CEPAT Journal of Computer Engineering: Progress, Application and Technology
Vol. 1, No. 1, May 2022, pp. 46-55
ISSN 2963-6728, DOI: https://doi.org/10.25124/cepat.v1i01.4863

A Review of Pathfinding in Game Development

Sara Lutami Pardede[1], Fadel Ramli Athallah[1], Fikri Dzulfikar Zain[1], Yahya Nur Huda[1], Ratna Astuti Nugrahaeni[1], Meta Kallista[1], Purba Daru Kusuma[1]
[1]Department of Computer Engineering, School of Electrical Engineering, Telkom University, Indonesia

Article Info

Article history:

Received May 19, 2022
Revised May 29, 2022
Accepted May 30, 2022

Keywords:

Pathfinding
Game programming
A* algorithm
Dijkstra
Breadth-First search

ABSTRACT

Pathfinding is one important method in many studies or works that consists of autonomous movements, such as robots, games, transportation, and so on. Pathfinding aims to find the most efficient route for the related autonomous entity. To date, there are many algorithms regarding pathfinding. Especially, there are four well-known pathfinding algorithms: A*, Theta*, Dijkstra, and Breadth-First Search (BFS). Due to this circumstance, this work aims to observe and review these four well-known algorithms. The discussion consists of the conceptual model or framework, the formalization, and the implementation. Through the comparison review, it is shown that every algorithm has its advantages and disadvantages. This problem often lies in the contradiction between the performance and the computational result. This work also shows that there is a lot of studies that modify or hybridize these algorithms with other methods, such as multi-agent system, metaheuristic, or any other searching method. This circumstance shows that although these algorithms are old, the works to implement or modify these algorithms are still interesting. Hybridization of these algorithms is needed to maintain their effectiveness while tackling its weakness. The result shows that these algorithms have been implemented in many games, so they are promising to be used in future game development, especially in constructing the non-playable character (NPC).

Corresponding Author:

Sara Lutami Pardede
Department of Computer Engineering
School of Electrical Engineering, Telkom University
Bandung, Indonesia
Email: saralutamipardede@student.telkomuniversity.ac.id

1. INTRODUCTION

Pathfinding is a well-known method that was used in many areas, from games, logistics, and autonomous robots. The objective of pathfinding is to give the most efficient route for the autonomous entity to reach its destination. In the game, pathfinding is mostly used for the Non-Player Character (NPC)'s movement, such as in the car racing game [1], in the educational game [2], tower game [3], and so on.

Non-Player Character (NPC) is a character in a video game that cannot be played by the player. Usually, NPC can be an object or other characters such as a villager, animal, plant, vehicle, or monster. In this era, NPC is programmed to make the game's content more varied so that the player does not feel bored while playing the game. For example, NPC is programmed to have an AI pathfinding algorithm so it can find its way to the destination point by the fastest or the shortest route.

Recently, there are many algorithms designed for pathfinding. Each algorithm was designed to tackle a certain objective, whether they are proposing the shortest path, promoting fewer computation resources, and so on. As it is stated in the no-free lunch theory, there is not any algorithm that is the best in all parameters or in solving all routing problems. The effectiveness of an algorithm is also affected by the routing problem it tries to solve. There are four well-known pathfinding algorithms: A*, Theta*, Dijkstra, and Breadth-First Search (BFS). These algorithms have been used widely in many studies regarding pathfinding and routing problems.

This work aims to observe these four well-known algorithms (A*, Theta*, Dijkstra, and BFS). The observation and discussion of every algorithm include the basic concept or framework, formalization, and the implementation of this algorithm in certain fields. Through this exploration, we hope that this work can enrich the discussion regarding pathfinding algorithms.

The remainder of the paper is organized as follows. The A* algorithm is discussed in the second section. The Theta* algorithm is discussed in the third section. The Dijkstra algorithm is discussed in the fourth section. The Best First Search algorithm is discussed in the fifth section. In the end, the conclusion of this work is summarized in the sixth section.

2. A* ALGORITHM

A* algorithm was a pathfinding algorithm that measures the heuristic distance between a given point, while the pathfinding itself relies on the search space on how the A* graph is represented on the field [4]. A* search the route first to see which route it will take, then compare the route and take the shortest path. The formalization can be seen in (1). Meanwhile, the illustration can be seen in Fig. 1.

