Drexel UNIVERSITY

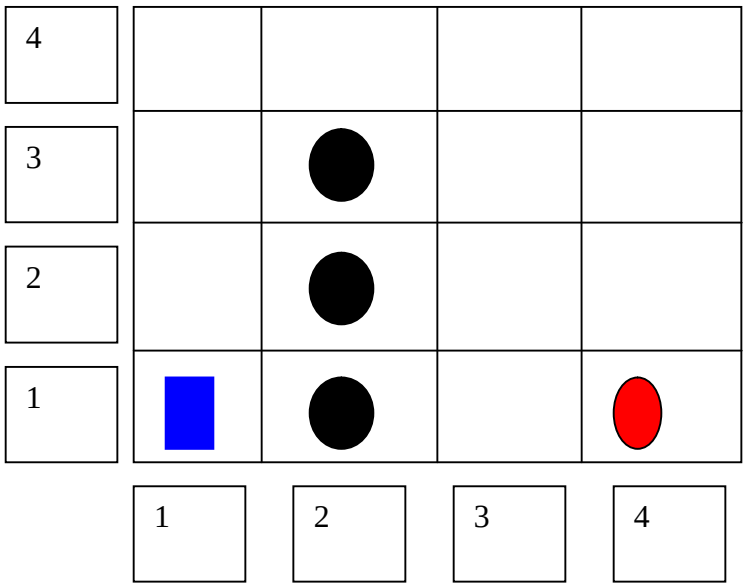
A* Algorithm

Simulation and Source Files

Introduction

Navigating a terrain and finding the shortest path to a Goal location is one of the fundamental problems in path planning. While there are many approaches to this problem, one of the most common and widely known is the A Star search.

We will try to arrive at the A star algorithm intuitively. Consider the problem of traversing the terrain shown below (diagonal movement is allowed).



The robot (Blue Rectangle) has to traverse the terrain and reach the goal (the red Oval). A brute force approach would be to start of with a square and identify all the squares that surround it. Move to one of the successor square and repeat this process until the goal square is found.

However this method is computationally intensive and does not always guarantee the best path to the target.

The key lies in identifying the appropriate successor square. Given some information regarding the location of the target we can try to make an educated guess. For example if you know your target lies to the east, explore squares to the east of your current location. If you know the heading of the target you

could always try to move to the square in that direction.

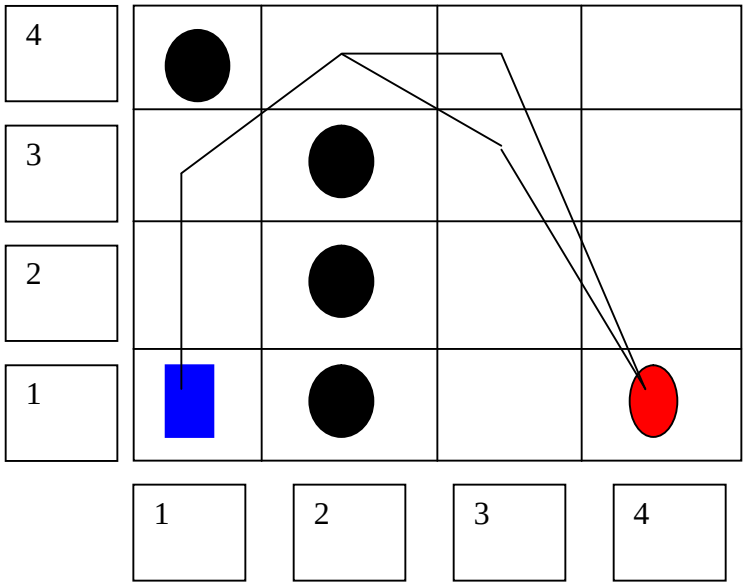
Can you come up with of a better way of determining which square needs to be selected?

A* uses the distance between the current location and the target and moves to the square that has the smallest distance. It evaluates squares (henceforth called a ‼ node¶) by combining h(n), the distance (cost) to that node and g(n), the distance(cost) to get from that node to the goal node. The total cost f (n) = g (n) +h (n) is calculated for each successor node and the node with the smallest cost f (n) is selected as a successor.

