1 Question 2

1.1 Part a

The A* algorithm is a pathfinding algorithm that combines the benefits of Dijkstra's algorithm and a heuristic approach to efficiently find the shortest path between two nodes in a graph. As explained on Lecture 4, it uses a priority queue to explore nodes based on the predicted total path length. This is computed as $f(x) = g(x) + h(x)$, where $g(x)$ represents the actual cost from the start node to the current node x, and $h(x)$ represents the heuristic estimate of the cost from x to the goal node.

To adapt Lavelle's ForwardSearch, the A* algorithm is used to extract the state with the lowest predicted path cost $f(x)$. Each cell in the grid has an associated label (marking states as Unvisited, Alive or Dead), a `parent` backpointer, a cost-to-come `path_cost` and a predicted cost `predicted_path_cost`. The algorithm takes into account the robot's current position, the goal position, and any obstacles in the environment to resolve duplicate paths and ensure that the optimal path to the goal is found.

Algorithm 1: A* Algorithm (adapted from Lavelle's ForwardSearch)

```
1:  function FORWARDSEARCH(x_I, X_G)
2:     Q.Insert(x_I) with Priority (0 + h(x_I))
3:     Mark x_I as Alive
4:     g(x_I) <- 0
5:
6:     while Q not empty do
7:        x <- Q.GetFirst()            // Extracts state with lowest f(x) = g(x) + h(x)
8:        if x in X_G then
9:           return SUCCESS
10:
11:       for all u in U(x) do         // Iterate through all available actions
12:          x' ← f(x, u)             // Compute next state (transition function)
13:          cost_to_come ← g(x) + c(x, u)
14:
15:          if x' is Unvisited then
16:             Mark x' as Alive
17:             Set Parent(x') ← x
18:             g(x') ← cost_to_come
19:             Q.Insert(x') with Priority (g(x') + h(x'))
20:
21:          else                     // "Resolve duplicate x'"
22:             if cost_to_come < g(x') then
23:                g(x') ← cost_to_come
24:                Set Parent(x') ← x
25:                Q.Update(x') with Priority (g(x') + h(x'))
26:                if x' is Dead then Mark x' as Alive
27:
28:       Mark x as Dead
29:
30:    return FAILURE
```

1.2 Part b