JULY 2021 UNIVERSITY OF SCIENCE, VNU - HCM

A* Algorithm

Shortest pathfinding algorithm

Prepared by

- 19127216 - Đặng Hoàn Mỹ
- 19127544 - Nguyễn Hoàn Hoài Tâm
- 19127609 - Đinh Quang Tú

Instructed by

Teacher Đinh Bá Tiến
Teacher Hồ Tuấn Thanh
Teacher Nguyễn Lê Hoàng Dũng
Teacher Trương Phước Lộc

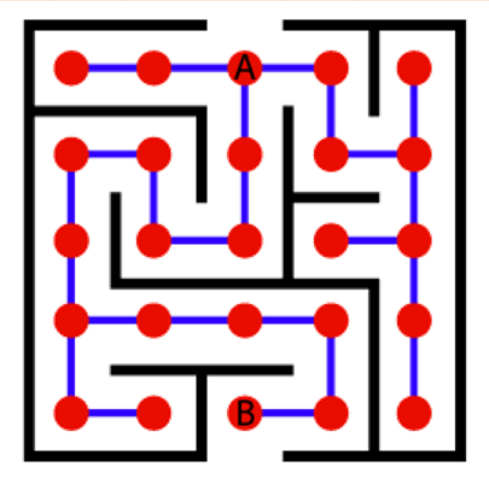
Table of Content

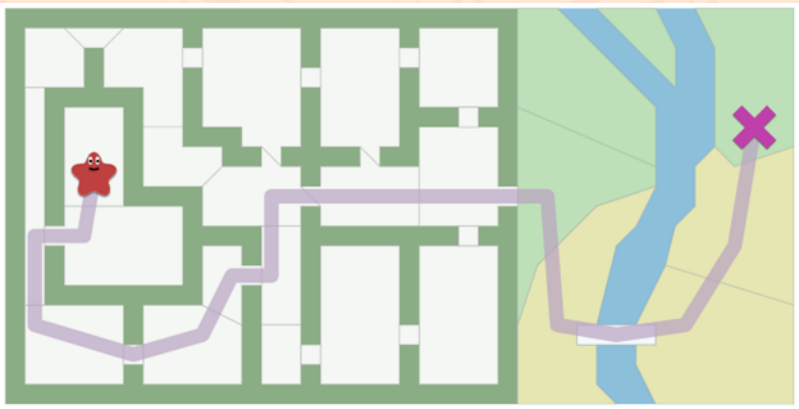
Introduction....3
History....4
Why we call it A*?....5
Terminology....6
Data Structure & Algorithm....8
Implement....9
Complexity....10
Graph Example....11
Comparison....12
Application....13
List of questions for Quiz....14
Source Code....17
Work Assignments....18
References....19

Introduction

The maze has become a classic game in the gaming world. There are thousands of games inspired by the maze. And the main task in which is probably mostly the player finding the way out of the maze in the shortest amount of time. There are many ways to complete this game. Many people choose to move forward without pre-calculating the path. However, in this way, when encountering an obstacle, the player will have to waste time to return and find a new path. Some people use pre-calculation of steps to find the most optimal path to the destination. Although pre-calculation can take quite a while, it will help players avoid obstacles.
From the 2 to play above we can see that it is important to make a plan before we move. Although they are usually time-consuming initially they are not trapped whereas moving first can be faster but fall into a trap.

The A* algorithm was born for that reason. In games, we often want to find paths from one location to another. We're not only trying to find the shortest distance; we also want save as much time as possible.