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

$g(n)$ = the actual cost of a starting point to a specific point
$h(n)$ = heuristic distance
$f(n)$ = the accumulative between the actual point and the heuristic distance

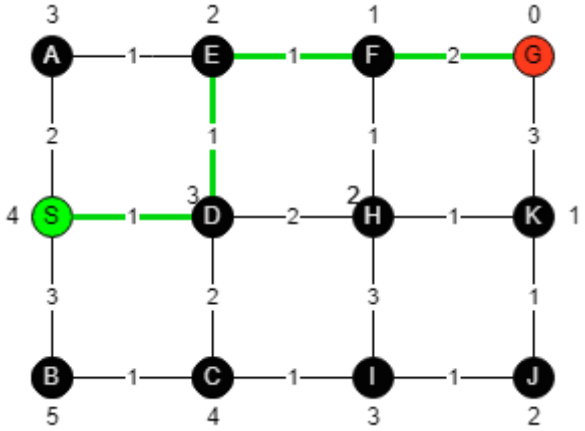

Figure 1. A* Algorithm Demonstration

Sabri et al. [5] proposed experiments with two scenarios: with obstacles and without obstacles. They also used various sizes of the map. In their work, there were two map range sizes: 200x200 and 500x500. There were ten iterations for every experiment. It is also compared between the A* algorithm and the Bee algorithm. The result of the obstacle-free map showed that the A* algorithm was better than the Bee algorithm. However, the Bee algorithm was significantly better than the A* algorithm in solving the 500x500 map with obstacles. They concluded that A* algorithm was ten times faster in the free obstacle game environment rather than the Bee algorithm. But in the complicated environment, the Bee algorithm performed better.

There was also a modification of A* algorithm for a 3D engine to eliminate unnecessary bends in the path. Smołka et al. [6] changed the heuristic to the Chebyshev metric. They also added the vertex penalties to avoid the zig-zag path. They also modified the excess vertices reduction in the path and the algorithm itself. In their work, they created ten test cases and subjected both algorithms to the same tests. There were seven simple and complex heightmaps with various grid sizes of 20x20 and 40x40 to test the algorithm's behavior. 2 heightmaps of a complex labyrinth with grid sizes 80x80 and 40x40 were real photos to check the algorithm in a more complex case. Through experiment, it is shown that the modified A* (A*MOD) was approximately

five times higher than the original A* algorithm. The smoother and more natural routes are produced. Moreover, up to 75% of redundancy was removed.

A* is proven in solving the shortest path problem without considering the obstacles if the generating nodes do not move toward the dead end. Compared to the Obstacle Tracing (OT) algorithm, there was a jagged movement pattern that is caused by the grid or when the gap between the A* node is larger than the gap between obstacles. Husniah et al. concluded that OT is better for runtime performance [7]. They used node count and travel length as a parameter with ten scenarios to test. Non-Player Character (NPC) with both A* and OT algorithm placed in the same spot and need to find a path to destination target. The map that the A* algorithm made by the space between parameters with a value range of 0.1, 0.2, and 0.3.

Another research was about the implementation of the A* algorithm into the autonomous agent in the game along with Navigation Mesh. By using the $f(n)$ score of the node that is generated by Navmesh's polygons, A* can find the shortest path from the AI's spawn point to the determined roaming point or the player's current location. Subrando et al. [8] used a testing ground map for analyzed the algorithm mechanics. After running some tests, both memory usage and the frame rate are less significantly changed. The frame rate falls from 55 FPS to 54 FPS (52 FPS at lowest) and the memory usage increased only from 2200 MB to 2357 MB (peak memory usage). All tests include the unreal engine 4's processes.

When the A* algorithm is hybridized with the dynamic pathfinding algorithm (DPA), a better result was obtained. Sazaki et al. [9] tested it with a qualitative testing technique on an empty track and racetrack that contains the obstacles. In the experiment, the combined A* algorithm and DPA reached the finish line in three out of five experiments. Contrary, the NPC with only DPA reached the finish line in all experiments. The paths are letter S, spirals, 60 degrees, and 30 degrees. But in the other direction, NPC with only DPA experiences failures on the empty racetrack.

When it came to a pathfinding game that has a 16x7 grid in a matrix consisting of game object nodes [10], the number of visited nodes is linearly proportional to the length of the route and the processing time. The nemy must find out which tree will be headed first and find each tree's route using Manhattan Distance. By using the classic heuristic Manhattan distance, we know that the more obstacles, the longer the processing time needed to find a route to the tree.

A* algorithm also can be applied along with fuzzy logic on a game made using the concept of AI. An example is a work conducted by Harsani et al. [11] on their Goat Foraging Games. There were 2 actors: the player and the enemy. The enemy behavior is determined by using Fuzzy logic. A* algorithm is used to determine the shortest distance between enemies and goats while the Manhattan heuristic is used to determine the distance between the enemy and player. They made 3 levels of difficulties, and one of them have 8 dead ends.

A* algorithm also can be implemented on Strategy and Maze Solving games. Barnouti et al. [12] tested the algorithm by using images that represent a map that belongs to a strategy map or represented maze. The map for strategy games was converted to three main colors while the maze only has black as a walkable path and black as a non-walkable path. In their experiment, a player needed to select the source and destination points, then the system will find the shortest path between the selected two points. More than 85% of images can find the shortest path between the selected two points.

