (*Search Algorithms in Artificial Intelligence.* , 2015) (Barnouti, Al-Dabbagh and Sahib Naser, 2016) . The most challenging aspect of pathfinding in video games is figuring out how to avoid obstacles while looking for the most efficient way out of the game area. Search Algorithms are widely utilised in the gaming industry. There are two search methods the Uninformed/ Blind Search Algorithm and the Informed Search Algorithm (MdRafiAkhtar and MayankAggarwal, 2021) .

1.1.1 Uninformed/ Blind Search Algorithms:

This is referred to as a blind search Because it has no comprehension of its domain. Blind searches can only discern the difference between a non-goal and a goal condition; they have no preference on which state (node) should be expanded next.

Breadth-first Search (BFS): This algorithm starts from the start node of a tree or graph and works its way to all the successor node at the current level before moving to the nodes in the next level (Milos Simic, 2021) . It searches breadthwise in a tree or graph. In Breadth First Search the frontier is actualized as a queue which works as First In First Out (FIFO) (Begum and Begam, 2021) . It is a bad strategy when every solution has a large path length (Patel and Patel, Oct 2019) .

Depth-first Search (DFS): This algorithm starts searching from the start node and explores each path to its greatest depth node before moving to the next path, before backtracking, to get the optimal path. Backtracking is an algorithmic strategy for addressing issues in a loop by seeking to systematically create a solution. The DFS uses a stack data structure which involves a LIFO (Last in Last Out) and backtracking.

Each of these algorithms will have the following characteristics:

- A problem graph or maze with a start node and a goal node.
- A **strategy,** describing the way the maze will be searched to get to the goal.
- A **data structure** that is used to hold all the potential states (nodes) from the present state.
- •A **tree** that appears as a result of travelling to the destination node. A solution **plan,** which the sequence of nodes from start to goal.

1.1.2 Informed Search Algorithms:

This search algorithm uses the idea of Heuristic search, the algorithm has prior information about the goal state with the aid of heuristics, and this aids the searching to be more effective.

Heuristic Function: this function guesses how close a particular state in a maze is to the goal. It is represented by h(p), and it calculates the cost of an optimal path between the pair of states.

h(p)= Estimated distance between node p and the objective node.

Types of informed search:

- Greedy Search (Best First Search)
- A* Search

Greedy Search (Best First Search): This is a combination of DFS and BFS algorithms. It selects the path which appears best now, it uses heuristic function and search. The Greedy search expands the node that is closest to the goal node and lower the value of h(p), the closer is the node from the goal.

A* Search: A* algorithm is widely used in pathfinding and graph traversal. It is a combination of Uniform cost search (UCS) and Greedy search, it is referred to as the most efficient search algorithm, it finds the shortest path from the start state using the total of the cost in UCS which is g(p) and the cost in Greedy search which is h(p). The total cost is given by f(p). (Mehta *et al.*)

$$F(p) = g(p) + h(p)$$

A* search algorithm is efficient if it meets the following two criteria:

- **Admissibility**: the first requirement for optimality is that h(n) is an admissible A* tree search heuristic. If: $0 \leq h^{*}(p) \leq h * p$ where h (p) is the true cost to a closest goal, a heuristic h is admissible (optimistic).
- **Consistency**.

The strategy is to always choose the node with the lowest f(p) value, If the heuristic function is admissible, then A* tree search will always find the least cost path. (*Search Algorithms in Artificial Intelligence.* , 2015) (MdRafiAkhtar and MayankAggarwal, 2021) . The A* search uses a data structure called priority queue.

The goal of this study is to identify which of the search algorithms Depth First Search (DFS), Breadth First Search (BFS), and A* is best for usage in a maze game. The findings of this study are likely to assist AI game creators in determining the optimal search algorithms to utilize when creating maze-related games.

RESEARCH METHODOLOGY

This study focuses on finding the quickest and most dependable route in a maze game. The research methodology employed in the study is comparative test of the three (3) search algorithms utilizing three (3) different mazes of the same dimensions with different obstacles. After testing the three mazes with the

three search algorithms, it can be determined which algorithm is the best to use in a maze game. The algorithm's performance is measured using the following two parameters:

1. Time complexity: The algorithm's time to find a solution.
2. Path Length: The Length that the algorithm requires to complete the search.

