

Assignment #4 – Heuristic Search

Overview

The heuristic search project uses the same code base and concepts as the previous assignment. For this reason I will only discuss the changes made and the new concepts that were introduced for this project. The obvious change that I made is using a heuristic search algorithm which uses the A* search algorithm. A* is similar to Dijkstra's Algorithm, which I used in the previous assignment, except it uses a heuristic in the algorithm to help determine which node is most likely on the path towards the goal node.

Implementation

The A* algorithm I implemented was the same algorithm given on wikipedia (http://en.wikipedia.org/wiki/A*_search_algorithm). The heuristic used is the Euclidean Distance between a given node and the goal node (http://en.wikipedia.org/wiki/Euclidean_distance). In order to accomplish this, each node choose a random (x,y) coordinate between 0 (inclusive) and 100 (exclusive). One thing to note is that nodes are not displayed using their (x,y) coordinates. This is because the Jung2 library (<http://jung.sourceforge.net/>) takes care of displaying the nodes for me.

To determine which node to choose next we are given three scores to give us more information about a node: the h-score, g-score, and f-score. The h-score is the heuristic value. The g-score of the node is the weight between a node and it's predecessor. Lastly, the f-score is the h-score plus the g-score. The f-score is the value used to prioritize the nodes. The node with the lowest f-score is the node that gets examined first. The idea is that the node at the top of the queue will more likely be closer to the goal node.

Results

On my homepage you will find four sample results. The first sample shows what happens when the algorithm can find no path; The list that holds the path is null and the length of the path is 0. The next two simply shows that the program does indeed find a path between the start and goal node. The last sample shows that the algorithm does not always find the shortest path. If we examine the results, we'll see that we take the path: 9->10->5 for a path length of 111. If we look a bit closer we'll notice that there is an even shorter path: 9->7->5 for a path length of 93. This is because A* does not always find the shortest path. A* is concerned with finding the first path to the goal node, which may not necessarily be the shortest.

A* performs better than the breadth first method since it picks nodes that are closer to the goal. For this reason it doesn't concern itself with nodes that are away from the goal since it is most likely that those nodes are not on the path. Dijkstra's algorithm on the other hand only looks at the weight of the node and not it's distance to the goal, when determining which node to examine next. It may spend time examining nodes that A* knows not to examine. However, as previously discovered, these nodes may sometimes lead to a shorter path.