A * algorithm is a searching algorithm that searches for the shortest path between the initial and the final cell. In this algorithm, we aim to reach the final position, i.e., the endpoint at the shortest possible time. We can imagine a graph with multiple square cells with multiple obstacles scattered randomly. The start and endpoint should have fixed positions.

We have 3 parameters in this algorithm, namely g(n), h(n) and f(n) bound by the relation

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

where h(n) is the heuristic value i.e., estimated cost of moving from the current position to the final position and g(n) being the cost of moving from the source position to the current position.

The actual cost may not be accurately predicted through the value of h(n), but we need to keep in mind that not ever-estimate h(n). At every step, the node with the lowest f(n) node is picked and then process that node.

A* algorithm starts with the initial cell. First, all the adjacent cells are considered, filtering out the inaccessible objects, like walls or any other obstacle. Calculating the f(n) value for all possible paths, finally, the one with lowest cost is picked. This process is recursively repeated till we reach the final cell.

Calculating g(n) is generally considered straightforward as it has an exact value. On the other hand, calculating h(n) is slightly twisted as 'estimation' is involved. Generally, one of the following three methods is used, Manhattan distance, Diagonal distance, or Euclidean distance.

- Manhattan heuristic follows the distance between two points measures along axes at right angle. It is computed by calculating the number of cells moved horizontally and vertically to reach the final cell from the current cell. Diagonal movements are prohibited.

$$h = |xstart - xdestination| + |ystart - ydestination|$$

- In Euclidean heuristic, movement is allowed in all directions. If we try run both simultaneously on the same maze, the Euclidean path finder favours a path along a straight line. This heuristic is more accurate but is slower because it has to explore a larger area.

$$h(n) = \sqrt{(n.x - goal.x)2 + (n.y - goal.y)2}$$

- The Diagonal heuristic allows the movement in 8 directions, allowing the path to pass through the vertices and edges of the cell.

The time complexity of A star algorithm depends on the heuristic. In the worst case, the number of nodes expanded is exponential in the length of the solution (the shortest path), but it is polynomial when the search space is a tree.

Sakshi Gupta

2020A8PS2036G