COMP30024 A.I Part A Report

How did you formulate this game as a search problem? Explain your view of the problem in terms of states, actions, goal tests, and path cots, as relevant.

RoPaSci 360 as a search problem is observable, deterministic, episodic, non-static but discrete.

States : various board layouts with various positions of tokens

Actions: movement between tokens, the six sides of a hexagon with swing actions too.

Goal test: Remove all of Lower's Tokens using available Upper Tokens.

Path cost: 1 per move

RoPaSci have a large number of possible states with a decent number of possible actions. As there are multiple tokens, upper and lower, to be considered, it gave the problem a much more interesting but complex dimension. For instance, instead of only searching for the one best path from one token to another, all tokens can only move one space at a time as they need to consider each other's locations for the possible case of a swing action or moving into an enemy's location and killing itself.

What search algorithm does your program use to solve this problem, and why did you choose this algorithm? Comment on the algorithm's efficiency, completeness, and optimality. If you have developed heuristics to inform your search, explain them and comment on their admissibility.

Breadth-First Search. I have chosen this algorithm, as though it takes a lot of space, it is a complete algorithm not too slow considering the number of states required but also optimal as every action costs an equal amount. BFS traverses the graph by appending all neighbor nodes and exploring layer wise however, I had originally considered to convert the BFS into an A star with the heuristic function being Euclidean distance, however was not able complete in time.

How do the features of the starting configuration (the position and number of tokens) impact your program's time and space requirements? For example, discuss their connection with the branching factor and search tree depth and how these affect the time and space complexity of your algorithm.

Starting configurations in general do not impact the programs time and space requirement as with BFS algorithms, it should stay generally the same depending more on the number of possible nodes. As BFS is a complete algorithm, it will take equally as long to computer the pathway between tokens where in their state, they are relatively close then with two tokens in which their further apart. However, if the starting configuration requires a token to retrace any steps, the code cannot take that and won't consider such moves. This is a huge mistake in the algorithm which I have not been able to work around. Which is that once a node had been visited by the token it will not be considered again, however this will only work when there are only one pathway and not multiple enemies for the token to catch. I considered refreshing my visited once a token was captured however that would cause infinite loop in predecessors.