COMP30024 AI: PROJECT PART A

Sean Maher 1079800 & Elizabeth Wong 1082634

1. ***Implementation and Relevant Data Structures:***

The pseudocode for the A* search is outlined below:

```
Set goal_state = False
While start node:
        ❖ calculate the neighbours of the start node
        ❖ find the f cost using the Manhattan distance heuristic
        ❖ add the lowest current f cost to processed list
        ❖ remove lowest current from to_search list
        ❖ remove neighbour duplicates from to_search

    If 2 nodes have the same f cost:

        ❖ take node with the lowest h cost

When goal_state = True:

        return processed nodes:

        for each node.connected in explored nodes:

                if node.location ==  current.connected:
                        ❖ add node.location to path
                        ❖ current = node

                        break

Return path
```

The data structures used were mainly lists and in order to easily access node attributes such as location, g, h, f and connected a data class Node() was initialized. Instead of using the python PriorityQueue library we implemented 2 main lists (to_start and processed) to mimic the actions of a priority queue where each node is sorted based on order and popped accordingly due to their respective lowest f costs.

1. ***Time and Space Complexity***

The average space complexity is $O(b^d)$ as the a_star_search() stores all generated nodes to be searched within its memory – to_search, processed and path.

The average time complexity for the a* function would be $O(b^d)$. The best case occurs when the start node is directly adjacent to the goal node – thus a time complexity of $O(d)$. The worst case $O(b^d)$ occurs when all possible nodes in the board are explored before the goal node is found and a path can be generated.

All these cases concerning time complexity are heavily dependent on the chosen heuristic, as a good heuristic allows a* to prune away many of the b^d nodes that a uniformed search would expand.

2. ***Heuristic Function and Admissibility***

What heuristic did you use and why? Show that it is admissible, and discuss the cost of computing the heuristic function, particularly in relation to the overall cost of the search.

The Manhattan distance was used as a heuristic where:

$$Mdist = |x^2 - x^1| + |y^2 - y^1|$$

Admissibility occurs when the estimated cost to read the goal state is not higher than the lowest possible cost from its current point in the path. Assuming that the cost of moving to each tile = 1, and the Manhattan distance gives us the shortest route disregarding blocks; the estimated cost will always be lower or equal to the lowest possible cost from its current point in path.

In the beginning Euclidian distance was considered as a possible heuristic, however there will be more computational overhead. Additionally, as Euclidian distance is shorter compared to the Manhattan distance, despite getting the same shortest path – a* will take longer to run.

3. ***<u>Challenge</u>***

Suppose the search problem is extended such that you are allowed to use existing board pieces of a specific colour as part of your solution path at no cost. An optimal solution is now defined as a minimal subset of unoccupied cells that need to be 'captured' by this colour in order to form a continuous path from the start coordinate to the goal coordinate. How would you extend your current solution to handle this? Discuss whether the heuristic you used originally would still be admissible.

The heuristic originally used would remain admissible as currently we assume the cost of moving to each tile = 1, but in the challenge scenario there are parts where the solution path will have no cost.

Methods that could be used to extend our current solution to handle this problem would be choosing a different heuristic function that finds the shortest path prioritising cells that are coloured. This would mean that we would have to be able to read into a data structure cells of specific colour to define a path from.