A* Path planning algorithm

```
function A*(start,goal,map_data)
   g_score[start] := 0    // Cost from start along best known path.
   h_score[start] := heuristic_cost_estimate(start, goal)// Estimated total cost from start to goal.
   f_score[start] := g_score[start] + h_score[start]
           closedSet := the empty set    // The set of nodes already explored.
   openSet := {start}    // The set of tentative nodes to be explored, initially containing the start node
   came_from := the empty map    // The map of navigated nodes.
   while openSet is not empty
      current := the node in openset having the lowest f_score[] value
      if current = goal
         return reconstruct_path(came_from, goal)
      openSet.remove(current)
      closedSet.add(current)
                     neighbor_nodes :=find_neighbors(current,map_data)
      for each neighbor in neighbor_nodes()
         if neighbor in closedSet
            continue
         tentative_g_score := g_score[current] + movementCost_between(current,neighbor)
         if neighbor not in openSet or tentative_g_score < g_score[neighbor]
            came_from[neighbor] := current
            g_score[neighbor] := tentative_g_score
            f_score[neighbor] := g_score[neighbor] + heuristic_cost_estimate(neighbor, goal)
            if neighbor not in openSet
               openSet.add(neighbor)
   return failure
function find_neighbors(current,map_data)
       x :=current node's x coordinate
```

```
                    y :=current node's y coordinate
        neighbor_nodes[]:=map_data(x-1,y)
                    neighbor_nodes[]:=map_data(x+1,y)
                    neighbor_nodes[]:=map_data(x,y-1)
                    neighbor_nodes[]:=map_data(x,y+1)
                    neighbor_nodes[]:=map_data(x-1,y-1)
                    neighbor_nodes[]:=map_data(x-1,y+1)
                    neighbor_nodes[]:=map_data(x+1,y-1)
                    neighbor_nodes[]:=map_data(x+1,y+1)
        return neighbor_nodes
function reconstruct_path(came_from,current)
    final_path := [current]
    while current in came_from:
        current := came_from[current]
        total_path.append(current)
    return final_path
```