**A \* Search Algorithm**

The algorithm explaining the functioning of A\* Search is as follows

**function** A\*(start, goal)

*// The set of nodes already evaluated*

closedSet := *{}*

*// The set of currently discovered nodes that are not evaluated yet.*

*// Initially, only the start node is known.*

openSet := *{start}*

*// For each node, which node it can be most efficiently reached from.*

*// If a node can be reached from many nodes, cameFrom will eventually contain the* *most efficient*

*previous step.*

cameFrom := an empty map

*// For each node, the cost of reaching that node form the start node.*

gScore := map **with** default value **of** Infinity

*// The cost of going from start to start is zero.*

gScore[start] := 0

*// For each node, the total cost of getting from the start node to the goal*

*// by passing by that node. That value is partly known and partly heuristic.*

fScore := map **with**  a default value **of** Infinity

*// For the first node, the total cost is completely heuristic.*

fScore[start] := heuristic\_cost\_estimate(start, goal)

**while** openSet **is** **not** empty

current := the node **in** openSet having the lowest fScore[] value

**if** current = goal

return reconstruct\_path(cameFrom, current)

openSet.Remove(current)

closedSet.Add(current)

**for** each neighbor **of** current

**if** neighbor **in** closedSet

**continue** *// Ignore the neighbor which is already evaluated.*

**if** neighbor **not** **in** openSet *// Discover a new node*

openSet.Add(neighbor)

*// The distance from start to a neighbor*

*//the "dist\_between" function may vary as per the solution requirements.*

tentative\_gScore := gScore[current] + dist\_between(current, neighbor)

**if** tentative\_gScore >= gScore[neighbor]

**continue** *// This is not a better path.*

*// This path is the best until now. Record it!*

cameFrom[neighbor] := current

gScore[neighbor] := tentative\_gScore

fScore[neighbor] := gScore[neighbor] + heuristic\_cost\_estimate(neighbor, goal)

return failure

**function** reconstruct\_path(cameFrom, current)

total\_path := [current]

**while** current **in** cameFrom.Keys:

current := cameFrom[current]

total\_path.append(current)

return total\_path