Another maze alike experiment was conducted by Permana et al. [13]. They compared the A* with another pathfinding algorithm, such as Dijkstra and Breadth-First Search (BFS). In their research, these algorithms were compared using 3 levels of Maze Runner game with the same obstacle position. The more level, the more obstacles to interfere with. Then, they measured each process time, path length, and the number of blocks through the computational process. They concluded that A* was the best algorithm in pathfinding, especially for maze games/grids. The computational time is low, and the reaching time is short.

You et al. [14] optimized the performance of the A* search to reduce the complexity of the heuristic function design for A*. They proposed evolutionary heuristic A* search, acronymized as (EHA*). In this proposed algorithm, a Genetic Algorithm (GA) is used to optimize the multi-weighted heuristic function (MWH). There are several agents in EHA*, and each of them contains random heuristic functions, then each agent competes and generates better descendants during the iteration. When finished, the algorithm returns to the optimized heuristic function set. They concluded that EHA* is capable enough to design and optimizes MWH function. It also can serve optimal solution paths with minimal computational costs. The summarization of these previous works can be seen in Table 1.

Table 1. Summary of Shortcoming Studies on A* Algorithm

No.	Author	Method	Result
1.	[5]	Compare A* algorithm with Bee Algorithm.	A* algorithm get 10 times better result than Bee algorithm in an obstacle-free map. However, the Bee algorithm gets a better result in a map with obstacles.
2.	[6]	Changed the A* heuristic to the Chebyshev metric and add vertex penalties to avoid zig-zag path (A*MOD).	The modified A* (A*MOD) was approximately five times higher than the original A* algorithm.
3.	[7]	Compare A* algorithm with Obstacle Training (OT).	OT is better on runtime performance than A* algorithm.
4.	[8]	Implemented A* algorithm along with Navigation Mesh (NavMesh).	Memory usage and the frame rate are less significantly changed.
5.	[9]	A* algorithm hybridized with the Dynamic Pathfinding Algorithm (DPA).	The combined A* algorithm and DPA reached the finish line in three out of five experiments. NPC with only DPA reached the finish line in all experiment. NPC with only DPA experiences failures on the empty racetrack.
6.	[10]	A* algorithm with classic heuristic Manhattan Distance.	The more obstacles, the longer the processing time needed to find a route to the tree.
7.	[11]	Implemented A* algorithm along with Fuzzy Logic.	The enemy behavior is determined by using Fuzzy logic. A* algorithm is used to determine the shortest distance between enemies and goats while the Manhattan heuristic is used to determine the distance between the enemy and player.
8.	[12]	A* algorithm.	More than 85% of images can find the shortest path between the selected two points.
9.	[13]	Compared A* with Dijkstra and BFS.	A* was the best algorithm in pathfinding, especially for maze games/grids. The computational time is low, and the reaching time is short.
10.	[14]	Evolutionary Heuristic A* Search (EHA*).	EHA* is capable enough to design and optimizes MWH function. Serve optimal solution paths with minimal computational costs.

3. THETA* ALGORITHM

One of the any-angle pathfinding algorithms was Theta* (Theta-Star). This algorithm was developed based on the A* (A-Star). A* can find the short path but not the true shortest path. Even after post-smoothing, it is restricted to moving from one parent vertex to only the neighbor vertex. Theta* is not restricted to this kind, instead, a vertex will check for each successor whether they have line-of-sight or not. The illustration of Theta* can be seen in Fig. 2.

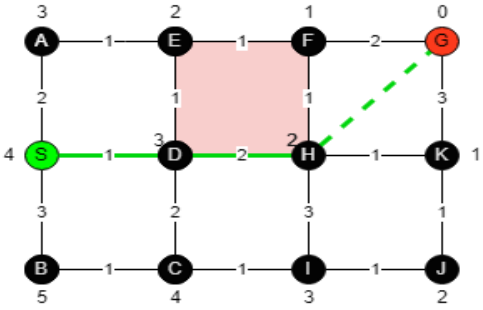

Figure 2. Theta* Algorithm Demonstration

Daniel et al. [15] studied path planning for robotics and video games. However, the grid path might not always be the true shortest path because its potential is artificially constrained to a multiple of 45 degrees. This shortcoming led to the introduction of Basic Theta* and Angle-Propagation Theta* referred to as Theta*. The authors tested the Basic Theta* and AP Theta* alongside with A* and Field D* in Map of Baldur's Gate, and Random Grid in the size of 100x100 and 500x500 with randomized blocked cells from range to 0% to 30% of the size of the map. A* and Field D* managed to find the short path but not the true shortest path. Basic Theta* does not guarantee the real shortest path. Angle-Propagation Theta* as the name suggests propagates angle to determine whether two vertices have line-of-sight. AP Theta* has proven to be able to find the short paths but because there were cases and conditions to be met before updating to the next vertex, it took a longer runtime than Basic Theta*.