2.1 Depth First Search Algorithm

Pseudocode

- Push the start cell in the Visited and unvisited
- Repeat until goal is reached or the unvisited empty:
    - Current cell= unvisited. Pop ()
    - For each direction (ESNW):
    - Child cell=Next Possible Cell
    - If Child Cell already explored do Nothing
    - Otherwise Append/push the Next Cell to both Explored and Frontier

2.2 Breadth First Search Algorithm

Pseudocode

- Add start cell in both visited and unvisited
- Repeat until the goal is reached or the unvisited empty:
    - Current Cell=unvisited. Pop (0)
    - for each direction (ESNW):
    - Child Cell = Next possible cell
    - If Child Cell already in visited list do nothing else
    - Otherwise Append/push Child Cell to both Visited and Unvisited

2.3 A* Algorithm:

The A* search algorithm is the most widely used path finding method in game Artificial Intelligence (3), One of the benefits of utilising the A* algorithm is that it will always discover the shortest path if the heuristic estimate is admissible (i.e. it never overestimates). While the heuristic must not overestimate, the A* algorithm makes the most efficient use of the heuristic function, expanding fewer nodes and finding an optimal path than any other algorithm employing the same heuristic. Furthermore, a more precise estimate of the heuristic improves the performance of the A* algorithm. Similarly, a heuristic's efficiency might be reduced by a less-than-accurate estimate (Y. C. Hui, E. C. Prakash and N. S. Chaudhari, 2004) .

The pseudocode for A* (5) is as follows:

1. Assume P is the first node.
2. Give P, g, h, and f values.
    - Where g is the cost of travelling from the current node to the destination node.
    - h denotes the anticipated cost of travelling from the source to the destination node.
    - The best estimate of the cost for the path travelling via the source node is f= the sum of g and h.
3. Add the first node to the list of open nodes (which is the priority queue).
4. Perform the following steps again:

    a. On the open list, find the node with the lowest f. This node is referred to as the current node.

    b. Make sure it's in the closed list.

    c. For each node that may be reached from the current node If it is on the closed list, ignore.

    • If it isn't already on the open list, add it. Make this node's parent the current node. This node's g, h, and f values should be recorded.

    • Whether it's already on the open list, evaluate if this is a better option. If this is the case, replace it to the current node's parent and recalculate the g and f values.

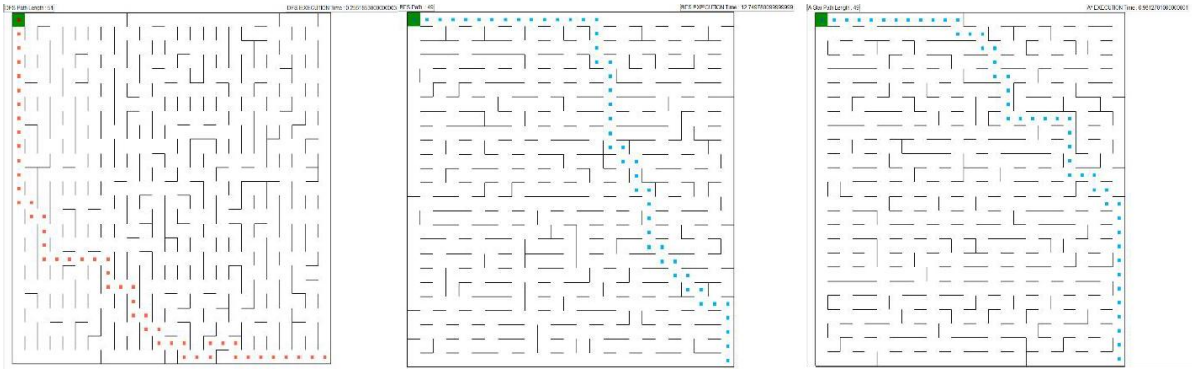
    d. Come to a halt when the goal node is added to the closed list, indicating that the goal has been located.

    ii) If the goal node is not located and the open list is empty, the goal has not been found. Tracing backwards from the source node to the destination node. That is the right course to take (Cui and Hao Shi, 2011) .