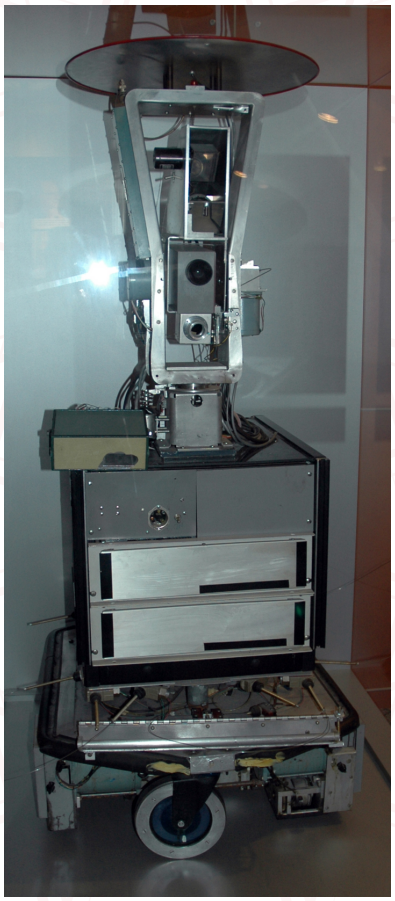


History

A achieves better performance by using heuristics to guide its search.*

Created as part of the Shakey project aimed to build a mobile robot that has artificial intelligence to plan its actions, A* was initially designed as a general graph traversal algorithm. Originally Nills Nilsson suggested using the Graph Traverser algorithm for Shakey's path planning. Graph Traverser is guided by a heuristic function h(n), the estimated distance from node n to the goal node: it entirely ignores g(n), the distance from the start node to n. Different from Nills Nilsson, Raphael suggested using the sum, g(n) + h(n). Peter Hart invented the concepts we now call admissibility and consistency of heuristic functions. A* was originally designed for finding least-cost paths when the cost of a path is the sum of its costs, but it has been shown that A* can be used to find optimal paths for any problem satisfying the conditions of a cost algebra. Because of its flexibility and versatility, it can be used in a wide range of contexts.

A* was developed in 1968 to combine heuristic approaches like Greedy Best-First-Search and formal approaches like Dijkstra's Algorithm. It's a little unusual in that heuristic approaches usually give you an approximate way to solve problems without guaranteeing that you get the best answer. However, A* is built on top of the heuristic, and although the heuristic itself does not give you a guarantee, A* can guarantee the shortest path.



Shakey the Robot in its display case at the Computer History Museum.

Why we call it A*?

The reason is that scientists first came up with an improved version of the Dijkstra algorithm they called A1. Later on, the inventors of A* discovered an improvement of A1 that they called A2. These people then managed to prove that A2 was actually optimal under some assumptions on the heuristic in use. Because A2 was optimal, it was renamed A*. In science, and in optimization in particular, a " * " symbol is often used to denote optimal solutions. Some also interpret the " * " as meaning "any version number" since it was proven impossible to build an "A3" algorithm that would outperform A2/A*.

By the way, in this context, "optimal" doesn't mean that it reaches the optimal solution, but that it does so while exploring the minimum number of nodes. Of course, A* is also complete, which means it reaches the optimal solution (if we use an consistent heuristic).

Terminology

What is heuristic?

Heuristic value h is the estimated cost of moving from the current state to the goal state.

f is the parameter of A* which is the sum of the other parameters G and H and is the least cost from one node to the next node. This parameter is responsible for helping us find the most optimal path from our source to destination. f = g + h

+ g is the cost of moving from one node to the other node. This parameter changes for every node as we move up to find the most optimal path.

+ h is the heuristic/estimated path between the current code to the destination node. This cost is not actual but is, in reality, a guess cost that we use to find which could be the most optimal path between our source and destination.

What are consistency and admissibility?

- The condition required for optimality when using A* tree search is known as **admissibility**.
- An **admissible heuristic** is used to estimate the cost of reaching the goal state in an informed search algorithm. In order for a heuristic to be admissible to the search problem, the estimated cost must always be lower than or equal to the actual cost of reaching the goal state.

$$\forall n,\ 0 <= h(n) <= h^*(n)$$

- To maintain completeness and optimality under A* graph search, we need an even stronger property than admissibility, **consistency**.
- A **consistent heuristic** is always less than or equal to the estimated distance from any neighboring vertex to the goal, plus the cost of reaching that neighbor.

$$h(A) <= cost(A, C) + h(C)\ \ \forall A, C\ with\ C\ is\ a\ successor\ of\ A$$

Terminology

A∗ is a best-first search starting from a specific starting node of a graph, it aims to find a path to the given goal node having the smallest cost.