Determining the Cost The distance between two nodes is simply determined by calculating the straight distance between the two nodes. Though this might not the true distance (given the fact that there are obstacles that need to be circumnavigated), it never overestimates the actual distance.

It can be shown that as long as cost is never overestimated, the algorithm is admissible i.e. it generates the optimal path.

Let us consider the case for a simple 4X4 Matrix



The start position is (1, 1). The successive node (only one in this case is (1, 2). There is no ambiguity, until the Robot reaches node (2, 4). Here there are two nodes (3, 4) and (3, 3). The successor node can be determined by evaluating the cost to the target from both the nodes.

f(n) for node (3, 3)
h(2,1) = 1 + 1 + sqrt(2) {Distance from N(1, 1) -> N(1,2) -> N(1,3)->N(2,1)}

f(n) = h(2, 1) + 1.414 (assuming each square is 1X1 units)
g(n) = sqrt((4 ┬ 3) ^2 + (1-3)^2) = 2.23
f(n)= 3.44+H(2,1)

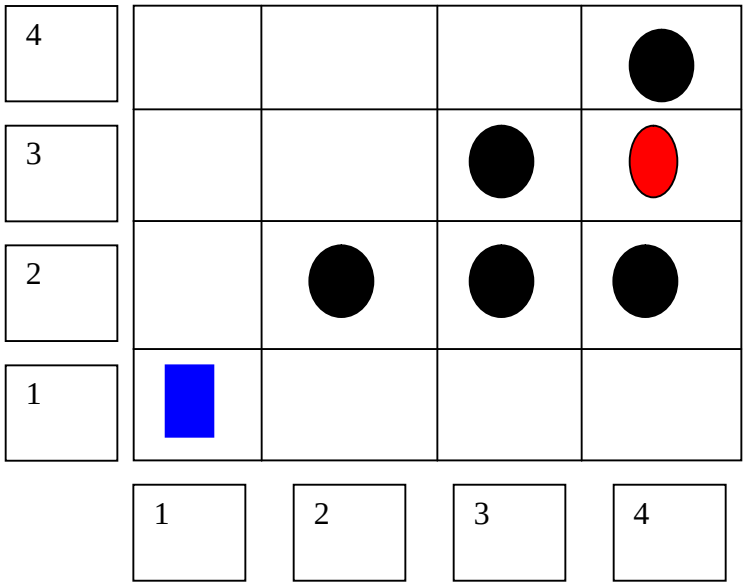
f(n) for node (3,4)
h(n)= h(2,1) + 1
g(n)=sqrt((4-3)^2 + (1-4)^2) = 4.16
f(n) = 4.16 + h(2,1).

Now f(n) for (3,3) has been found to be the smallest of the two, hence the successor node is f(n).

The robot can now move to the node (3, 3) and continue expanding the successor nodes as above, until the goal node is reached.

Dead End

What happens if the robot runs into a dead end? Consider the terrain



From what we have learnt so far Node (2, 1) will be chosen as the successor node instead of Node (1, 2). The robot will continue to traverse the route until it ends up at the block at Node (4, 1).

We need to add a mechanism by which the robot:

