Subject: COMP30024 Artificial Intelligence

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

**Implementation of A\* search**

Priority queue was used to implement A\* search algorithm as it allows us to order the nodes by priority. The function *pathfinding(board: Board)* takes in Board class that pre-processes data read from a json file input and calls *find\_print\_path(board)* that prints the solution output. Classes were used for priority queue, nodes and board (data from input json) for better visualisation and reduce time complexity.

Our first implementation was using a 2D array, but considering the time complexity and space complexity, priority queue was then used. [code adapted from Red Blob Games[[1]](#footnote-1)]

**Data Structures**

Priority Queue (used in *pathfinding*)

* priority\_queue was initialised by adding start\_node read in from json input
* loops through priority\_queue and pops the highest priority node to find the next best node within their existing adjacent nodes(cells) which are not occupied until goal is found
* *valid\_adjacent\_nodes* returns and array of adjacent nodes
* time complexity O(logn), space complexity O(n) where n is the number of nodes

Dictionary (used in Nodes class)

* came\_from stores the previous node location of a particular node
* cost\_so\_far stores the cost of a given node

Tuple

* used when storing data into dictionary as a key

Array

* *valid\_adjacent\_nodes* returns all the neighbour cells of a given node in an array
* occupied nodes read in from input json file stored in blocks does not change, therefore it is reasonable to use an array and it does not have an “origin node” (came\_from = NULL) nor a cost (cost\_so\_far = 0)

**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 straight-line distance(Euclidean distance) is a wise choice.

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

Furthermore, we tested our algorithm on 15 test cases written by ourselves and it all returns the shortest path cost. However, there was a test case where it resulted in cost that is by 1 more than our own hand calculation since the A\* search relies more on Euclidean distance than the actual cell cost.

**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 on 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)