International Journal Of Information System & Technology
Vol. 1, No. 2, (2018), pp. 1-8

Comparative Analysis of Pathfinding Algorithms A *, Dijkstra, and BFS on Maze Runner Game

Silvester Dian Handy Permana[1], Ketut Bayu Yogha Bintoro[2], Budi Arifitama[3], Ade Syahputra[4]

[1,2,3,4] Informatics Study Program, Faculty of Creative Industry and Telematics, Universitas Trilogi
[1]handy@trilogi.ac.id, [2]ketutbayu@trilogi.ac.id, [3]budiarif@trilogi.ac.id, [4]adesyahputra@trilogi.ac.id,

Abstract

*Maze Runner game is a game that requires pathfinding algorithm to get to the destination with the shortest path. This algorithm is used in an NPC that will move from start node to destination node. However, the use of incorrect algorithms can affect the length of the computing process to find the shortest path. The longer the computing process, the longer the players have to wait. This study compared pathfinding algorithms A *, Dijkstra, and Breadth First Search (BFS) in the Maze Runner game. Comparison process of these algorithms was conducted by replacing the algorithm in the game by measuring the process time, the length of the path, and the numbers of block played in the existing computing process. The results of this study recommend which algorithm is suitable to be applied in Maze Runner Game.*

Keywords: *Pathfinding, A*, Dijkstra, BFS, Maze Runner*

1. Introduction

These days, AI have spread through the game world. Some games have started implementing AI in its game. Varying from action games, adventure, action adventure, RPG, simulation, strategy, sports, to idle game. AI spreads in all kind of growing game genre. The use of AI can bring revenue from the game. Players can feel challenged and buy some of the facilities provided by the game. It is not rare for the game that use AI to be better than games that does not use AI [1].

Maze Runner game is a labyrinth created by players. The player have to block the path of the NPC that has been given pathfinding algorithm with Tetris-like block. Tetris itself has a block like letter I, J, L, O, S, T, and Z in the game [2]. Players are provided with various blocks with these letters to block the course of the NPC. The more blocks traversed by the NPC, the more score the player will get. Therefore, the NPC needs the fastest path to the destination node[3].

Pathfinding algorithm is required to determine the fastest path possible. There are several algorithms to find the fastest path. The algorithms are A*, Dijkstra, and Breadth First Search (BFS). These algorithms are the best pathfinding algorithms [4]. Each of these algorithms has their own weaknesses and strengths in the process of determining the fastest path. This research will find the best algorithm for the Maze Runner game.

The A Star or A* algorithm is one of the search algorithms that analyze inputs, evaluates a number of possible paths and generate solutions. The A* algorithm is a computer algorithm that is used extensively in graph traversal and the search for paths along with the efficient path planning process around the points called nodes [5]. Dijkstra's algorithm uses the principle of greedy, in which each step is selected with the minimum weights that connect a selected node with another unselected node [6]. While Breadth First Search is a method that performs a wider search that extends a node pre-orderly, extending a node and then extending all neighbors of the last node. After that, extend the unextended nodes and is neighboring with the extended nodes, and so on [7].

ISSN : 2580-7250

The comparison process in this algorithm was conducted by measuring 3 variables of pathfinding computation. These three variables are the measurement of the process time, the length of the path, and the numbers of block played in the existing computing process. In the process of measuring time, time will be measured in milliseconds (ms). The calculated time is the time needed from start node to destination node. The time will start when NPC starts running. The length of the path is measured by counting the number of blocks traversed by the NPC. Blocks played in the computation process are calculated from the number of blocks (nodes) counted before the pathfinder determines its path.

The purpose of this research will determine the best algorithm to be applied in this Maze Runner game. The algorithm which will be recommended to be applied in the game is considered to be taken from various aspects of the comparison test results. The results of this research are expected to help game developers to implements the best algorithm for making maze-based games.

