This file provides pseudo code for all the functions you need to implement.

- You need to implement the drive(goalstates,inputs) function.
- In drive(goalstates,inputs) function you can free to implement any search algorithm covered in class (explain your algorithm in solution pdf), Your algorithm should perform well on different scenarios.
- To provide you assistant in implementation this pdf provides pseudo code for Drive function, A* search and its helper functions [heuristic cost and path reconstruction],
- Pseudo codes for all 4 functions [Drive, A*, heuristic cost and path reconstruction] are provided below, you can just convert them into python code to complete the assignment.
- Submit the function code in pdf file. The code has to compile and run as part of the simulator without issues.

Algorithm 1 drive(goalstates, inputs)

- 1: extract start and goal states from environment
- 2: for for all goal states do
- 3: goalReached,path= $A_Star(start, goal)$
- 4: Find best path from all paths received [1 path received for 1 goal]
- 5: [Best path, would the shortest path in case of goal is reachable otherwise it would be the longest path, how far traveled before being blocked]
- 6: Compute action sequence for best path
- 7: retrun action sequence

For more details (and pseudo code) of "A star search" go through the class slides and wiki page https://en.wikipedia.org/wiki/A*_search_algorithm

closedSet - The set of nodes already evaluated

openSet - The set of currently discovered nodes that are not evaluated yet.

cameFrom - which node it can most efficiently be reached from.

gScore - the cost of getting from the start node to that node.

fScore - cost of getting from the start node to the goal. by passing that node.

```
Algorithm 2 A-Star(start, goal)
```

```
1: closedSet = [start]
 2: openSet = []
 3: cameFrom = []
 4: gScore =
 5: gScore[start] = 0
 6: fScore = []
 7: fScore[start] = heuristic\_cost\_estimate(start, goal)
 8: current = []
 9: while openSet is not empty do
     current = the node in openSet having the lowest fScore value
10:
     if current = goal then
11:
        return Goal Reached, reconstruct_path(cameFrom, current)
12:
     remove current from openSet and fScore
13:
14:
     add current to closedSet
     for all actions do
15:
        neighbor = applyAction(current, action)
16:
        if neighbor = current then
17:
          path blocked, stop, evaluate next action
18:
        tentative\_gScore = gScore[current] + Distance from start to neighbor
19:
        if neighbor not in openSet then
20:
          Add neighbor in openSet
21:
        else if tentative\_gScore \ge gScore[neighbor] then
22:
          stop, evaluate next action
23:
        cameFrom[neighbor] = current
24:
        gScore[neighbor] = tentative\_gScore
25:
        fScore[neighbor]
                                                 gScore[neighbor]
26:
        heuristic\_cost\_estimate(neighbor, goal)
27: if current = goal then
28:
      return Goal Reached, reconstruct_path(cameFrom, current)
29: else
     return Goal Not Reachable, reconstruct_path(cameFrom, current)
30:
```

Algorithm 3 $heuristic_cost_estimate(start, goal)$

1: return [how far ahead is goal state]/2

Algorithm 4 $reconstruct_path(cameFrom, current)$

```
1: total_path = []
2: total_path.append(current)
3: while current in cameFrom.keys do
4: total_path.append(current)
5: current = cameFrom[current]
6: total_path.append(current)
7: return total_path
```