ASSIGNMENT-3

-DHRUMIL LOTIYA, 21D070026

I have implemented a planner using the A* algorithm to navigate a robot from its current position to a target position while avoiding obstacles. The summary of the approach is as follows:

- **Node Representation:**
  Each position in the grid is represented as a NodeInfo object, which stores information about the position, the cost from the start, the cost value, the parent node, the time step, and the heuristic cost.
- **Priority Queue:**
  A priority queue (openList) is used to store nodes based on their total cost (sum of cost from start and heuristic cost). Nodes with lower total cost have higher priority.
- **Heuristic:**
  The Manhattan distance heuristic is used to estimate the cost from each node to the target position. This heuristic is admissible and efficient for grid-based pathfinding.
  Mathematically, the Manhattan distance d between two points (x1,y1)(x1,y1) and (x2,y2)(x2,y2) can be expressed as:

  $$d = |x2 - x1| + |y2 - y1|$$

  In my planner, it provides a lower bound on the actual cost required to reach the goal, making it admissible (i.e., it never overestimates the true cost). The heuristic guides the search algorithm by prioritizing nodes that are closer to the goal, leading to more efficient exploration of the search space.
- **A Search***:

The A* search algorithm iteratively expands nodes from the priority queue until either the goal is reached or there are no more nodes to explore.

At each iteration, the algorithm selects the node with the lowest total cost from the priority queue and explores its neighbors.

For each neighbor, the algorithm computes the cost from the start through the current node and estimates the total cost using the heuristic.

If a neighbor has not been visited before or has a lower cost from the start through the current node, it is added to the priority queue.

The search continues until the goal is found or the priority queue is empty.

- **Path Reconstruction:**

  Once the goal is found, the algorithm reconstructs the path by backtracking from the goal node to the start node using parent pointers.

  The optimal actions (movement directions) for each time step are stored in actionMapX and actionMapY.

- **Action Selection:**

  Finally, the code selects the optimal action for the current time step based on the robot's current position and updates action_ptr accordingly.

RESULTS FOR ALL MAPS

Map1:
target caught = 1
time taken (s) = 2640
moves made = 2639
path cost = 5344

Map2:
target caught = 1
time taken (s) = 5012
moves made = 1528
path cost = 2010743

Map3:
target caught = 1
time taken (s) = 243
moves made = 242
path cost = 243

Map4:
target caught = 1
time taken (s) = 379
moves made = 266
path cost = 379

Map5:
Target caught=0

Map6:
target caught = 1
time taken (s) = 140
moves made = 0
path cost = 2800

Map7:
Target caught = 0

Map8:
target caught = 1
time taken (s) = 431
moves made = 430
path cost = 431

Map9:
target caught = 1
time taken (s) = 368
moves made = 367
path cost = 368

For Map5 and Map7, it is not possible for my planner to catch the target because of the time limit constraint, that is the target vanishes after the time limit is exceeded and hence the robot is not able to catch the target and also since the target vanishes, memory gets null value. Due to this, code gives rise to segmentation faults.

Instructions to compile code

Follow the same instructions as that given in the question to compile code, no extra files are needed.