2. Rudimentary

Research conducted by Aqsa Zafar et al with the title, "Analysis of Multiple Shortest Path Finding Algorithm in Novel Gaming Scenario" discusses the comparison of 5 pathfinding algorithms. The five algorithms are Breadth-First Search (BFS), Depth-First Search (DFS), Best-First Search, Dijkstra's Algorithm, and A* Algorithm. Although using these five pathfinding algorithms, the discussion in the research only compare Dijkstra's Algorithm and A * Algorithm with comparative table. In the discussion, game images from Age of Empire and Civilization V are provided but not yet clearly explained about the use and the application of the game. What can be drawn from this research is the attention to the A* algorithm make it seen as the best algorithm compared to the others [8].

Research with the title, "Pathfinding Algorithm Efficiency Analysis in 2D Grid" done by Zarembo and Sergejs compares A*, BFS, Dijkstra, HPA*, and LPA* in two-dimensional sized grids. BFS has the slowest results. This result can be explained by the fact that the algorithm's operating principle is very simple and it does not use any heuristics. Dijkstra's algorithm is faster than BFS, but slower than the other algorithms. A* and LPA* have the same value of performances. LPA* is faster in smaller grids (64, 128, 256), but A* is faster in larger grids (512, 1024). Which leads to the conclusion, that LPA* is better suited for smaller pathfinding problems, while A* is better used to solve larger problems [9].

Research conducted by Singhal and Harish with the title, "A Review of the Paper of Navigation and Pathfinding using Mobile Cellular Automata" describes pathfinding, cellular automata and software agents. This research studied the pathfinding category and how is it used in finding the distance between 2 nodes. This research focuses on cellular automata and for complex modeling systems and computing tools. This research uses Breadth-first search, Dijkstra's algorithm, and A* to find the distance between the cellular automata [10].

The study titled, "Pathfinding car racing game using dynamic pathfinding algorithm and algorithm A*" conducted by Sazaki, et.al. used pathfinding algorithm for NPC to race against player in car racing game. The pathfinding method used by NPCs in this game is A* algorithm to find the shortest path on the track and combined with Dynamic Pathfinding Algorithm to avoid static or dynamic obstacles in its path. The experimental results in this study show that the combination of both methods can be implemented well in car racing games with the track conditions is blocked with static obstacles. While moving on the track with dynamic obstacles, the combination of both methods passes through the course only under certain conditions [11].

3. Research Methodology

The problem appointed in this research is the shortest and most efficient route search in Simulation Game especially Maze Runner. The research methodology used in this research is a comparative test using 3 levels in a Maze Runner game with the same obstacle positions for each tested algorithm. The case measured in this comparative test are the process time, the length of the path, and how many blocks being played in the existing computational process. In this research, researcher will use 3 different levels with different Tetris blocks at each level. After completing the test with 3 different levels, it can be decided which algorithm is best to be implemented in this Maze Runner game.

4. Result and Discussion

Maze Runner is a game that asks its players to block the path of NPC (Non Player Character) from the start node to destination node. This game has rules and freedom in the game. Players are prohibited from fully blocking (closing access) the path or imprisoning the NPC at the start node, this is so that the NPC can run from the start node to the destination node. Players can only interfere with course taken by the NPC. Players are given a limited number of obstacles to be arranged freely within the maze. The obstacles are Tetris block that looks like the letters I, J, L, O, S, T, and Z. The obstacle can be arranged by dragging the obstacles from its column to the arena / maze. Before being placed into the maze, players can also rotate the obstacle by 90 degrees with the rotate button. After all the obstacles are placed, players can press the GO button to give command for the NPC to walk towards the destination node. Figure 1 shows the Maze Runner game that asks its players to place all of the obstacles into the given map.