RESULTS

The research was carried out on a laptop with a 2.6 GHz dual-core Intel Core i5 processor and 8.0 GB of RAM. Python was the programming language used in the development of the application program. To construct customized random mazes, the pyamaze module was utilized. This module makes use of the Tkinter GUI framework, which is a built-in Python framework. The pyamaze module helps to generate random mazes, the pseudocode of the DFS, BFS and A* algorithm was implemented in a 25 by 25 maze with a horizontal and vertical pattern. Every single time the simulation runs, a new 25 by 25 maze is generated.

In this study, the use of Depth First Search (DFS), Breadth First Search (BFS) and A star (A*) Algorithm in the 25 by 25 maze was compared. The study used the path length and the execution time to perform the comparison for each search algorithm using different obstacle layouts for 3 levels for each search algorithm.



1a **1b** **1c**

Figure 1. Pathfinding of a Maze Game at the First Obstacle Layout using DFS (1a), BFS (1b), and A* (1c) algorithms

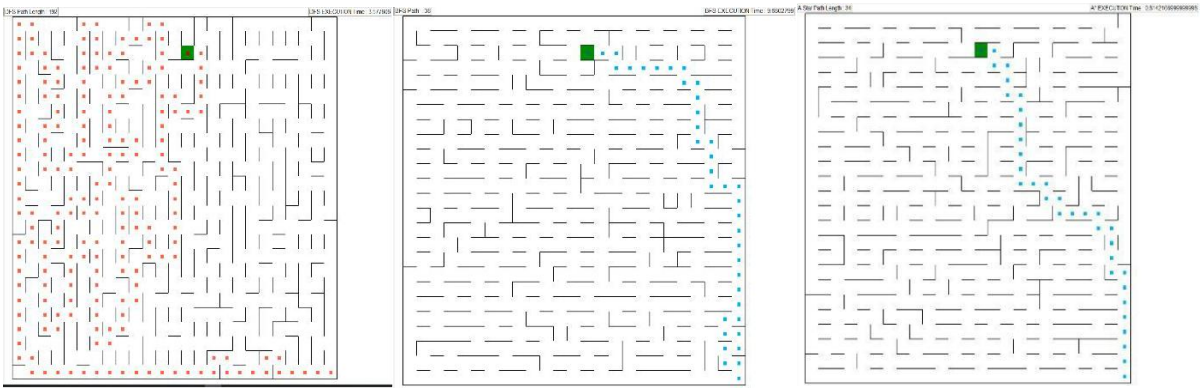
The Figure 1 above, shows the result of the simulation for the Maze game at the first obstacle layout. The first obstacle layout is a 25 by 25 maze with a horizontal pattern, the goal is at the upper right of the maze (1,1). The figure 1a shows the agent's moves with DFS algorithm, and 1b shows the agent's moves with the BFS algorithm while Figure 1c shows the agents moves using the A* algorithm.

Table 1: Comparison of Algorithms for the first Obstacle Layout of the Maze Game

	DFS Algorithm	BFS Algorithm	A* Algorithm
Path Length	51	49	49
Execution Time (milliseconds)	0.2562	12.7498	0.9613

From Table 1 above the path length for the first level was 51 for the DFS algorithm with the execution time of 0.2562 milliseconds, the path length for the BFS and the A* Algorithm was the same at 49 while the execution time of the BFS was 12.7498 milliseconds and the execution time for the A* was 0.9613, from this it can be deduced that the DFS algorithm the reaches the goal at a larger distance but in less execution time compared to the BFS and the A* algorithm, the BFS algorithm has a longer execution time compared to the A* algorithm, Hence, we can conclude that at the first level the A* algorithm did better considering the path length and the execution time.