Completeness and optimality of A* Algorithm

Completeness

The completeness of each search strategy - if there exists a solution to the search problem, is the strategy guaranteed to find it given infinite computational resources?

And the A∗ Algorithm is complete if all the step costs exceed finitely and branching factor b is finite.

Optimality

The optimality of each search strategy - is the strategy guaranteed to find the lowest cost path to a goal state.

The A∗ Algorithm is optimal in graph-search when the h(n) is consistent.

Whenever A∗ selects a node n for expansion, the optimal path to that node has been found. With n' is a successor of n,

- There would have to be another frontier node n′ on the optimal path from the start node to n (by the graph separation property).
- f is non-decreasing along with any path → f(n′) < f(n) → n′ would have been selected first.

Data Structure & Algorithm

I decided to approach this algorithm by using frontier.

This frontier is a min-heap for a priority queue - it keeps the smallest value at the top. It is suitable for the frontier to keep track of the node that may be traversed in the next steps.

This min-heap priority queue uses the min-heap data structure which supports operations such as insert, minimum, extract-min, decrease-key. In this implementation, the weight of the edges is used to decide the priority of the vertices. Lower the weight, higher the priority and higher the weight, lower the priority.

A* uses the heuristic to reorder the nodes so that it's more likely that the goal node will be encountered sooner. And the priority queue will rearrange that to get the best node that we can visit next.

The f(n) formula used for organizing the value in the frontier is:

- f is the parameter of A* which is the sum of the other parameters G and H and is the least cost from one node to the next node. This parameter is responsible for helping us find the most optimal path from our source to destination. f = g + h

+ g is the cost of moving from one node to the other node. This parameter changes for every node as we move up to find the most optimal path.
+ h is the heuristic/estimated path between the current code to the destination node. This cost is not actual but is, in reality, a guess cost that we use to find which could be the most optimal path between our source and destination.

Implement

The main technique for this algorithm is using frontier as a priority queue to put the nodes while waiting. Then get the smallest cost of a path from that and put in the visited to mark up what has been visited. When we catch the goal, we start to reverse the path (the parents of visited nodes) that we saved in the frontier. Then, we return the path of that node. If there is no goal, we traverse all the children of that node and put it in the frontier with the formula f(n) = g(n) + h(n).

```
visited = [ ] //save the visited node in this, some other paper could mention it as expanded states
path = [ ] // the path will return
parents = [ ] // save the parents of the child node every time it find the path
frontier = <priority_queue> // priority queue which has the smallest on the top
push [ heuristics[START], START, -1 ] // the f value, the state, its parent
while frontier: // loop until the frontier is NOT empty
    heuristic, node, parent = top of frontier //get the smallest value f(node)
    pop the top
    if node not in visited:
        visited add node
        parents [node] = parent
        if node is GOAL: // if goal is reached, get the path and return
            while parents [node] ≠ 1:
                path add node
                node = parents [node]
            path add start
            reverse the path
            return path
    traverse all the child of the node using for
        calculate the f(child) = g(child) + h(child)
        frontier push [ f(child), child, node ]
```

Complexity

One major practical drawback is its $O(b^d)$ space complexity, as it stores all generated nodes in memory.

Time Complexity

The time complexity of A$*$ depends on the heuristic. In the worst case of an unbounded search space, the number of nodes expanded is exponential in the depth of the solution (the shortest path) d: $O(b^d)$, where b is the branching factor (the average number of successors per state).