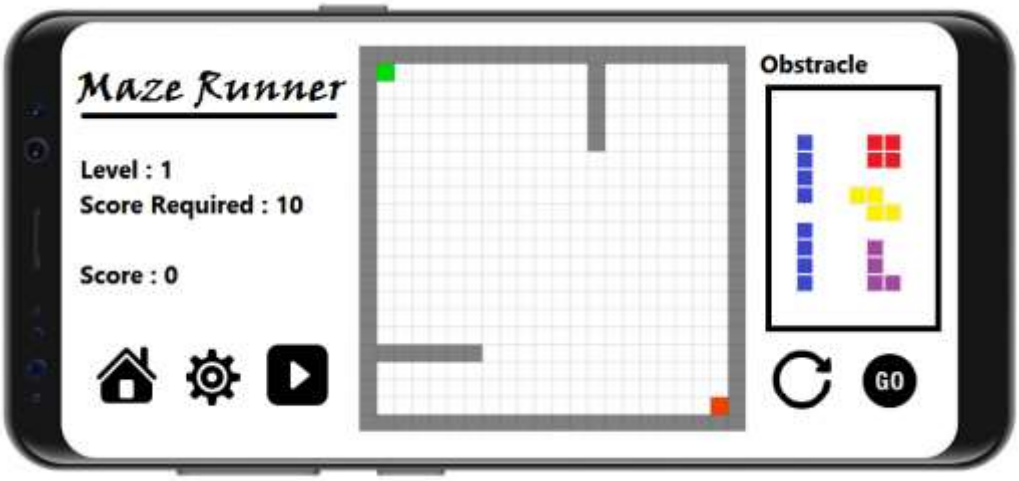

Figure 1. Maze Runner Interface

Maze Runner game has various levels. Each level has different obstacles and different score requirement to pass the level. The Next button will be enabled if the player can use all the obstacles and reach the scores required after pressing the GO button. When the GO button is pressed, the NPC (Green box) will then search for the shortest path to the destination node (Red box) with the applied algorithm. The score will be calculated from the number of squares used by the NPC as the path to the destination node.

In this research, we compared the use of the A*, Dijkstra, and Breadth First Search (BFS) algorithms applied to the NPC to find a way to the destination node. This research will use level 1, 2, and 3 to do comparison for each algorithm with the same obstacle layout at each level. Experiments conducted at level 1, 2, and 3 will arbitrarily place all of

the obstacles between the start node and the destination node in order to block the path of the NPC.

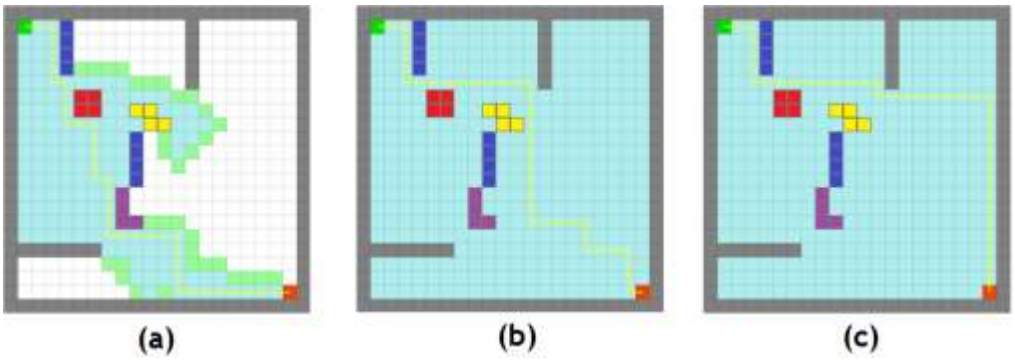

Figure 2. Pathfinding of Maze Runner Game at Level 1 using A* (a), Dijkstra (b), and BFS (c) algorithms

Figure 2 above is the computation results for Maze Runner game at level 1. Obstacles are arranged in such a way that the path of the NPC, which should only be diagonally straight to the destination, becomes slightly turned aside. Figure 2a shows that the NPC moves using A* and 2b algorithms shows that the NPC moves using Dijkstra algorithm. While Figure 2c shows that the NPC moves using BFS algorithm. The calculation results of these three algorithms can be seen in table 1 below.

