Nicholas Swift

329 Followers About Follow

Easy A* (star) Pathfinding

Nicholas Swift Feb 28, 2017 · 6 min read

Today we'll being going over the A* pathfinding algorithm, how it works, and its implementation in pseudocode and real code with Python 🐍.

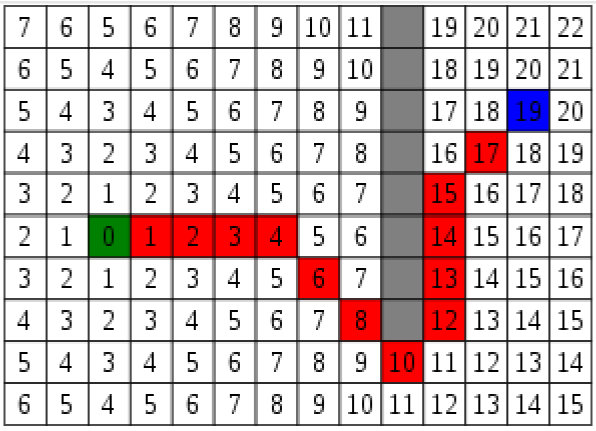
Looking for just pseudocode or source code? Scroll down!

If you're a game developer, you might have always wanted to implement A* as character (or enemy) pathfinding. I know a couple of years ago I did, but with my level of programming at the time I had issues with the current articles out there. I wanted to write this as an easy introduction with a clear source code example to A* pathfinding so anyone can easily pick it up and use it in their game.

F = G + H

One important aspect of A* is `f = g + h`. The f, g, and h variables are in our Node class and get calculated every time we create a new node. Quickly I'll go over what these variables mean.

- F is the total cost of the node.
- G is the distance between the current node and the start node.
- H is the heuristic — estimated distance from the current node to the end node.



7	6	5	6	7	8	9	10	11		19	20	21	22
6	5	4	5	6	7	8	9	10		18	19	20	21
5	4	3	4	5	6	7	8	9		17	18	19	20
4	3	2	3	4	5	6	7	8		16	17	18	19
3	2	1	2	3	4	5	6	7		15	16	17	18
2	1	0	1	2	3	4	5	6		14	15	16	17
3	2	1	2	3	4	5	6	7		13	14	15	16
4	3	2	3	4	5	6	7	8		12	13	14	15
5	4	3	4	5	6	7	8	9	10	11	12	13	14
6	5	4	5	6	7	8	9	10	11	12	13	14	15

wow such numbers, very color

Awesome! Let's say `node(0)` is our starting position and `node(19)` is our end position. Let's also say that our current node is at the the red square `node(4)` .

G

> *G is the distance between the current node and the start node.*

If we count back we can see that `node(4)` is 4 spaces away from our start node.

We can also say that G is 1 more than our parent node (`node(3)`). So in this case for `node(4)` , `currentNode.g = 4` .

H

> *H is the heuristic — estimated distance from the current node to the end node.*

So let's calculate the distance. If we take a look we'll see that if we go over 7 spaces and up 3 spaces, we've reached our end node (`node(19)`).

Let's apply the Pythagorean Theorem! `a² + b² = c²` . After we've applied this, we'll see that `currentNode.h = 7² + 3²` . Or `currentNode.h = 58` .

But don't we have to apply the square root to 58? Nope! We can skip that calculation on every node and still get the same output. Clever!

With a heuristic, we need to make sure that we can actually calculate it. It's also very important that the heuristic is always an underestimation of the total path, as an

F

> *F is the total cost of the node.*

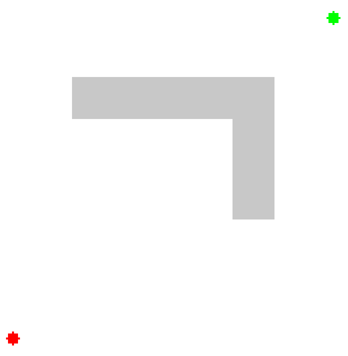
So let’s add up h and g to get the total cost of our node. `currentNode.f = currentNode.g + currentNode.h`. Or `currentNode.f = 4+ 58`. Or `currentNode.f = 62`.

