A* algorithm implementation to enemy in Complex World

Problem Statement: Considering the complex 3D world problem, which remind most of us to the Pac-Man game introduced in early 80s. This world is having a problem statement to trap the agent as fast as possible. For this, we need to implement the A* algorithm on enemy to quickly trap the agent and catch to solve the problem.

Introduction: The A* algorithm is a path-finding or graph traversal algorithm. It can be used to reach the target from the source with having the least cost associated on the path traversed. It is also referred as informed search algorithm as involve movement based on the cost associated to reach target, commonly refer to use heuristic search.

Implementation & Statistical Explanation:

- Algorithm is completely focused to trap the agent. This can be explained by the path of enemy moving towards the agent. Once enemy move in neighbour position, following the agent till got trapped.
- Algorithm is completely focus on least score possible to agent. This can be explained by the very high number of bad steps when enemy moves in close to the agent. Provided reference to one case in the corner cases below.

There is a 50 X 50 grid representation stored using a 2-D array and we have 4 different objects (maze, wall, agent, enemy) used to show in this grid view. The perimeter of the grid are the spot positions having the wall and setting the scope of movement. The maze boxes are randomly distributed in one third (1/3) positions of the spots possible. These cosmetic changes are introduced updating the existing code template.

The algorithm is implemented to have each grid values as available Spots. Each Spot use to hold certain set of properties i.e., f, g, h, i, j, maze, wall, neighbours set, previous. Please refer below explanation for the properties:

i – It is the column index value in 2D array

j – It is the row index value in 2D array

f – It is the overall cost to reach the target

g – It is the cost to reach the neighbour values

h – It is the heuristic i.e., cost to reach target from the possible neighbours

maze – It is the Boolean value to represent maze in the grid

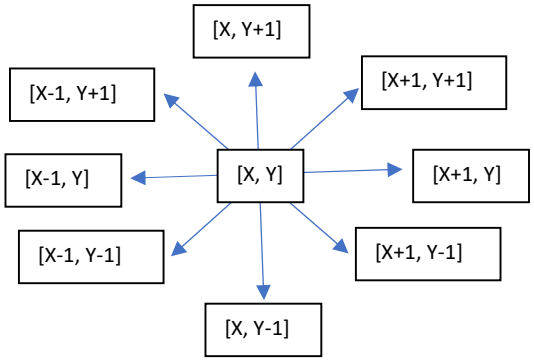
wall – It is the Boolean value to represent wall in the grid

neighbours set – It is the list of all neighbours for a Spot in the grid

previous – It is a pointer to the predecessor Spot in the grid

The implemented solution focus entirely on having the agent trapped and catch by the enemy in least possible steps i.e., agent made minimum number of best possible moves. This can be justified by expanding the neighbour set of the enemy to move if possible, in all 8 directions considering the adjacent neighbours are not occupied by a maze or a wall.

The heuristic plays an important role in good A* algorithm implementation. The algorithm is implemented to have a heuristic less than the cost to reach the agent. If we select a heuristic greater than the cost to reach the agent, the performance will be less accurate. Here, the heuristic or *h(n)* is the exact cost of reaching the destination and it is wisely chosen based base on the type of movement i.e., straight-line or diagonally.



The ***Manhattan Distance Heuristic*** is applied when having a straight-line path to move towards the agent. It is called Manhattan method because it is calculated as total number of spots moved in straight-line ignoring the diagonal movement and mazes.

$$h = |x_{start} - x_{destination}| + |y_{start} - y_{destination}|$$

The ***Euclidean Distance Heuristic*** is applied when having a diagonal movement towards the agent. This is slightly more accurate that the Manhattan counterpart. It is the straight-line distance between the neighbour set spots available on diagonal movement and the agent. It may be more accurate but turns slower because to explore the large grid area.

$$h = \sqrt{(x_{start} - x_{destination})^2 + (y_{start} - y_{destination})^2}$$

The algorithm is made to trap the agent and considered trapped when enemy reach the agent and the agent is blocked not possible to make any further move. However, even considering to have minimum steps possible for the same.

Corner Cases & Future Work:

In some instances, the agent's random move leads to scenario of not falling in trap places which in turn result in increasing the number of steps or run over in very few near to no cases. However, the trap is delayed but the score is minimal for agent it means counted in bad steps are high (enemy is trailing the agent to fall in a trap). *So, even considering these corner case number of bad steps are more than half of number of steps.*

In future work, try to implement counter to A* algorithm on the movement of agent.