Table 1. Comparison of Algorithms Computation on the First Level of Maze Runner Game

Components	A* Algorithm	Dijkstra's Algorithm	BFS Algorithm
Length	38	38	38
Time	0.3500 ms	0.3000 ms	0.8000 ms
Computed Blocks	323 blocks	738 blocks	738 blocks

From Table 1 above, the same results of 38 blocks traveled were obtained from the distance traveled by NPC to the destination node. With these three algorithms, players will get a score of 38. The computational time required by Dijkstra's algorithm is faster than A* and BFS. A* has 0.0500 ms longer computation time than Dijkstra. The computational process of A* algorithm is slightly slower than the Dijkstra and BFS algorithms. It can be deduced that A* algorithm consumes less memory for its computation process than any other algorithm.

At level 2 of the Maze Runner game, players are given more obstacles to interfere with the process of finding the shortest path for the NPC, but the distance between both nodes is closer. Players are given 17 pieces of Tetris-like block. There are 5 pieces of block that looks like the letter O, 6 pieces looks like the letter I, 4 pieces looks like the letter L, a single piece looks like the letter J, and another single piece that looks like the letter Z. At level 2, the minimum score required to unlock level 3 is 30 points. This means that it takes a minimum of 30 NPC steps to achieve that score. The three algorithms are tested after all the obstacles are placed in the maze. Pathfinding performed by the three algorithms can be seen in Figure 3.

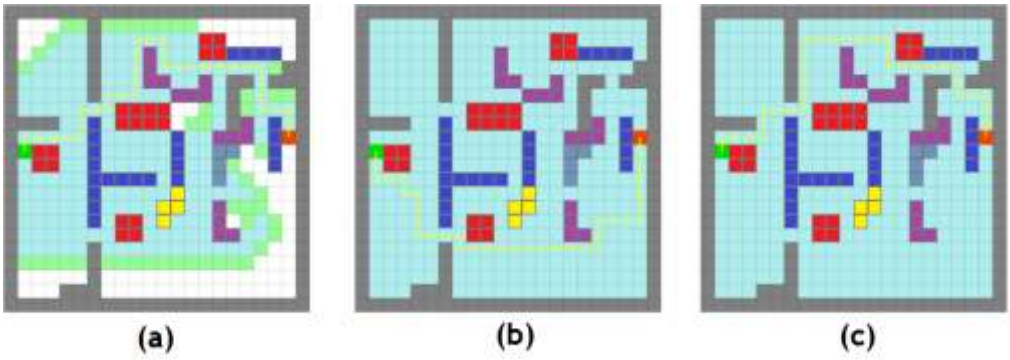

Figure 3. Pathfinding of Maze Runner Game at Level 2 using A* (a), Dijkstra (b), and BFS (c) algorithms

In Figure 3 above, A* algorithm has a path similar to the BFS algorithm. Although similar, the path taken by A* algorithm is closer to the border of the placed obstacles while the BFS algorithm is closer to the walls by default. This proves that A* algorithm extends the arranged obstacles and crawls to the goal.

Table 2. Comparison of Algorithms Computation on the Second Level of Maze Runner Game

Component	A* Algorithm	Dijkstra Algorithm	BFS Algorithm
Length	34	34	34
Time	0.4000 ms	0.6000 ms	0.8000 ms
Computed Blocks	415 blocks	618 blocks	618 blocks

From Table 2 above, the same results of 34 blocks were obtained from the number of squares traveled by the NPC to get to the destination node and thus, give a score of 34 to the player. The computation time difference between the three algorithms is 0.2000 ms. The A* algorithm has the fastest computation time than the other algorithms. The computational process needed for A* algorithm is less than Dijkstra and BFS algorithms with 415 blocks. Both Dijkstra and BFS algorithms extend 618 blocks to search for the shortest path.

