COMP30024 Artificial Intelligence
Project Part A

Michael Xie 1181068
Harold Liu 1166132

Object-oriented features of python 3 were used in implementing the a star algorithm by declaring class **hex** as its own class representing each hex cell on the board with functions calculating hexagonal Manhattan distance, as well as neighbouring hex cells via two class methods. Neighbouring hex cells are declared the child cell of the centre node, so that we are able to backtrack along the path after the goal is reached, thereby generating a tree structure.
After scanning in data from input, we then initialise two arrays. One which contains every neighbouring cell from the current cell to be considered by the evaluation function, and one that contains every cell that has already been considered by the function.
We then keep looping through unexplored cells until the goal is reached or the array of unexplored cells is exhausted. At each state, the algorithm compares possible cells to explore next using the evaluation function of the predefined heuristics function - that is - hexagonal Manhattan distance. Once the target cell is found, we terminate the search and initialise another array which records the path taken by backtracking through each hex cell's parent cell.

Derived from the general formulae for A-star search algorithm with an expected solution depth is d=n (as 2n is is maximum distance on a given board, and n is the expected distance assuming start point and end point is random) for a given n*n sized board and a branching factor of b=6 for each hexagonal shape (which is self-evident given the shape).
The worst-case space complexity is therefore O(b^d), as this is the maximum number of shapes on the grid.
The worst-case time complexity is also O(b^d), as the algorithm will need to check every hexagonal shape once.

We decided to use hexagonal Manhattan distance or the equivalent of Manhattan distance with a hexagonal coordinate system. We chose to use this heuristic because it is simple to implement as each cell on the board is given as a 2-dimensional vector. We also don't have to worry about decimals as cells are always incremented by integers. In addition, the graph can only be traversed cell by cell, hence reducing the benefits of euclidean distance. We calculate the hexagonal Manhattan distance by first computing the r (row) and q (column) displacement between two cells as rvec and qvec.
Then applying the formula: $(abs(rvec) + abs(qvec) + abs(rvec + qvec))/2$
We take absolute values to ensure that negative displacements do not affect the final distance.
The heuristic is admissible as it calculates the absolute minimum number of cells in order to traverse from one cell to another, hence, the solution path can only be equal to or more than the Manhattan distance (depending on the positions of preoccupied cells).

The heuristic function requires 3 calculations and hence 3 operations to compute. Therefore, in relation to the overall cost of the search, it would take (3n) time where n is the number of cells, hence O(n).

The proposed search problem can be solved by variations in heuristics function in the current version of the search algorithm. The current algorithm being used will no longer be admissible, as it is possible in the given case that a route with longer Manhattan distance will be considered more desirable. The current heuristic function will therefore no longer satisfy h(n) <= h*(n), as some cell n may have h(n) > h*(n) (if the route is not direct and contains turns or u-turns). A simple heuristic function can be implemented by h'(n) = h(n) - (number of cells of usable colour). This will satisfy the constraint of admissibility. Alternatively, it is possible to produce h''(n) <=h*(n) that reduces branching by greater degree (that is, produces result more similar to that of h*(n)), by calculating the number of cells that need to be captured via uninformed search. This may increase the search efficiency theoretically but may become empirically slower due to overhead of uninformed search.