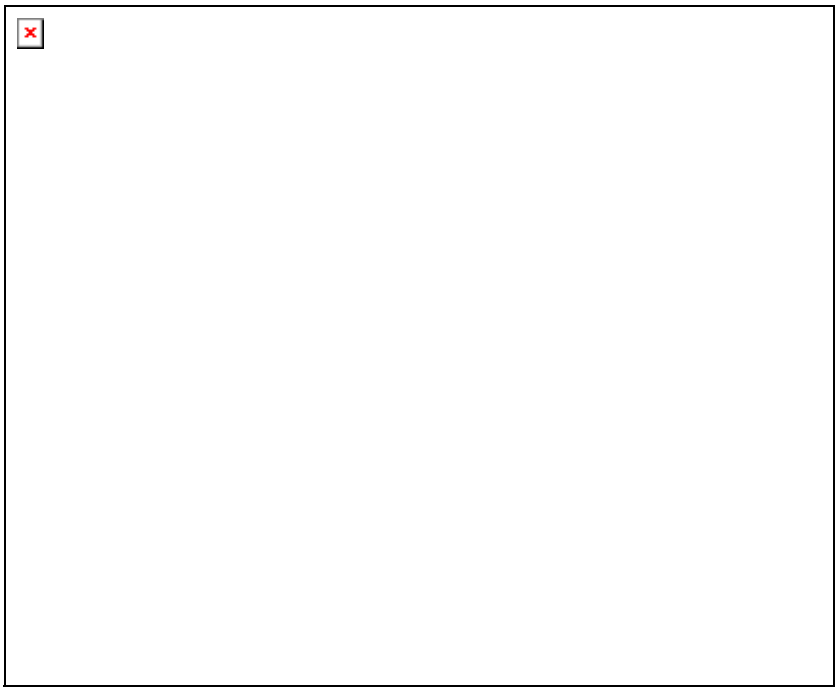
a) Explores alternate routes once it lands up at a dead end.
b) Avoids traversing paths that it knows leads to a dead end.

This is done by maintaining two lists OPEN and CLOSED. The list OPEN stores all successive paths

that are yet to be explored while list CLOSED stores all paths that have been explored.

The list OPEN also stores the parent node of each node. This is used at the end to trace the path from the Goal to the Start position, thus generating the optimal route.

Consider the figure below. The start node has 2 successors (2, 1) and (1, 2). From the initial calculation (2, 1) is chosen and the robot travels along that node, however ones it reaches the dead end, it discards the node (2, 1) and takes the second successor (1, 2) and explores that route.



Once the goal node is reached the parent nodes are found and tracked back to the start node to get the complete path.

In the above example N(4,3) -> N(3,4)->N(2,3)->N(1,2)->N(1,1) gives the optimal path.

From the above conditions the following algorithm can be obtained.

The A* Algorithm

1) Put the start node on the list OPEN and calculate the cost function f (n). {h (n) = 0; g(n) = distance between the goal and the start position, f(n) = g(n).}
2) Remove from the List OPEN the node with the smallest cost function and put it on CLOSED. This is the node n. (Incase two or more nodes have the cost function, arbitrarily resolve ties. If one of the nodes is the goal node, then select the goal node)
3) If n is the goal node then terminate the algorithm and use the pointers to obtain the solution path. Otherwise, continue
4) Determine all the successor nodes of n and compute the cost function for each successor not on list CLOSED.

5) Associate with each successor not on list OPEN or CLOSED the cost calculated and put these on the list OPEN, placing pointers to n (n is the parent node).
6) Associate with any successors already on OPEN the smaller of the cost values just calculated and the previous cost value. (min(new f(n), old f(n)))
7) Goto step 2.

Matlab Program

The Matlab Program consists of the following files

a) A_star1.m: This is the main file that contains the algorithm. This needs to be executed to run the program.
b) distance.m: This is a function that calculates the distance between 2 nodes.
c) Expan_array.m: This function takes a node and returns the expanded list of successors, with the calculated fn values. One of the criteria↕ s being none of the successors are on the CLOSED list.
d) insert_open.m: This function populates the list OPEN with values that have been passed to it. The arguments are xval,yval,parent_xval,parent_yval,hn,gn,fn .
e) min_fn.m: This function takes the list OPEN as one of its arguments and returns the node with the smallest cost function.
f) Node_index.m: This function returns the index of the location of a node in the list OPEN.

Exercises

1) Consider that the robot has certain physical constraints (Ex. Can only take paths that are wide enough for it to travel). If the optimal path leads it through a narrow region, then it is not feasible. Alter the program to then generate an optimal path which is physically feasible with respect to a certain constraint.
2) The above program handles 2D terrains. Modify the program to generate paths in 3 Dimensions.