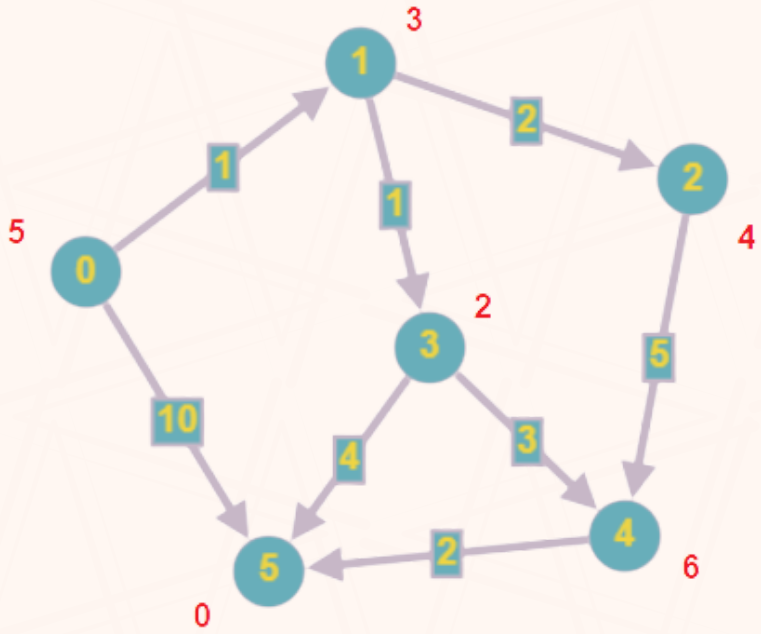
Space Complexity

The space complexity of A$*$ is roughly the same as that of all other graph search algorithms, as it keeps all generated nodes in memory.

$O(|V|) = O(b^d)$, V is the number of vertices in a graph. The worst case is traversing all of the vertices in the graph.

Graph Example

We start from 0 to 5 as a goal.



The formula is f(n) = g(n) + h(n), then we apply to the source: f(0) = 5 + 0 and put it to the frontier.

Next, there are 2 ways from 0, we put 0 to the visited and the frontier is 1 5.

- 0 -> 1 is f(1) = 1 + 3 = 4
- 0 -> 5 is f(5) = 10 + 0 = 10

Because 1 is on the top of the frontier, we put it in visited, then check the children of 1 because 1 is not the goal, the frontier will be 3 2 5.

- 1 -> 2 is f(2) = (cost from 0 to 2) 3 + 4 = 7
- 1 -> 3 is f(3) = (cost from 0 to 2) 2 + 2 = 4

Next, 3 is the top of the frontier, we put it in visited, then check the children because 3 is not the goal, the frontier now will be 5 (0 -> 1 -> 3 -> 5), 2, 5 (0 -> 5), 4.

- 3 -> 4 is f(4) = (cost from 0 to 4) 5 + 6 = 11
- 3 -> 5 is f(5) = (cost from 0 to 5) 6 + 0 = 6

Now, goal 5 is on the top of the frontier, we reverse the path after getting the parents of the nodes when we approach goal 5. Then we have the path which is **0, 1, 3, 5**.

Comparison

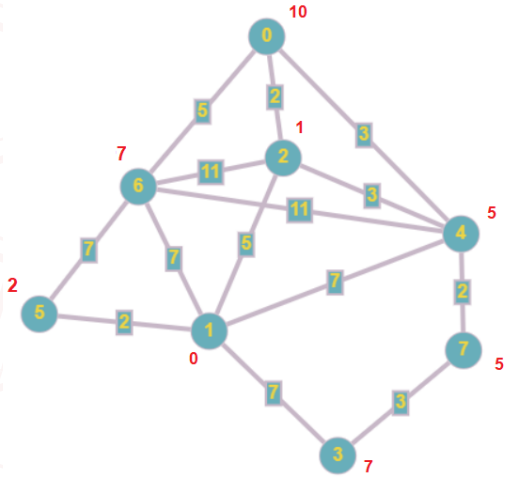
The major thing in A⋆ Algorithm is about heuristics which is important in how optimal of the solution path.

Better than Dijkstra's Algorithm with ADMISSIBLE heuristic function.