At level 3 of Maze Runner game, players are given even tougher challenges. The start and destination node at this third level is located adjacently and only separated by a wall. The player is required to arrange the obstacles so that the NPC moves at least 50 steps. Players are given 15 pieces of Tetris-like blocks. There are 6 pieces of block that looks like the letter O, 2 pieces looks like the letter I, 2 pieces looks like the letter L, a single piece looks like the letter J, 2 pieces that looks like the letter S, a single piece looks like the letter T, and another single piece looks like the letter Z. The three algorithms are then being tested after all the obstacles are placed in the maze. The test results of the algorithms are shown in figure 4.

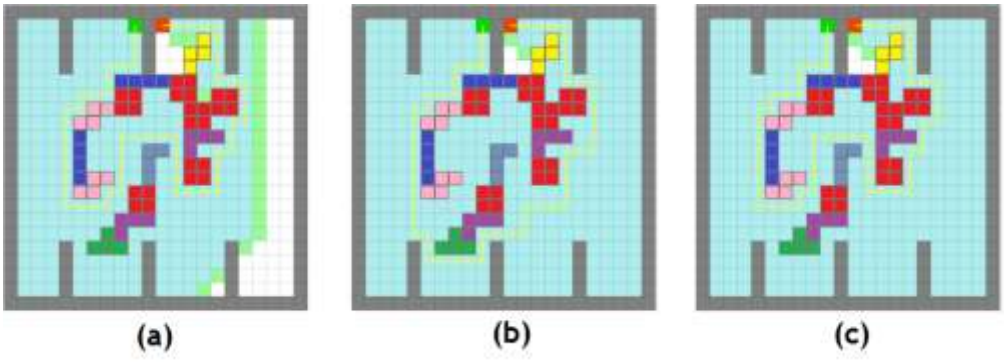

Figure 4. Pathfinding of Maze Runner Game at Level 3 using A* (a), Dijkstra (b), and BFS (c) algorithms

In Figure 4 above, A* algorithm has the same path as BFS algorithm. Although both has the same path taken, A* algorithm has fewer computed blocks than BFS algorithm. While Dijkstra algorithm has a unique path that is goes through the bottom route even though the number of blocks on both routes are the same. This is because at the time of computation, Dijkstra's algorithm meets the last node in the middle of the upper route. Therefore, the extended path is the bottom path and the algorithm reaches the destination node with the lower path.

Table 3. Comparison of Algorithms Computation on the Third Level of Maze Runner Game

Component	A* Algorithm	Dijkstra Algorithm	BFS Algorithm
Length	58	58	58
Time	1.1000 ms	0.9000 ms	1.0000 ms
Computed Blocks	504 blocks	624 blocks	624 blocks

The number of blocks required for the NPC to reach the destination node on the three algorithms is the same and the player scores 58. The computation time difference between the three algorithms is 0.1000 ms. Dijkstra's algorithm is faster than any other algorithm. In terms of computation, the A* algorithm is slightly slower than Dijkstra and BFS Algorithms with 504 blocks or a difference of 120 computed blocks. Dijkstra and BFS algorithms extend 624 blocks to search for the shortest path.

5. Conclusion

Based on the results of this research and evaluation conducted, it can be concluded that:

1. A*, Dijkstra, and Breadth First Search can be used to find the shortest in Maze Runner Game.
2. A* is the best algorithm in pathfinding especially in Maze game / grids. This is supported by the minimal computing process needed and a relatively short searching time.
3. The use of the right algorithm can make the game better in terms of computing process, memory usage, and computing time.

6. Suggestion

Some suggestions that can be taken from this research are:

1. To continue this research with a modified A* algorithm to get better computation results.
2. Make a test for A* to find the shortest path in other cases such as racing game, strategy, sport, etc.

References

[1] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General Video Game AI: a Multi-Track Framework for Evaluating Agents, Games and Content Generation Algorithms," 2018.

[2] H. M. Vo, "Optimizating Power Consumption Using Multi-Bit Flip-Flop Technique," pp. 0–3, 2017.

