COMP30024 Assignment 1 - Report

Lucas Fern (1080613),
Hugo Lyons Keenan (XXXXXXXX)

March 30, 2021

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 costs, as relevant.

The game was thought of as a tree of board states, with the root node as the initial game state and children of each node being all board states reachable by legal moves. In this model, states are individual board states and actions are sets of moves that 'upper' can take to enter a new state. In traversing the tree, path cost is thought of as the number of moves though board-state space to get from one state to another - this is equivalent to the number of walks down branches of the tree. The win condition is met when all lower tokens have been defeated, this is the goal test.

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.

The program uses the A* algorithm to determine the most favourable actions. This search algorithm was chosen as it is complete for problems with finite state spaces like RoPaSci360, as well as being optimal when given a consistently admissible heuristic. A* uses a combined evaluation function: $f(n) = h(n)+g(n)$, where $g(n)$ is the cost to get from the starting node to node n (known) and h is the estimated cost to get from node n to the goal node. In choosing a heuristic function, $h(n)$, we wanted to guarantee admissibility so as to ensure optimality, while estimating the true cost of the path to a high degree of accuracy to increase performance. For these reasons we chose to use hexagonal Manhattan distance; the shortest path to the goal through adjacent hexagons. [lets talk more about the calculation of the heuristic, but waiting until I'm convinced ours is admissible]

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.

The time complexity of the program is exponential in the product of error in $h(n)$ and the length of the solution set of moves. This means that configurations with 'obstacles' between lower and corresponding upper tokens, as well as configurations with corresponding tokens far away from one another require more time. These also require more space, as each board state visited needs to be kept in memory to make sure the algorithm does not back track. Additionally, as the number of upper tokens in the starting configuration increases, the branching factor increases exponentially. This is because the number of possible actions from one board state is the number of terms in the cartesian product of the possible moves for each upper token. For example, with just one upper token the maximum number of actions is 6. With two it is 36, without even considering swing actions. This means that solutions for configurations with many upper tokens can be more laborious to find. Additionally, although the impact of lower tokens is not as great as upper tokens, having many can cause the solution to be very long, further increasing run time/space requirements.