At the second level of the maze game, a totally different maze is generated for the second obstacle layout with a 25 by 25 maze with a horizontal pattern and a different goal tending towards the upper center of the maze (3,14), and the three algorithms were tested. The simulation result for the Maze game at the second obstacle layout is shown in Figure 2 below.



2a **2b** **2c**

Figure 2. Pathfinding of a Maze Game at the Second Obstacle Layout using DFS (2a), BFS (2b), and A* (2c) algorithms

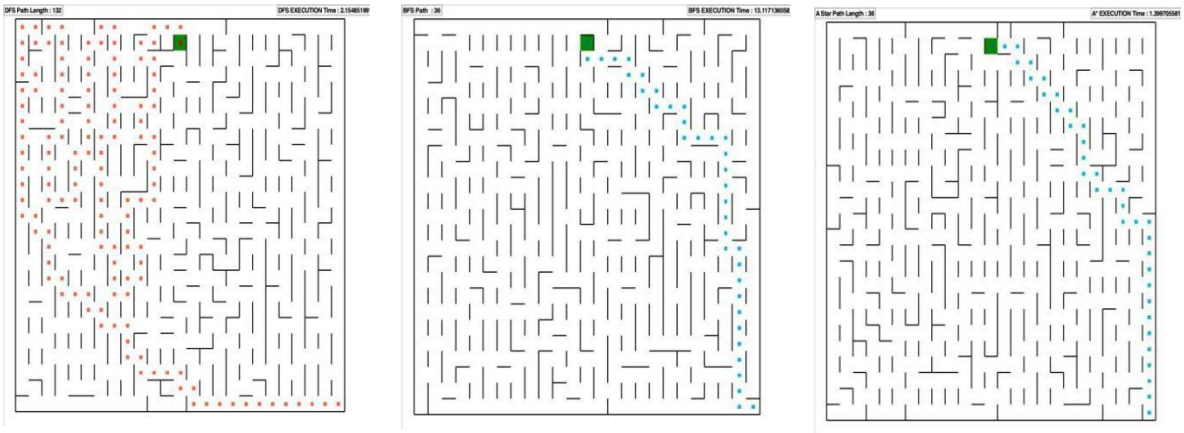
The figure 2 above, shows the result of the simulation for the Maze game at the second obstacle layout. The figure 2a shows the agent's moves with DFS algorithm, and 2b shows the agent's moves with the BFS algorithm while Figure 2c shows the agents moves using the A* algorithm.

Table 2: Comparison of Algorithms for the Second Obstacle Layout of the Maze Game

	DFS Algorithm	BFS Algorithm	A* Algorithm
Path Length	192	38	34
Execution Time (milliseconds)	3.5776	9.6802	0.8142

From Table 2 above the path length for the 2nd level was 192 for the DFS algorithm while the execution time was 3.5776 milliseconds, 38 was the path length for the BFS Algorithm while the execution time was 9.6802 milliseconds, for the A* algorithm the path length is 34 and the execution time is 0.8142 milliseconds , from this it can be deduced that the DFS algorithm the reaches the goal at a larger distance in less execution time compared to the BFS algorithm. The A* algorithm on the other hand outdid the DFS and the BFS as it finds the goal node in the shortest length and the shortest time.

At the third level of the maze game, a totally different maze was generated for the third obstacle layout with a 25 by 25 maze but with a vertical pattern, a different goal tending towards the upper center of the maze (2,13), and the three algorithms were tested. The simulation result for the Maze game at the third obstacle layout is shown in Figure 3 below.



3a **3b** **3c**

Figure 3. Pathfinding of a Maze Game at the third Obstacle Layout using DFS (3a), BFS (3b), and A* (3c) algorithms

The figure 3 above, shows the result of the simulation for the Maze game at the third obstacle layout. The figure 3a shows the agent's moves with DFS algorithm, and 3b shows the agent's moves with the BFS algorithm while Figure 3c shows the agents moves using the A* algorithm.

