Name : Neha Mohan Tilak

Project III

Topic : Path-finding using A* Algorithm

CMSC 671

Path-finding using A* Algorithm

Neha M. Tilak, *Master's Student, Computer Science*, UMBC

***Abstract*—Path-finding is a problem of finding the route between two points while avoiding any obstacles and minimizing costs. This problem is of particular interest as it has applications in many areas like exploration, spying, video games, terrain analysis, road planning, robotics etc. The aim of this project is to implement a path-finder for a large grid with a start point, an end point and number of obstacles placed throughout the grid. In addition to this, there will be terrain information in the grid that gives information regarding elevation of the location. An optimum path will minimize the number of hops between start and endpoint while avoiding paths that require climbing up. The informed search technique called A* is used for finding this path. The performance of this algorithm for different heuristics is tested in this project.**

I. INTRODUCTION

THE family of informed search techniques use problem-specific knowledge beyond the definition of the problem itself. The general approach is the best-first search that uses an evaluation function f(n) for selecting the intermediate states in a problem. The most widely known form of best-first search is called A* search. It evaluates nodes by combining g(n), the cost to reach the node, and h(n), the cost to get from the node to the goal.[1]

II. PROBLEM DEFINITION

The aim of the project is to demonstrate the use of A* algorithm for solving the path-finding problem. The input to the path-finder is a two-dimensional array. Each index in the array indicates a location that the path can visit. Some of the positions on the array are marked as obstacles- the path cannot a visit an obstacle location. The terrain is described in terms of absolute elevations. Finally the start location and the goal location is given. The aim is to find the path between start and finish with following constraints- the path should be as short as possible, the path should avoid obstacles, the path should avoid climbing up as much as possible.

A. Cost function

The cost function g(n) is the actual cost of reaching a location n on the grid. As per my implementation, I have allowed eight legal moves from any location – up, down, left, right, right-up, right-down, left-up and left-down. My path-finder assigns a cost of 10 for moving up, down, right or left and cost of 14 for moving right-up, right-down, left-up or left-down. The intuition behind this is that the diagonal distance for two cells will be equal to 14.4 as per Pythagoras theorem.

B. Heuristics function

The heuristic function h(n) estimates the cost of reaching goal state from current state. I have tried using two heuristic functions – Euclidean distance and Manhattan distance and observed that both give satisfactory results when the function approaches actual cost.

The Euclidean distance between two points (x1,y1) and (x2,y2) is given as

Euclidean distance = $\sqrt{(x2 - x1)^2 + (y2 - y1)^2}$

The Manhattan distance between two points (x1,y1) and (x2,y2) is given as

Manhattan distance = |x2-x1| + |y2-y1|

C. Evaluation function

The evaluation function is the sum of the cost function and the heuristic function.

$$f(n) = h(n) + g(n)$$

III. IMPLEMENTATION

I have used C++ for implementing the A* algorithm for path-finder. A two-dimensional matrix is used for storing the location data of start, finish and obstacles. Another two-dimensional integer array maintains the absolute elevations of each location on the grid. The cost of climbing up or down is proportional to the relative elevation of two points.

The algorithm requires us to compute the evaluation function for each child state of the current state. The A* uses the best-first approach and chooses the child with the minimum evaluation function value. For implementing this, I have used a priority queue that keeps the node with the least f(n) value at the top. For every iteration of the loop, the node at the top of the queue is popped. If the node corresponds to the goal location, then the loop exits and the least cost path to the goal node is found. If the popped node is not the goal node, then upto 8 new child nodes are created (corresponding to the legal directions) , their evaluation function is computed and they are pushed back into the queue. If the new node created was already present in the queue and the evaluation function of the new node is lesser than that of old node, then the old node will be replaced with new node. Thus the parent node closest to the child is maintained on the queue.

Finally, a third two-dimensional array is used for displaying the best path from start point to endpoint. This array maintains the direction of the parent node for every node. So tracing back from finish location to the source location gives the path

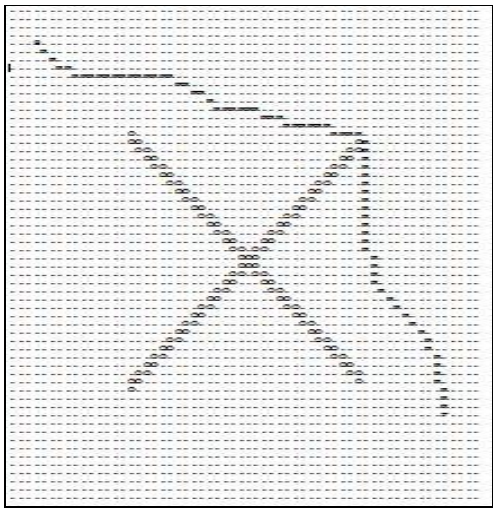
taken. The route taken is displayed with 'R' on the grid, the start location is denoted by a 'S' and the finish location is denoted by a 'F'. The obstacles are denoted by a 'O'.

The algorithm was executed for different heuristics. The following section lists out the result for each of them.