Dijkstra' Algorithm does not use a Heuristic Function to expand the node, it looks only for the path that minimizes the cost to reach an unvisited node directly connected to the nodes already visited, it will find the shortest path finally, but would have to explore all the nodes not directly connected to the sink. A⋆ with the help of a good Heuristic Function (must be an admissible Heuristic: never overestimates the cost to reach the goal) could reach immediately the sink always expanding the right node in the frontier. So if g(n)=cost to reach node n and h(n)=an admissible heuristic function we get f(n)=g(n)+h(n) the evaluation function used by A⋆. Instead, Dijkstra knows only its frontier and misses the Heuristic information, thus it works as well as A⋆ when heuristic is weak.

Experiment in Graph

Actually, the number of nodes expanded to arrange their children to the frontier and find out the path in Dijkstra and other algorithms which use a priority queue to represent its frontier.



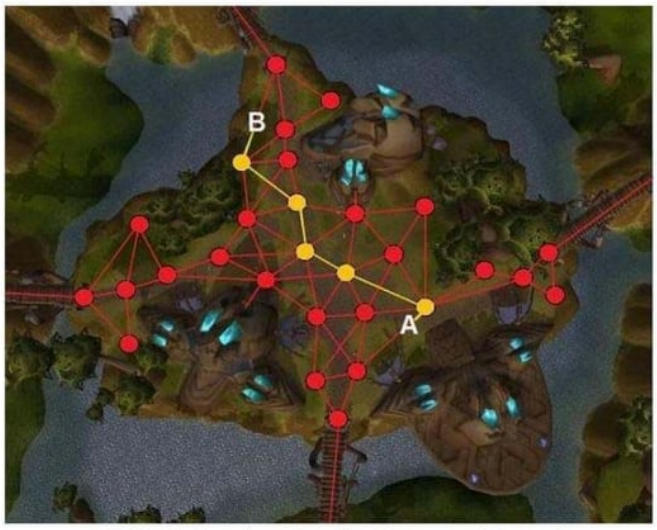
For example, with this graph, we check the path from 0 to 1:

- Dijkstra's Algorithm have to expand double number of nodes to find the same path. (0, 2, 4, 7, 6, 1)
- A⋆ Algorithm just expands only 3 nodes exactly the same path. (0, 2, 1)

In conclusion, A⋆ has done better work in finding the path, even the result is the same, the heuristics shown that how strong they are.

Application

- A* algorithm is an advanced BFS algorithm that searches for shorter paths first rather than longer paths. A* is optimal as well as a complete algorithm.
- A* Algorithm is one of the best and popular techniques used for pathfinding and graph traversals.
- A lot of games and web-based maps use this algorithm for finding the shortest path efficiently. One example of this is the very popular game- Warcraft III.

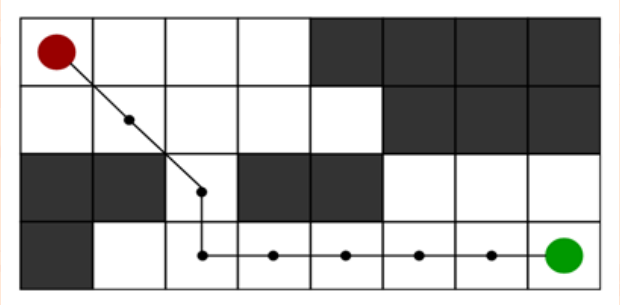


(a) Navigating from A to B using waypoint graph.

(b) Navigating from A to B on NavMesh.

The Pathfinder: GameplayKit Pathfinding Basics

This game automatically creates mazes then uses GameplayKit to solve them, highlighting the shortest path through each maze.



