**COMP30024 Artificial Intelligence**

**Project Part A: Searching**

Authors: Ying Shan Saw (1118861) & Joeann Chong (1152260)

**Implementation of A\* search**

We decided on using a **Priority Queue** heap structure to implement A\* search algorithm as it allows us to order the nodes by priority, whereby the shorter the “cost” to reach the goal, the more likely the algorithm is to choose the node as the next step.

The function pathfinding(board: Board)takes in Board class that pre-processes data read from an input json file and calls find\_print\_path(board: Board)that prints the solution onto the terminal.

Our first implementation was using multiple 2D arrays and linked lists to store our nodes and their respective paths, but considering the time complexity and space complexity, priority queue was a better choice overall. [code referenced from Red Blob Games[[1]](#footnote-1)] We also modified the code to utilise some Object Oriented Programming to better consolidate the information about each structure. Information regarding each class is available in the code comments and metadata.

We decided on using tuples to parse/generate the coordinates as we are using them as a key for the dictionary.

**Heuristic**

The hex cells are of same size and the cost path from one cell to another are all 1. Therefore, we believe that using Axial Hex distance is a permissible choice.

* path cost = 1 + axial hex distance, since 1 is always constant
* assume by calculating just axial hex distances gives us the best result
* overall time complexity O(n)

Axial hex distance is derived from cube system where there exists a plane at x+y+z=0 on a cube grid, with hex coordinates (q, r, s) stored and the algorithm needs to ensure that the sum of these coordinates equals to 0. It stores (q, r) and calculates s=-q-r since q+r+s=0. [heuristic referenced from Red Blob Games[[2]](#footnote-2)]

We tested our algorithm on 15 self-authored test cases and confirmed that all returns the shortest path cost (based on A\* search).

To discuss the A\* search in a more generic view,

**Challenge**

We would consider ignoring the occupied cells to find the shortest path to the goal. However, there might be a case where a successful capture result in a broken path(not continuous), then we will have to check the adjacent nodes for the two nodes to make sure that it is continuous.

The heuristics that we used would not be useful since it does not focus on ‘capturing’, which may cause problems later because they may be existing cells that are blocking the goal oath although it is still accessible via capturing.

1. https://www.redblobgames.com/pathfinding/a-star/implementation.html [↑](#footnote-ref-1)
2. https://www.redblobgames.com/grids/hexagons/ [↑](#footnote-ref-2)