IV. RESULTS

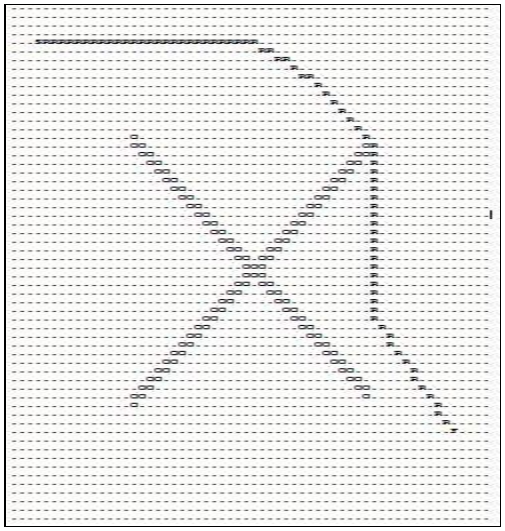
The path-finder was tested for the two heuristics – Euclidean distance and Manhattan Distance. Each of them was weighted tested for different weight factors so as to check the effect of heuristics on the performance of the algorithm. The performance was measured in terms of the intermediate nodes created and the length of the path computed by the algorithm. The results for different cases are given below:

A. For h(n) = $\sqrt{(x2-x1)^2+(y2-y1)^2} * 10$



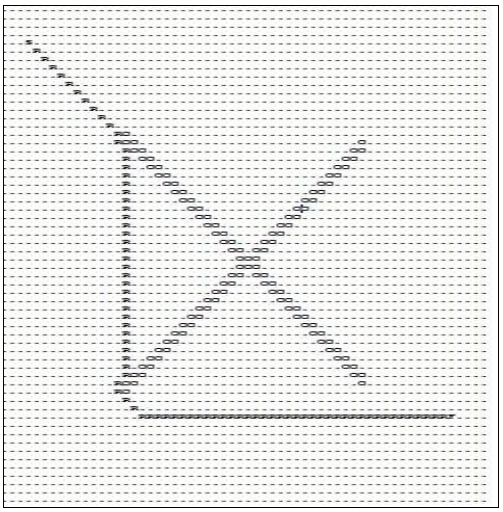
Intermediate nodes generated = 1825
Path length = 75

B. For h(n) = $\sqrt{(x2-x1)^2+(y2-y1)^2}$



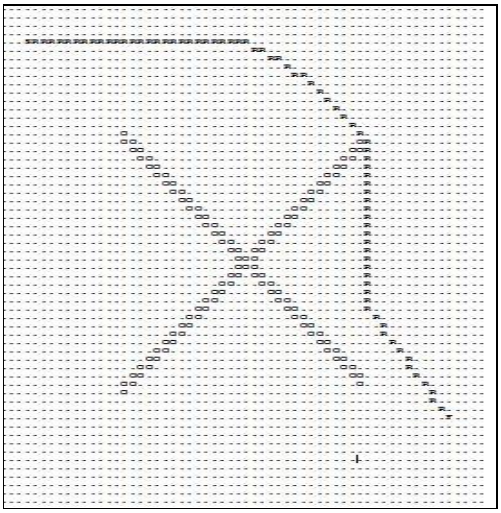
Intermediate nodes generated = 3380
Path length = 75

*C. For h(n) = |x2-x1| + |y2-y1| * 10*



Intermediate nodes generated = 719
Path length = 83

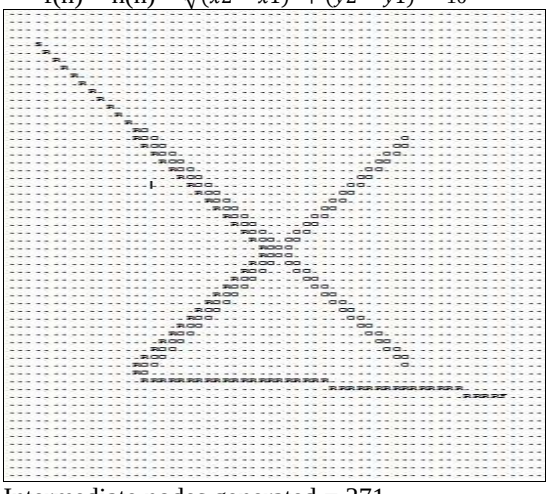
D. For h(n) = |x2-x1| + |y2-y1|



Intermediate nodes generated = 3373
Path length = 75

E. For Greedy choice :

f(n) = h(n) = $\sqrt{(x2-x1)^2+(y2-y1)^2} * 10$



Intermediate nodes generated = 371
Path length = 83

V. ANALYSIS

It can be seen from the results that both the heuristics used give a path from the start to the finish location. So the search is a complete one. However, if the evaluation function maps the actual cost more closely (as in Results section A), then the goal is reached in fewer intermediate steps. This conclusion follows from the fact that part A generates just 1825 intermediate nodes compared to the 3380 nodes created in part B. The same rule holds for the Manhattan distance heuristic. The greedy approach gives a speedy solution at the cost of optimality.

VI. CONCLUSION

The choice of the correct heuristics is important for any informed search algorithm. It affects the optimality and completeness of the solution. The A* algorithm gives a complete search technique for solving the path-finding problem for the two heuristics used.