Phuc and Dong [16] applied the dynamic weight on the Theta* in the two-dimensional grid map. In every loop, the system must check the line of sight between the expanding cells with the parent of the current cell causing more time needed to find the solution. Based on it, they added dynamic weight values to decrease

the time needed to find the solution. In their experiment, they compare 3 different kinds of theta* in four different sizes of the map (20x20, 30x30, 40x40, and 50x50). On each map, there are 20% of the size of the map is not passable. The results show that dynamic weight on Theta* in all maps has the lowest runtime but in map 50x50 the path length is the highest and in map 30x30 number of nodes is the highest.

Firmansyah et al. [17] compared the A* and Basic Theta* algorithms in the pathfinding games that were built based on the android. The map of the game would be square grids with 6 levels: 5x5, 10x10, 25x25, 50x50, 75x75, and 100x100. The map would have random obstacles, a random starting point, and a random goal point. The result would be compared based on the criteria of completeness (guarantee of the discovery of solution if exist), time complexity (runtime), and optimality (path length and nodes searched). The results show that while A* and Basic Theta* have the same criteria completeness and relatively the same time complexity, the A* algorithm has the advantage of fewer nodes searched, and Basic Theta* has the advantage of optimality of shortest route.

For the modern games, Le et al. [18] applied A* and Theta*. It is also combined with the navigation mesh for polygon structures in three-dimensional games. By converting two maps of popular games, Dota 2 and League of Legends, in 7 different locations of the map into two-dimensional grid maps including the position of obstacles, player, and goal point they managed to demonstrate the A* and Theta* algorithm with and without NavMesh. Theta* results were close to the true path found by humans and the movement looked more natural, but the times taken by Theta* were longer about 70% than A*.

Mendonca and Goodwin [19] introduced Cluster Theta* (C-Theta*). This algorithm was the improved version of Theta*. This algorithm tries to maintain the properties of its parents. This proposed algorithm improves the computational time by using additional information. This information is provided by clustering regions into the high-low density areas based on the number of blocked nodes on the given grid map while performing the search from source to destination. The experiments are conducted using 50x50 and 100x100 grid maps with random obstacles of 20% and 50% density. C-Theta* results in path length were shorter than A* but longer than Theta*, and in runtime C-Theta were faster than Theta* but slower than A*.

Oh and Long [20] tested the Basic Theta*, Strict Theta*, and Recursive Strict theta* by using taut paths. Strict Theta* is implemented the same way as Theta*, except with an additional constant-time tautness check. If the path is not taut, an additional value will be added to the distance. The map that was used was divided into intro categories. Randomly generated maps used were the size of 500x500, with either 6%, 20%, or 40% blocked tiles. “Obstacles” maps and maze maps were of the size of 512x512. Game maps were taken from games like Baldur’s Gate, Starcraft, and Warcraft III. The results show that Strict Theta* found a shorter path than Theta* with a small cost running time. The Recursive Strict Theta* improves even further and almost always finds a taut, and thus believably optimal towards the goal.

Permana et al. [21] created Maze Runner: Angel and Demon, a pathfinding game using C-Theta* for angel NPC and Player as a Demon. NPC has the objective to find a path to the goal while the Player arranged obstacles in a 25x25 grid map with the limitation on certain cells that can not be placed as an obstacle to ensure that NPC will always find the goal. The results have proven that C-Theta* always found a path to the goal if the solution exists. The summarization of these previous works can be seen in Table 2.

Table 2. Summary of Shortcoming Studies on Theta* Algorithm

No.	Author	Method	Result
1.	[15]	Path planning in robotics and video games introduced Basic Theta* and Angle-Propagation Theta* and compared them with A* and Field D*.	A*, Field D*, and Basic Theta* managed to find a short path but Basic Theta* does not guarantee to find the real shortest path. Angle-Propagation was proven to be able to find a short path but took a longer runtime than Basic Theta*.
2.	[16]	Dynamic weight on the Theta* in the two-dimensional grid maps.	Of the 3 different kinds of Theta*, the Dynamic Weight Theta* has the lowest runtime on every map but map 50x50 has the highest path time and map 30x30 has the highest number of nodes visited.
3.	[17]	Compared A* and Basic Theta* in android games with 6 levels maps.	While A* and Basic Theta* have the same criteria completeness and relatively the same time complexity, the A* algorithm has the advantage of fewer nodes searched, and Basic Theta* has the advantage of optimality of the shortest route.
4.	[18]	Applied A* and Theta* in three-dimensional games and combined them with Navigation Mesh.	Theta* results were close to the true path found by humans and the movement looked more natural, but the times taken by Theta* were longer about 70% than A*.
5.	[19]	Introduced Cluster Theta*, a variant of Theta* that used additional information by clustering regions.	C-Theta* results in path length were shorter than A* but longer than Theta*, and in runtime, C-Theta was faster than Theta* but slower than A*.

