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 $drive(goalstates, inputs)$

extract start and goal states from environment
for for all goal states **do**
  goalReached,path= $A\_Star(start, goal)$
Find best path from all paths received [1 path received for 1 goal]
[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]
Compute action sequence for best path
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 $A\_Star(start, goal)$

```
closedSet = [start]
openSet = []
cameFrom = []
gScore =
gScore[start] = 0
fScore = []
fScore[start] = heuristic_cost_estimate(start, goal)
current = []
while openSet is not empty do
    current = the node in openSet having the lowest fScore value
    if current = goal then
        return Goal Reached, reconstruct_path(cameFrom, current)
    remove current from openSet and fScore
    add current to closedSet
    for all actions do
        neighbor = applyAction(current, action)
        if neighbor = current then
            path blocked, stop, evaluate next action
        tentative_gScore = gScore[current] + Distance from current [redacted] to neighbor
        if neighbor not in openSet then
            Add neighbor in openSet
        else if tentative_gScore ≥ gScore[neighbor] then
            stop, evaluate next action
        cameFrom[neighbor] = current
        gScore[neighbor] = tentative_gScore
        fScore[neighbor] = gScore[neighbor] + heuristic_cost_estimate(neighbor, goal)
if current = goal then
    return Goal Reached, reconstruct_path(cameFrom, current)
else
    return Goal Not Reachable, reconstruct_path(cameFrom, current)
```

Algorithm 3 $heuristic\_cost\_estimate(start, goal)$

```
return [how far ahead is goal state]/2
```

Algorithm 4 $reconstruct\_path(cameFrom, current)$

```
total_path = []
total_path.append(current)
while current in cameFrom.keys do
    total_path.append(current)
    current = cameFrom[current]
total_path.append(current)
return total_path
```