Time to use `f` = g + h

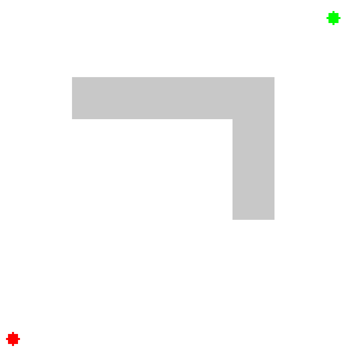
Alright, so that was a lot of work. Now with all that work, what am I going to use this `f` value for?

With this new `f` value, we can look at all our nodes and say, “Hey, is this the best node I can pick to move forward with right now?”. Rather than running through every node, we can pick the ones that have the best chance of getting us to our goal.

Here’s a graphic to illustrate. On top, we have Dijkstra’s Algorithm, which searches without this `f` value, and below we have A* which does use this `f` value.



Dijkstra’s Algorithm (Wikipedia)



Dijkstra's Algorithm

So taking a look at Dijkstra's algorithm, we see that it just keeps searching. It has no idea which node is '*best*', so it calculates paths for them all.

A* Algorithm

With A*,we see that once we get past the obstacle, the algorithm prioritizes the node with the lowest `f` and the '*best*' chance of reaching the end.

A⭐ Method Steps — from Patrick Lester

I've pasted the steps for A* from Patrick Lester's article that you can check out here. The same website is also listed below in resources. This is an insanely good explanation, and is why I decided to go with it rather than writing it again.

1. Add the starting square (or node) to the open list.

2. Repeat the following:

A) Look for the lowest F cost square on the open list. We refer to this as the current square.

B). Switch it to the closed list.

C) For each of the 8 squares adjacent to this current square …

- If it is not walkable or if it is on the closed list, ignore it. Otherwise do the following.
- If it isn't on the open list, add it to the open list. Make the current square the parent of this square. Record the F, G, and H costs of the square.
- If it is on the open list already, check to see if this path to that square is better, using G cost as the measure. A lower G cost means that this is a better path. If so, change the parent of the square to the current square, and recalculate the G and F scores of the square. If you are keeping your open list sorted by F score, you may need to resort the list to account for the change.

- Add the target square to the closed list, in which case the path has been found, or
- Fail to find the target square, and the open list is empty. In this case, there is no path.

3. Save the path. Working backwards from the target square, go from each square to its parent square until you reach the starting square. That is your path.

Pseudocode

Following the example below, you should be able to implement A* in any language.

```
// A* (star) Pathfinding

// Initialize both open and closed list
let the openList equal empty list of nodes
let the closedList equal empty list of nodes

// Add the start node
put the startNode on the openList (leave it's f at zero)

// Loop until you find the end
while the openList is not empty

    // Get the current node
    let the currentNode equal the node with the least f value
    remove the currentNode from the openList
    add the currentNode to the closedList

    // Found the goal
    if currentNode is the goal
        Congratz! You've found the end! Backtrack to get path

    // Generate children
    let the children of the currentNode equal the adjacent nodes

    for each child in the children

        // Child is on the closedList
        if child is in the closedList
            continue to beginning of for loop
        child.f = child.g + child.h

        // Child is already in openList
        if child.position is in the openList's nodes positions
            if the child.g is higher than the openList node's g
                continue to beginning of for loop

        // Add the child to the openList
        add the child to the openList
```

Source Code (in Python 🐍)

Feel free to use this code in your own projects.

Update: Please see the comments on my gist here, and a fork of my gist here — It includes bug fixes that are present in the code below.

Resouces

There are some awesome websites below you should check out. I especially recommend A* Pathfinding for Beginners.

A* Pathfinding for Beginners

The first thing you should notice is that we have divided our search area into a square grid. Simplifying the search...

www.policyalmanac.org

A* search algorithm - Wikipedia

Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute (now SRI International) first described the...

en.wikipedia.org

Searching using A* (A-Star)

"Coloring" a node means marking it to show we've gone there. This prevents a search from searching the same node more...

web.mit.edu

Programming Algorithms Python Game Development Data Structures