6.	[20]	Tested the Basic Theta*, Strict Theta*, and Recursive Strict theta* by using taut paths.	Strict Theta* found a shorter path than Theta* with a small cost running time. The Recursive Strict Theta* improved even further and almost always finds a taut, and thus believably optimal towards the goal.
7.	[21]	Created Maze Runner: Angel and Demon, a pathfinding game using C-Theta* for angel NPC and Player as a Demon.	C-Theta* always found a path to the goal if the solution exists.

4. DIJKSTRA ALGORITHM

Dijkstra algorithm is an algorithm that is used to find the shortest paths between two nodes by terminating the process when the objective was achieved. Dijkstra's algorithm is also known as uniform cost search. It can also be seen as an instance of the more general Best-First Search algorithm [22], [23].

There is a directed graph $G = (V, E)$ with n nodes and e arcs, where V is the set of nodes and E is the set of arcs. $W=(i,j)$ is the weight of the arc $<i, j>$/ If there not exists, $W= (i,j)$ [24]. The algorithm complexity is presented as O ((V + E) *log(V) = O (E* log(V)) [25]. The illustration is shown in Fig. 3.

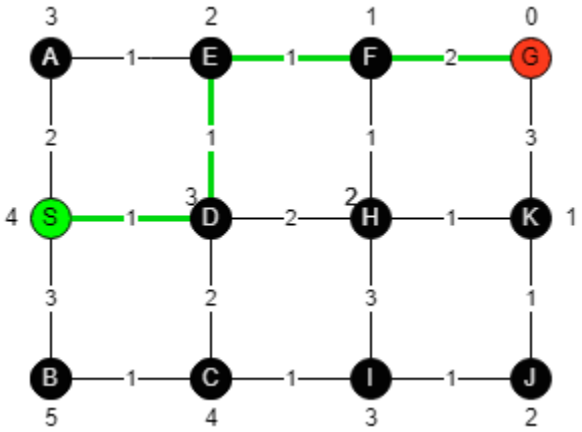

Figure 3. Dijkstra Algorithm Demonstration

Yujin and Xiaoxue used Dijkstra to find the shortest path problem to improve the efficiency of parking spaces in the parking areas. It implemented the balance function between distance and time. It is also combined with the impedance function model. Through simulation, it was known that the algorithm is more optimized by considering the balance of time and distance than by considering one of the factors alone. This proposed algorithm is reasonable and effective. Moreover, it is also promising to be implemented in the intelligent parking system [24].

Iqbal et al. [25] used the Modified Dijkstra Shortest Path algorithm (MDSP). This proposed algorithm uses multiple parameters to find the valid shortest path for road networks where the result was MDSP algorithms prove that the proposed algorithm efficiently finds the shortest path for the road network with minimum time complexity.

Bachri et al. [22] used the Dijkstra algorithm and node combination to find the shortest path in Geographical Information Systems (GIS) where the result is with node combination and Dijkstra algorithm was succeeded in finding the optimal route in the case study route in Taman Sub-district, Sidoarjo Regency, East Java, Indonesia. The results show that the travel distance or the travel time formed the starting point to its destination.

Bozyiğit et al. [26] modified the Dijkstra algorithm for public transport route planning. The objective was to propose the ideal route for the end-users. This ideal route was commonly used by the passengers. A set of rules defined on Dijkstra was used to avoid long walking distances and multiple transfers. This study shows that Dijkstra is the most popular algorithm for finding the shortest path. Meanwhile, this algorithm is not ideal enough to be implemented in the public transport system. In this work, the penalty is also defined. This proposed algorithm was tested on the real-world transport network of Izmir. This algorithm is also compared with the original form of Dijkstra. The result shows that the MDA is much better than Dijkstra Algorithm in the context of the number of transfers and walking distance.

Wang [27] uses three algorithms to compare the shortest path issue where he uses the Dijkstra algorithm, the Bellman-Ford algorithm, and Floyd–Warshall algorithm where those three algorithms have different from each other. Dijkstra algorithm was mainly designed for the graph with non-negative weight nodes. Contrary, the Bellman-Ford algorithm can deal with the shortest path problem with negative weights. They are used to draw the optimal solution for the shortest path. Ironically, the Bellman-Ford algorithm produces excessive redundancy, and its efficiency is low. Dijkstra algorithm also can only be used in single-source shortest path problems. On the other hand, the Floyd-Warshall algorithm is competitive in finding the shortest path between any two points. The summarization of these previous works can be seen in Table 3.

Table 4. Summary of Shortcoming Studies on BFS Algorithm

No	Author	Method	Result
1.	[24]	Combine Dijkstra Algorithm with the impedance function model.	The algorithm is reasonable and effective. Moreover, it is also promising to be implemented in the intelligent parking system.
2.	[25]	Modified Dijkstra Shortest Path algorithm (MDSP).	The algorithm efficiently finds the shortest path for the road network with minimum time complexity.
3.	[22]	Dijkstra algorithm and node combination.	The algorithm is succeeded in finding the optimal route from starting point to the destination.
4.	[26]	Modified Dijkstra Algorithm (MDA).	The result shows that the MDA is much better than Dijkstra Algorithm in the context of several transfers and walking distance.
5.	[27]	Compare the Dijkstra algorithm, the Bellman–Ford algorithm, and Floyd–Warshall algorithm.	Bellman-Ford algorithm produces excessive redundancy, and its efficiency is low. Dijkstra algorithm also can only be used in single-source shortest path problems. Floyd-Warshall algorithm is competitive in finding the shortest path between any two points.

