AIFA Assignment-1 Report
Problem-1 (Multi-agent path-finding)

1. Saransh Gupta | 17QM30005

2. Mridul Agarwal | 17QE30008

3. Aniruddha Chattopadhyay | 17QE30007

4. Divyanshu Sheth | 18QE30008

The code is kept in the Jupyter-Notebook file (AIFA-ASSIGNMENT-final.ipynb). In order to run the code, all the cells need to be run sequentially.
The inputs are taken from the text files:

1. dimensions.txt: It specifies the length and breadth of the rectangular grid considered as the floor map of the warehouse for this problem. Length (x dimension) is separated by breadth (y dimension) here.
2. obstacles.txt: It contains the locations of obstacles on the grid (y dimension (row) followed by x (column))
3. bot_positions.txt: It contains the initial and the final positions of the robots. Each line contains first the initial position (y dimension (row) followed by x (column)), and then the final position of the robot.
4. locations.txt: It contains the initial and final locations of the objects. Each line contains first the initial location (y dimension (row) followed by x (column)), and then the final location of an object.

Inputs to the problem: The positions of obstacles, initial/final location of robots and items to be delivered in the grid.

Results: For each robot, its path is returned as the final output. Robots first move from their initial position to the position of one of the objects, and then towards

the final location for the object and eventually get back to the final position specified for the robot.

Requirements: Python 3, heapq library, jupyter-notebook

Algorithm:
There are three main steps followed in the algorithm used in the solution. The first one is the assignment of tasks to the agents involved in a particular case. The second step involves finding the shortest route between two given points in the map in question. The third step involves resolution of conflicts in the trajectories of two or more agents in the problem.

A list of lists, where each inner list represents a list of work allotted to a robot, is used in allocation of tasks. We calculate the heuristic used as the distance of a robot from an item pick-up location + distance to the delivery station, and based on this, work is allocated to the agents in the representation specified above. A search algorithm is used to make sure all robots are given some task at the final state.

The A* algorithm is used for calculating the minimum distance between the item's source and destination, and here the manhattan distance is the heuristic implemented. In Python, heapq is used for implementing the priority queue used in A*. If the paths of two or more robots collide, one or more of the robots are reassigned to different paths by checking the individual times that they spend idle at the end, after the completion of all the tasks assigned to them. The robots that have the most idle time are given higher precedence in rerouting, as that will lead to the least amount of extra time spent due to the rerouting. After all rerouting, paths of the bots no longer collide and the solution is obtained. A list is outputted for each robot with the locations of the cells it must travel to in its journey, in the final result.

Shortcomings (Possible Improvements):

The avoidance of collisions and the assignment of tasks can be further optimized. With respect to collisions, certain scenarios can result in there being no path for the robot to change tracks, for eg, when two bots meet on a path which is surrounded by a chain of obstacles on both sides, wherein the bots will be required to retreat and retrace their paths. Also, the optimal usage of temporary storage locations on the grid can lead to further optimization of the solutions.