Table 3: Comparison of Algorithms for the Third Obstacle Layout of the Maze Game

	DFS Algorithm	BFS Algorithm	A* Algorithm
Path Length	132	36	36
Execution Time (milliseconds)	2.1548	13.1171	1.3997

From Table 3 above the execution time of the BFS algorithm, was higher than DFS and A* algorithm although the A* algorithm computed faster than both algorithms at 1.3997milliseconds. The path length of the DFS was comparatively high at132 compared to the BFS and the A* algorithm which tied at 36 each. Hence the A* algorithm did a better job as it found the goal in the shortest path and in a short amount of time.

CONCLUSION & DISCUSSION

Based on the findings of this investigation and analysis, it can be inferred that:

a. In Maze Game, one can employ Depth First Search, Breadth First Search, and A* algorithms to locate the shortest path to the goal.
b. The Depth First Search algorithm is not an effective pathfinding algorithm since it takes a long time to discover the path to a destination.
c. In Maze games and grids, the A* algorithm is the best pathfinding method. This is backed up by the fact that the execution procedure takes only a few milliseconds and that searching for the goal takes a relatively short time.
d. Using the proper algorithm may improve the game's computational process, memory use, and processing time.
e. The Informed search is faster and more likely to discover the best solution than the Uninformed search.

SUGGESTIONS

I would recommend making changes to the Depth First Search, Breadth First Search, and A* Algorithms to improve the results, and then doing another comparison using a different case study, such as a war simulation.

REFERENCES

Barnouti, N.H., Al-Dabbagh, S.S.M. and Sahib Naser, M.A. (2016) 'Pathfinding in Strategy Games and Maze Solving Using A Search Algorithm', *Journal of Computer and Communications,* 4(11), pp. 15-25. doi: 10.4236/jcc.2016.411002.

Begum, I.P. and Begam, I.S. (2021) 'Finding Optimal Algorithm in Artificial Intelligence', *International Journal of Computer Science and Mobile Computing,* 10(7), pp. 84-90. doi: 10.47760/ijcsmc.2021.v10i07.012.

Cui, X. and Hao Shi (2011) 'Direction Oriented Pathfinding In Video Games', *International Journal of Artificial Intelligence & Applications,* 2(4), pp. 1-11. doi: 10.5121/ijaia.2011.2401.

Handy Permana, S.D. *et al.* (2018) 'Comparative Analysis of Pathfinding Algorithms A , Dijkstra, and BFS on Maze Runner Game', *IJISTECH (International Journal of Information System & Technology),* 1(2), pp. 1. doi: 10.30645/ijistech.v1i2.7.

MdRafiAkhtar & MayankAggarwal (2021) *Search Algorithms in AI.* Available at: https://www.geeksforgeeks.org/search-algorithms-in-ai/ (Accessed: 29/11/2021).

Mehta, P. *et al. A Review on Algorithms for Pathfinding in Computer Games A Review on Algorithms for Pathfinding in Computer Games.*

Milos Simic (2021) 'The Informed Vs. Uninformed Search Algorithms', .

Patel, A.R. and Patel, A. (Oct 2019) 'Comparative Analysis of Stereo Matching Algorithms', IEEE. Available at: https://ieeexplore.ieee.org/document/8992965 doi: 10.1109/UEMCON47517.2019.8992965.

Search Algorithms in Artificial Intelligence. (2015) Available at: https://www.javatpoint.com/ai-uninformed-search-algorithms (Accessed: 29/11/2021).

Suchit Purohit, Suraj Singh and Palak Doshi (2019) *Travel Maze Using A* technique.* Slide Player: Palak Doshi.

Y. C. Hui, E. C. Prakash and N. S. Chaudhari (2004) 'Game AI: artificial intelligence for 3D path finding', - *2004 IEEE Region 10 Conference TENCON 2004.* doi: 10.1109/TENCON.2004.1414592.

Ziyad Mohammed, 1. (2019) *Artificial Intelligence Definition, Ethics and Standards.*

☆ A Comprehensive and Comparative study of DFS, BFS, and A* Search Algorithms in a solving the Maze Transversal problem

* Corresponding author at: Teesside university UK.

E-mail addresses: B1397468@live.tees.AC.Uk (P. Iloh)

Received 20 March 2022;
Accepted 10 April 2022
Available online 30 April 2022