5. BREADTH FIRST SEARCH ALGORITHM

Breadth-first Search (BFS) is an algorithm for searching the nodes of a graph in order by their hop count from the beginning. It is also one of the oldest algorithms and most fundamental graph traversal algorithms that influence many other algorithms. BFS describes a deterministic way of searching and exploring the final node. Iterative loops over a queue of vertices are a standard definition of BFS definition [28]. The illustration of BFS is shown in Fig. 4.

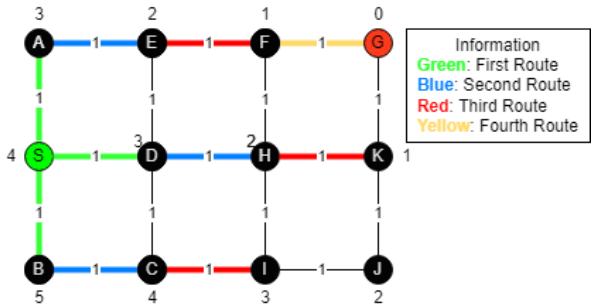

Figure 4. BFS Algorithm Illustration

The complexity of this algorithm can be presented as *O (V + E)*. *V* represents the number of vertices. And *E* represents the number of edges. Moreover, it depends on the data structure that is used to represent the graph. If it is an adjacency matrix, it will be *O (V)*. If we use an adjacency list, it will be *O (V+E)* [29].

Sularno et al. [30] studied to determine the shortest route to the shelter that can be used for the people when a disaster came by using the BFS algorithm. The reason why the authors chose the BFS algorithm was that it uses another heuristic algorithm, because in terms of the distance of the route. This theory will be implemented on a mobile application using Global Positioning System (GPS) and Geographic Information System (GIS) software to do the mapping process in real-time. Using the BFS algorithm as a basis, the result of the test was very precise and can be applied to real applications.

Permana et al. [13] compared the BFS algorithm with A*, DFS, Dijkstra, and Breadth-First Search (BFS) in the maze runner game. There are 3 variables that will be measured on the pathfinding computation; process time, the length of the path, and the number of blocks played. Time will be measured in milliseconds

(ms) where the counting will start from start-node to destination-node when the NPC starts to move. The map is to be used as a 2D map with a lot of variations of obstacles. There will be dynamic obstacles, static obstacles, and a combination of those two. As the result, A* is the best algorithm when used in maze games because of the minimal computing process and time taken to the final point but BFS has a similar result to A*, the big difference is that BFS has more wasteful computers process than A*.

Palanisamy and Vijayanathan [29] addressed Multi-Agent System (MAS) to create a new method for finding the shortest route while using the BFS algorithm. They converted the graphs into several clusters by using the bi-connected region method and assigning AI to each cluster to perform the breadth-first search. The result shows that using clusters can significantly increase the calculation time.

Zhang et al. [31] used the BFS to integrate the architectures of CPU and GPU using different traversal orders. The BFS implementation can process graphs of more than 67 million vertices and one billion edges, and it is executed at approximately 2.1 GTEPS on a single integrated architecture. And then, the result of this study also showed that the energy efficiency of BFS was better than state-of-the-art BFS on integrated architectures.

Lina et al. [32] compared BFS with Depth Limited Search (DLS) algorithm in Sudoku game. Sudoku is a puzzle game from Japan that consists of 81 squares consisting of 9 columns and 9 rows. The players will fill the box with numbers 1 to 9 and there must be no repetition of numbers in one column. After conducting the conclusions, it was said that the DLS algorithm was more efficient and faster than the BFS algorithm. BFS has its own advantages such as it was more structured and systematic that is able to find all possible numbers in each box, but it requires a lot of computer-memory processes. The drawback of this research is that it is only limited to a 3x3 level of Sudoku.

Ramadhani et al. [33] studied the implementation of the Breadth-First Search (BFS) algorithm in the darkness maze games based on desktop intelligent agents. This research has 6 stages: concept, design, material collecting, assembly, testing, and distribution. The first stage; concept is a stage for determining the objectives and who is the that will be the user. Design is a stage for making specifications regarding recruitments for the program. Material collecting is a stage for collecting the components that will be needed in the development process. Assembly is a stage for making all the materials. Testing is a stage to check whether the application has errors or not. Lastly, distribution is a stage for multiplying the results that have been granted. This study uses the third-person shooter (TPS) genre for its research, to find a patch finder that functions as a direction and made the non-playable character (NPC) has a sensor and actuator. This made the NPC will know the position of the player through the environment around. The summarization of these previous works can be seen in Table 4.

