

 [BigZaphod](#) / [AStar](#)


Join GitHub today






[Dismiss](#)

GitHub is home to over 20 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)

C Implementation of the A* Pathfinding Algorithm

 **8** commits **1** branch **0** releases **2** contributorsBranch: **master** ▾[New pull request](#)[Find file](#)[Clone or download ▾](#)**BigZaphod** Merge pull request [#3](#) from rofl0r/patch-1 ...Latest commit [f5f3ef1](#) on 17 Apr 2017

 OSX Demo	Validated with Xcode 4.4.	6 years ago
 .gitignore	Added ignores	6 years ago
 AStar.c	include stdint.h	10 months ago
 AStar.h	Messed with some data types and added goalNode as a parameter to the ...	6 years ago
 README.md	Initial commit.	6 years ago

 [README.md](#)

A*

This is my implementation of A* in C. It uses a binary heap to implement the priority queue and an indexed array for fast lookups of previously visited nodes.

Overview

To use it, you must define 3 functions which provide information about node connectivity, sorting, and heuristics. A "node" for purposes of this implementation is simply a pointer to any data you want. You must also specify the size of the node data you're pointing to. If you're just using pointers, the size is simply something like `sizeof(MyNodeStruct *)`, however this implementation is capable of storing whole structures (such as `{int x; int y;}`) as well.

To find a path, first populate a `ASPathNodeSource` structure with the relevant pointers and node data size and then call `ASPathCreate()` with a start and goal node. Any context pointer passed into `ASPathCreate()` will be passed along to the various callback functions so you can use that to access your map or whatever you need.

The result of `ASPathCreate()` is an `ASPath` structure which stores the resulting path (if any). If there's no path, the `ASPathGetCount()` will return 0 and `ASPathGetCost()` will return `INFINITY`. You must call `ASPathDestroy()` when you're done with the resulting path or else you will leak memory. The `ASPath` structure does not store any reference to the original `ASPathNodeSource` used to make it. It is entirely self-contained and may be copied with `ASPathCopy()`.

`ASPathNodeSource.nodeComparator()` must return -1, 0, 1 in such a way that the given nodes will be sorted in some order (the exact order such as ascending or descending, etc. is unimportant). This works just the same as any typical C sorting function should. This function is used when accessing the internal index to lookup previously visited nodes.

`ASPathNodeSource.nodeNeighbors()` is called whenever a node is visited. You are expected to use `ASNeighborListAdd()` to add new nodes to the list of possible neighbors for the given node and the cost to move from the given node to that new neighbor.

`ASPathNodeSource.pathCostHeuristic()` must return the "best guess" for how far away the two nodes are from each other. This is the cost heuristic. Please read up on how A* works to know more about this, but for a simple 2D grid this function typically computes something as simple as the Manhattan distance between the two given nodes.

This implementation knows nothing about coordinates, grids, or anything spatial. It is up to you, by way of the callback functions, to indirectly supply this information to the algorithm when generating neighbors for a given node. The implementation makes no assumptions about the shape of your data or what you might be using it for.

Author

Created by Sean Heber (Twitter: [@BigZaphod](#)).

License

Copyright (c) 2012, Sean Heber. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of Sean Heber nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL SEAN HEBER BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.