[HW2-1] Introduction to AI [CS 510]

Name: Shrey Soni

Student Id: ss5767

Date: 1/18/25

Finding shortest path

1. Suppose the state space consists of all positions (x, y) in the plane. How many states are there? How many paths are there to the goal?
    - If the state space consists of all positions (x, y) in the plane, the state space is continuous, which means all possible positions can become state. So, the number of states is **infinite**.
    - Even if there are only valid paths to avoid obstacles, the states are still infinite; So, there are **infinitely many possible paths** from the start(S) to the goal(G). it could be lines, curves, etc. It is due to an infinite number of states.

2. Explain briefly why the shortest path from one polygon vertex to any other in the scene must consist of straight-line segments joining some of the vertices of the polygons. Define a good state space now. How large is this state space?
    - **The shortest path from one polygon vertex to any other in the scene consists of a straight line** because, the shortest distance between two lines is a straight line, so any path other than a straight line would increase the total path length.
    - Also, while restricting paths to straight lines between vertices, we reduce the complexity of the problem while ensuring that the shortest route is possible.
    - Since all the obstacles are convex polygons, the path would be a straight line and would never go inside them, so it would overcome the negative path around them without requiring curving.

- **A good State space is defined as** A set of vertices of all polygons, with start (S) and goal (G) points. These vertices represent the possible points where the shortest path may turn while avoiding obstacles. The state space is good if it avoids all the other possible points of (x, y) as the shortest path will never require visiting them.
- **The total size of state space,** if v is the total number of vertices across all the polygons, and adding Start (S) and Goal (G) in that, the **total size of state space = v + 2**

3. **Define the necessary functions to implement the search problem, including an ACTIONS function that takes a vertex as input and returns a set of vectors, each of which maps the current vertex to one of the vertices that can be reached in a straight line. (Do not forget the neighbors on the same polygon.) Use the straight-line distance for the heuristic function.**
    - The goal is to find the shortest path between Start S and Goal G while navigating the convex polygonal obstacles. Below are the functions that would be useful while finding the shortest path from Start to Goal.
        i. **Actions function:** The actions function finds the vertices that can be reached from the current vertex. For example, if the current point is at vertex A, and there is a straight line to vertex B and no obstacles in between them, then B is reachable from A.
        We can use the helper function which checks whether there is a straight line between two vertices intersected by any obstacles.
        - Actions Functions (vertex, vertices, obstacles):
            - Input:
                - current vertex (x, y)
                - Vertices: list of all vertices including Start and Goal
                - Obstacles: list of polygons
            - Output:
                - List / Set of reachable vertices
            - We can loop through the vertices and find whether the vertex is reachable from the current vertex or not using the helper function.
        - Helper function (vertex1, vertex2, obstacles):
            - Input:
                - vertex1, vertex2: two vertices
                - obstacles: list of polygons
            - Output:
                - It will return a Boolean, True if the path is clear, False if there is an obstacle detected in-between the path.

ii. **Transition function:** This function calculates the cost of moving between two vertices. We can calculate it using the Euclidian distance formula. This helps us to find the total cost of the path.

- Function Transition (v1, v2):
    - Input:
        - v1, v2: vertices (x,y)
    - Output:
        - Distance between v1 and v2

iii. **Heuristic function:** This function estimates the cost of reaching the goal from a given vertex. this function guides the algorithm to explore the paths that are closer to the goal, which improves efficiency.

- Function Heuristic (vertex, goal):
    - Input:
        - Vertex: current vertex
        - Goal: destination vertex
    - Output:
        - Distance between the vertex and Goal.