Table 4. Summary of Shortcoming Studies on BFS Algorithm

No.	Author	Method	Result
1.	[30]	Determine the shortest route to the shelter that can be used for the people when disaster came by using the BFS algorithm	The result of the test was very precise and can be applied to real applications by using the Global Positioning System (GPS) and Geographic Information System (GIS)
2.	[13]	Compared the BFS algorithm with A*, DFS, Dijkstra, and Breadth-First Search (BFS) in the maze runner game	A* is the best algorithm when used in maze games because of the minimal computing process and time taken to the final point but BFS has a similar result to A*, the big difference is that BFS has a more wasteful computers process than A*
3.	[29]	Using Multi-Agent System (MAS) to create a new method for finding the shortest route while using the BFS algorithm	The result shows that using clusters can significantly increase the calculation time
4.	[31]	Used the BFS to integrate the architectures of CPU and GPU using different traversal orders	The energy efficiency of BFS was better than state-of-the-art BFS on integrated architectures
5.	[32]	compared BFS with Depth Limited Search (DLS) algorithm in Sudoku game	DLS algorithm was more efficient and faster than the BFS algorithm, but BFS has its own advantages such as it was more structured and systematic that can find all possible numbers in each box
6.	[33]	Studied the implementation of the Breadth-First Search (BFS) algorithm in the darkness maze games based on desktop intelligent agent	This study uses the third-person shooter (TPS) genre for its research, to find a patch finder that functions as a direction and made the non-playable character (NPC) has a sensor and actuator that will make the NPC will know the position of the player through the environment around

6. CONCLUSION

This work has demonstrated the discussion of pathfinding, especially for four well-known algorithms: A*, Theta*, Dijkstra, and BFS. This study also shows that these algorithms are still used widely in many shortcoming studies related to pathfinding, especially in game development. Moreover, these four old-fashioned algorithms have been modified and combined with other methods, such as multi-agent systems, metaheuristics, and so on. These algorithms are also implemented in a broader area. This work also shows that each of these algorithms has its own advantages and disadvantages so the selection of these algorithms is based on the nature of the problem that is tried to solve or the environment where this algorithm will be implemented. This dilemma often lies in the performance and the computational resource. As a result, these four algorithms are highly recommended to be implemented in future studies or works in pathfinding and autonomous movement. Moreover, the implementation and improvisation of these algorithms in a broader area are challenging. Studies conducting the hybridization of this algorithm with algorithms are also interesting.

REFERENCES

[1] Y. Sazaki, H. Satria and M. Syahroyni, "Comparison of A and dynamic pathfinding algorithm with dynamic pathfinding algorithm for NPC on car racing game," in *11th International Conference on Telecommunication Systems Services and Applications (TSSA)*, 2017.

[2] D. Kurniadi, A. Mulyani and R. S. Maolani, "Implementation of Pathfinding Algorithm in Sundanese Land History Educational Game," in *2nd International Conference on Innovative and Creative Information Technology (ICITech)*, 2021.

[3] G. T. Galam, T. P. Remedio and M. A. Dias, "Viral Infection Genetic Algorithm with Dynamic Infectability for Pathfinding in a Tower Defense Game," in *18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2019.

[4] L. Husniah, R. Mahendra and A. S. Kholimi, "Comparison Between A* And Obstacle Tracing Pathfinding In Gridless Isometric Game," in *2018 5th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, 2018.

[5] A. N. Sabri, N. H. Radzi and A. A. Samah, "A study on Bee algorithm and A algorithm for pathfinding in games," in *2018 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, 2018.

[6] J. Smołka, K. Miszta, M. Skublewska-Paszkowska and E. Łukasik, "A* pathfinding algorithm modification for a 3D engine," in *III International Conference of Computational Methods in Engineering Science (CMES'18)*, 2019.

[7] L. Husniah, R. Mahendra, A. S. Kholimi and E. Cahyono, "Comparison Between A And Obstacle Tracing Pathfinding In Gridless Isometric Game," in *2018 5th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, 2018.

[8] T. Subrando, D. Fitrianah and F. A. Prasetyatama, "Implementation of a* algorithm within navigation mesh in an artificial intelligence based video games," in *International Journal of Engineering & Technology (IJET)*, 2018.

[9] Y. Sazaki, H. Satria and M. Syahroyni, "Comparison of A and dynamic pathfinding algorithm with dynamic pathfinding algorithm for NPC on car racing game," in *2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA)*, 2017.

[10] A. Candra, M. A. Budiman and R. I. Pohan, "Application of A-Star Algorithm on Pathfinding Game," in *5 th International Conference on Computing and Applied Informatics (ICCAI 2020)*, 2020.

[11] P. Harsani, I. Mulyana and D. Zakaria, "Fuzzy logic and A* algorithm implementation on goat foraging games," in *IOP Conference Series: Materials Science and Engineering.*, 2017.

