Report on Assignment 1 AI60005

The algorithm used to solve the assignment is Conflict Based Search (CBS). In the *multi-agent pathfinding* problem (MAPF) we are given a set of agents each with respective start and goal positions. The task is to find paths for all agents while avoiding collisions.

CBS is a two-level algorithm. Any particular robot is assigned the closest task to it and the solution is then found out using low level CBS (based on A* algorithm). At the high level, a search is performed on a Conflict Tree (CT) which is a tree based on conflicts between individual agents. Each node in the CT represents a set of constraints on the motion of the agents. At the low level, fast single-agent searches are performed to satisfy the constraints imposed by the high-level CT node. In many cases this two-level formulation enables CBS to examine fewer states than A* while still maintaining optimality. The *multi-agent pathfinding* (MAPF) problem is a generalization of the single-agent pathfinding problem for k>1 agents . It consists of a graph and a number of agents. For each agent, a unique start state and a unique goal state are given, and the task is to find paths for all agents from their start states to their goal states, under the constraint that agents cannot collide during their movements. In many cases there is an additional goal of minimizing a cumulative cost function such as the sum of the time steps required for every agent to reach its goal.

Psuedo code for CBS

High Level search

```
Input: grid instance
R.constraints = {}
R.solution = find individual paths using the low level()
R.cost = cost_function(R.solution)
insert R to frontier
while frontier not empty do
    node <- minimun cost node from the frontier
    Validate the paths in node until a conflict occurs.
    if node has no conflict then
        return node.solution//node id the goal

    C <- first conflict(robot_1,robot_2, time) in node
    for each robot in C do
        A <- new node
        A.constraints <- node.constraints
        A.sloution <- node.solution
        Update A.solution by invoking low-level(robot)
        A.cost = cost_function(A.solution)
        Insert A to frontier
```

```

Low Level search is based on A* finds the simplest solution for all robots
```
make an frontier containing only the starting node
make an empty explored\\(to keep track of all explored nodes)
while (the destination node has not been reached):
  consider the node with the lowest f score in the frontier
  if (this node is our destination node) :
           we are finished
  if not:
    put the current node in the explored and look at all of its neighbors
    for (each neighbor of the current node):
      if (neighbor has lower g value than current and is in the explored) :
                  replace the neighbor with the new, lower, g value
                  current node is now the neighbor's parent
      else if (current g value is lower and this neighbor is in the frontier):
                  replace the neighbor with the new, lower, g value
                  change the neighbor's parent to our current node

               else if this neighbor is not in both lists:
                   add it to the frontier and set its g
```

### Heuristics Used
Heuristic used is based on the manhattan distance from a particuler node
```
heuristic_value = |current_x_coordinate - goal_x_coordinate| + |current_y_coor
dinate - goal_y_coordinate|
```
Tasks are also assigned using the same logic

| | -> mod function

### Task Assignment
Task is assigned using a heuristic which takes into account the manhattan dist
ance between the task and robot and assigns the shortest distance task to the
robot
```
task_assignment_value = |robot_start_x_coordinate - task_pick_up_x_coordinate|
 + |robot_start_y_coordinate - task_pick_up_y_coordinate|
```
### Searched in segments
A* algorithm(low level search) is runned in segments like from robot start to
task pickup then from task pickup to task drop off and then from task drop off
 to robot destination

## Results
```

For a 1 Robot system -
input and output is given in the following respective files above - input_data .yaml and schedule.yaml

Input_data.yaml

```yaml
robots:
-   start: [0,3]
    goal: [3,1]
    name: R1
task:
-   start: [3,3]
    goal: [0,1]
    name: T2
map:
    dimensions: [4, 4]
    obstacles:
    - !!python/tuple [1, 1]
    - !!python/tuple [2, 2]
    temporary:
    - !!python/tuple [0,0]
```

Schedule.yaml

```yaml
cost: 14
schedule:
  R1:
  - t: 0
    x: 0
    y: 3
  - t: 1
    x: 1
    y: 3
  - t: 2
    x: 2
    y: 3
  - t: 3
    x: 3
    y: 3
  - t: 4
    x: 2
    y: 3
  - t: 5
    x: 1
    y: 3
  - t: 6
    x: 1
    y: 2
  - t: 7
    x: 0
    y: 2
  - t: 8
    x: 0
    y: 1
  - t: 9
    x: 0
    y: 0
  - t: 10
    x: 1
    y: 0
  - t: 11
    x: 2
    y: 0
  - t: 12
    x: 3
    y: 0
  - t: 13
    x: 3
```

	y: 1