2D Grid having several obstacles and we start from the source red cell to reach towards a goal green cell

List of questions for Quiz

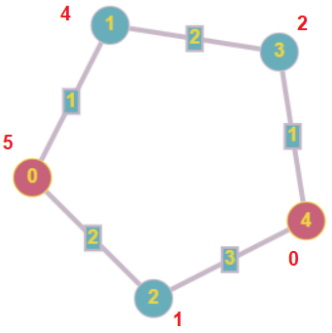
1. A* Algorithm does ...
    a. **path-finding and graph traversals, calculate from heuristics and path cost**
    b. finds shortest paths in a directed weighted graph with positive or negative edge weights (Floyd–Warshall Algorithm)
    c. finds the least-cost path from a given initial node to any goal node (B* Algorithm)
    d. explores as far as possible along each branch before backtracking (BFS)
2. In which project and in what year was the A* Algorithm first published?
    a. **Shakey project (1968)**
    b. Plankalkül programming language (1945) - BFS
    c. Richard Korf (1985) - IDA*
    d. Harpy Speech Recognition System (1976) - Beam
3. What is the meaning of A in A*?
    a. **Admissible**
    b. Acoustics
    c. Agree
    d. Awesome
4. What is heuristic value?
    a. **the estimated cost from the current vertex to the goal vertex**
    b. the smallest cost from the start vertex to the goal vertex
    c. the cost from the root node to the current node
    d. the value for the best solution to a given problem

List of questions for Quiz

5. The A* Algorithm is developed based on:
    a. Dijkstra's Algorithm
    b. BFS Algorithm
    c. IDA* Algorithm
    d. Floyd–Warshall Algorithm
6. What is the TIME and SPACE complexity of A* Algorithm?
    a. **O(b^d)**
    b. O(h)
    c. O(h) - O(|V|)
    d. O(V) - O(h)
7. What is NOT an application of A* Algorithm?
    a. **Google Maps**
    b. Warcraft III
    c. Graph Traversal
    d. n-puzzle Solving
8. What formula does the A* Algorithm calculate the path to the next node?
    a. **f(n) = g(n) + h(n)**
    b. f(n) = g(n) / h(n)
    c. g(n) = f(n) + h(n)
    d. h(n) = g(n) + g(n)
9. Is Dijkstra's Algorithm better than A* Algorithm when the heuristic function is admissible?
    a. **True**
    b. False

List of questions for Quiz

10. What is the path from 0 to 4 with A* algorithm?



a. **0 2 4**

b. 0 1 3 4

c. 0 4

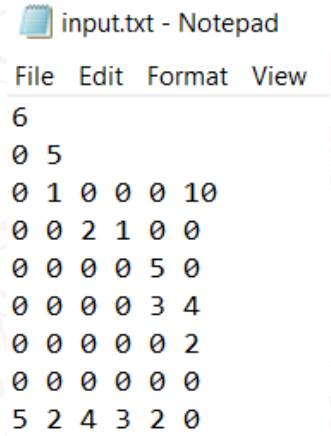
d. 0 3 4

Source Code

Input file - input.txt

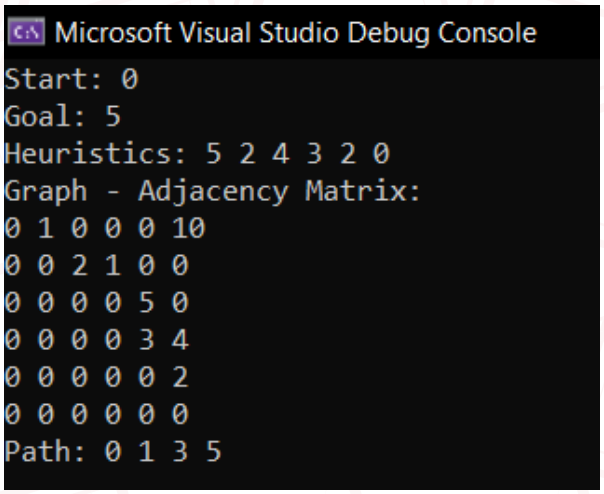
- The first line is the number of vertices in the graph n.
- The second line is 2 numbers, the first is the start vertex, then the goal vertex.
- The next n lines are the adjacency matrix of the graph.