[3] B. Arifitama and A. Syahputra, "Implementing Augmented Reality on a Multiple Floor Building as a Tool for Sales Product Knowledge," Int. J. Technol. Bus., vol. 1, no. 1, 2017.

[4] S. M. Kim, M. I. Peña, M. Moll, G. N. Bennett, and L. E. Kavraki, "A review of parameters and heuristics for guiding metabolic pathfinding," J. Cheminform., vol. 9, no. 1, pp. 1–13, 2017.

[5] H. Reddy, "PATH FINDING - Dijkstra ' s and A * Algorithm ' s," pp. 1–15, 2013.

[6] P. Singal Dcsa and R. R. S. Chhillar, "Dijkstra Shortest Path Algorithm using Global Positioning System," Int. J. Comput. Appl., vol. 101, no. 6, pp. 975–8887, 2014.

[7] A. A. Jamal and W. J. Teahan, "Alpha multipliers breadth-first search technique for resource discovery in unstructured peer-to-peer networks," Int. J. Adv. Sci. Eng. Inf. Technol., vol. 7, no. 4, 2017.

[8] A. Zafar, K. K. Agrawal, W. Cdr, and A. Kumar, "Analysis of Multiple Shortest Path Finding Algorithm in Novel Gaming Scenario," Intell. Commun. Control Devices, Adv. Intell. Syst. Comput., pp. 1267–1274, 2018.

[9] I. Zarembo and S. Kodors, "Pathfinding Algorithm Efficiency Analysis in 2D Grid," Environ. Technol. Resour. Proc. Int. Sci. Pract. Conf., vol. 2, p. 46, 2015.

[10] P. Singhal and H. Kundra, "A Review paper of Navigation and Pathfinding using Mobile Cellular Automata," vol. 2, no. I, pp. 43–50, 2014.

[11] A. Sazaki, Yoppy, Primanita and M. Syahroyni, "Pathfinding Car Racing Game Using Dynamic Pathfinding Algorithm and Algorithm A *," 3rd Int. Conf. Wirel. Telemat. 2017, pp. 164–169, 2017.

Authors

Silvester Dian Handy Permana was born in Yogyakarta, Indonesia on November 26, 1990. He graduated from Universitas Atma Jaya Yogyakarta - Indonesia, with a degree in Informatics Engineering and later received a Master's degree in Information Technology from Universitas Indonesia - Indonesia.

He currently works for Universitas Trilogi as Informatics Lecturer, and have experience in lecturing for more than 4 years. His research interests include Game Application, Game Algorithm and Complexity, and Multimedia.

Ketut Bayu Yogha Bintoro was born in Jakarta, Indonesia on May 14, 1986. He graduated from Universitas Gunadarma - Indonesia, with a degree in Information System and later received a Master's degree in Computer Science from Universitas Gadjah Mada - Indonesia.
He currently works for Universitas Trilogi as Informatics Lecturer, and have experience in lecturing for more than 4 years. His research interests include Artificial Intelligence, Swarm Intelligence, Multi-Agent System, Intelligence Agent, and Complex Adaptive System.

Budi Arifitama was born in Rome, Italy on May 23, 1986. He graduated from Universitas Gunadarma - Indonesia, with a degree in Informatics Engineering and later received a Master's degree in Management Information System from Universitas Gunadarma.
He currently works for Trilogi University as Informatics Lecturer, and have experience in lecturing for more than 4 years. His research interests include Augmented Reality, Human Computer Interaction, and Multimedia.

Ade Syahputra was born in Medan, Indonesia on May 08, 1983. He graduated from Universitas Gunadarma - Indonesia, with a degree in Informatics Engineering and later received a Master's degree in Information and Communications Technology Management from University of South Australia, Adelaide.
He currently works for Trilogi University as Informatics Lecturer, and have experience in lecturing for more than 5 years. His research interests include Augmented Reality, Human Computer Interaction, and Supply Chain Management.