[12] N. Barnouti, S. Al-Dabbagh and M. Naser, "Pathfinding in Strategy Games and Maze Solving Using A* Search Algorithm," in *Journal of Computer and Communications*, 2016.

[13] S. Permana, B. Arifitama, K. Bintoro and A. Syahputra, "Comparative Analysis of Pathfinding Algorithms A *, Dijkstra, and BFS on Maze Runner Game," *International Journal of Information System and Technology,* vol. 1, no. 2, 2018.

[14] Y. F. Yiu, J. Du and R. Mahapatra, "Evolutionary Heuristic A Search: Heuristic Function Optimization via Genetic Algorithm," in *2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, 2018.

[15] K. Daniel, A. Nash, S. Koenig and A. Felner, "Theta*: Any-Angle Path Planning on Grids," in *arXiv e-prints*, 2014.

[16] B. P. Le and L. Ki-dong, "Applying Dynamic Weight on Theta Star Path-finding Algorithm in 2D Grid Map," in *2015 International Conference on Intelligent Computing, Electronics System and Information Technology*, 2015.

[17] E. R. Firmansyah, S. U. Masruroh and F. Fahrianto, "Comparative Analysis of A and Basic Theta Algorithm in Android-Based Pathfinding Games," in *6th International Conference on Information and Communication Technology for The Muslim World (ICT4M)*, 2016.

[18] P. T. Le, N. T. Truong, M. S. Kim, W. So and J. H. Jung, "Applying Theta* in Modern Game," *Journal of Computers,* vol. 13, no. 5, 2018.

[19] P. Mendonca and S. Goodwin, "C-Theta: Cluster Based Path-Planning on Grids," in *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2015.

[20] S. Oh and H. W. Leong, "Strict Theta*: Shorter Motion Path Planning Using Taut Paths," in *Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*, 2016.

[21] S. D. Permana, K. B. Bintoro, B. Arifitama and A. Syahputra, "Maze Runner : Angel and Demon Path Finding Game Application using C-Theta* Algorithm," in *The 1st International Conference on Computer Science and Engineering Technology*, 2018.

[22] A. Fitro, O. S. Bachri, A. I. S. Purnomo and I. Frendianata, "Shortest path finding in geographical information systems using node combination and dijkstra algorithm," *International Journal of Mechanical Engineering and Technology (IJMET),* vol. 9, no. 2, pp. 755-760, 2018.

[23] G. Qing, Z. Zheng and X. Yue, "Path-planning of Automated Guided Vehicle based on Improved Dijkstra," in *2017 29th Chinese Control And Decision Conference (CCDC)*, 2017.

[24] Yujin and G. Xiaoxue, "Optimal Route Planning of Parking Lot Based on Dijkstra Algorithm," in *International Conference on Robots & Intelligent System*, 2017.

[25] M. Iqbal, K. Zhang, S. Iqbal and I. Tariq, "A Fast and Reliable Dijkstra Algorithm for Online Shortest Path," *SSRG International Journal of Computer Science and Engineering (SSRG – IJCSE),* vol. 5, no. 12, 2018.

[26] A. Bozyiğit, G. Alankuş and E. Nasiboğlu, "Public Transport Route Planning: Modified Dijkstra's," in *International Conference on Computer Science and Engineering (UBMK)*, 2017.

[27] X. Z. Wang, "The Comparison of Three Algorithms in Shortest Path Issue," in *Journal of Physics Conference Series*, 2018.

[28] H. Jason, "The Nature of Breadth-First Search," in *School of Computer Science, Mathematics, and Physics James Cook University Australia*, 2019.

[29] V. Palanisamy and S. Vijayanathan, "Cluster Based Multi Agent System for Breadth First Search," in *2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer)*, 2021.

[30] S. Sularno, D. P. Mulya, R. Astri and D. Mulya, "Determination of The Shortest Route Based on BFS Algorithm for Purpose to Disaster Evacuation Shelter," *Scientific Journal of Informatics,* vol. 8, no. 1, 2021.

[31] F. Zhang, H. Lin, J. Zhai, J. Cheng, D. Xiang, J. Li, Y. Chai and X. Du, "An Adaptive Breadth-first Search Algorithm on Integrated Architectures," *The Journal of Supercomputing,* vol. 74, p. 6135–6155, 2018.

[32] T. N. Lina and M. S. Rumetna, "Comparison Analysis of Breadth First Search and Depth Limited Search Algorithms in Sudoku Game," *Bulletin of Computer Science and Electrical Engineering,* vol. 2, no. 2, pp. 74-83, 2021.

[33] S. Ramadhani, H. Barholomius and E. Hanifah, "Implementation of the Breadth First Search Algorithm in the Darkness Maze Game Based on Desktop Intelligent Agent," *TEPIAN Agricultural Polytechic of Samarinda,* vol. III, no. 2, pp. 125-129, 2021.