The last line is the series of heuristics of the graph put in order.



Output in the console

- Each information that in the input file would be printed out and the path is also shown.



Work Assignments

- Find out the introduction and history
- Some terminologies (heuristic, f(n) = g(n) + h(n))
- Application

Quang Tú

- Notes for implementation, time/ space complexity
- Some terminologies (consistency and admissibility, completeness and optimality)
- Comparison
- Write source code (Graph)
- Graph demo (draw -> result)
- Find the questions for Quiz and put the questions on Kahoot
- Report composer and PowerPoint arranger

Hoàn Mỹ

- Template for PowerPoint and report

Hoài Tâm

References

Being grateful to these websites, videos and our courage to complete acknowledging A Algorithm.*

- A. (n.d.). GitHub - aliwalker/graph_search_algorithm: Some graph search algorithm. An assignment of Introduction to AI. GitHub. Retrieved July 31, 2021, from https://github.com/aliwalker/graph_search_algorithm
- Chatterjee, M. (2021, June 11). A* Search Algorithm in Artificial Intelligence (AI). GreatLearning Blog: Free Resources What Matters to Shape Your Career! https://www.mygreatlearning.com/blog/a-search-algorithm-in-artificial-intelligence/
- de Souza, J. V. A. (2014, August 28). Inteligent Control - A* Algorithm. Dudevictor GitHub IO. https://jsouza.dev/en/projects/controle-inteligente/?hcb=1
- Edpresso Team. (2021, July 30). What is the A* algorithm? Educative: Interactive Courses for Software Developers. https://www.educative.io/edpresso/what-is-the-a-star-algorithm
- Elgabry, O. (2018, May 28). Path Finding Algorithms - OmarElgabry's Blog. Medium. https://medium.com/omarelgabrys-blog/path-finding-algorithms-f65a8902eb40
- GameplayKit Programming Guide: Pathfinding. (2016, March 21). Apple Developer. https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/Pathfinding.html?hcb=1
- GeeksforGeeks. (2021, July 9). A* Search Algorithm. https://www.geeksforgeeks.org/a-search-algorithm/
- Tim, T. W. (2020, July 16). A* Pathfinding Visualization Tutorial - Python A* Path Finding Tutorial. YouTube. https://www.youtube.com/watch?v=JtiK0DOeI4A&feature=youtu.be

References

Being grateful to these websites, videos and our courage to complete acknowledging A Algorithm.*

- Leo Willyanto Santoso, Alexander Setiawan, Andre K. Prajogo. (2010). Performance Analysis of Dijkstra, A* and Ant Algorithm for Finding Optimal Path. http://fportfolio.petra.ac.id/user_files/04-021/MICEEI2010.pdf
- Red Blob Games: Introduction to A*. (n.d.). Red Blob Games. Retrieved July 31, 2021, from https://www.redblobgames.com/pathfinding/a-star/introduction.html
- Wikipedia contributors. (2021a, May 29). A* search algorithm. Wikipedia. https://en.wikipedia.org/wiki/A*_search_algorithm
- Wikipedia contributors. (2021b, July 26). Priority queue. Wikipedia. https://en.wikipedia.org/wiki/Priority_queue
- What does the star in the A* algorithm mean? (2016, March 5). Stack Overflow. https://stackoverflow.com/questions/35817230/what-does-the-star-in-the-a-algorithm-mean
- Dijkstra's Algorithm vs A* Algorithm detailed comparison. (n.d.). Slant. https://www.slant.co/versus/11584/11585/~dijkstra-s-algorithm_vs_a-algorithm
- Sharma, N. (n.d.). Note 1. In Introduction to Artificial Intelligence (pp. 8–13). EECS Instructional and Electronics Support - University of California, Berkeley. https://inst.eecs.berkeley.edu/~cs188/fa19/assets/notes/note01.pdf
- Why is A* faster than Dijkstra. (2014, May 25). Stack Overflow. https://stackoverflow.com/questions/23853559/why-is-a-faster-than-dijkstra