AI Group Project 1 Report

Group Name : CedSam
Student Names : Jun Li Chen, Samantha Min Syuen Goh
Student IDs : 1043258, 1013524

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

States	: Board configurations
Actions	: Slide a Token to adjacent hexes / Swing around own (Upper) class Tokens
Goal tests	: Elimination of all Lower-class Tokens on the Board
Path costs	: Cost of slide/swing actions of Upper tokens, which are both equal

We decided to formulate the game into a search problem by first identifying an initial state and a goal state. The initial state is the given configuration of tokens on the board, and the goal state is the elimination of all Lower tokens on the board. As both initial state and goal state are board configurations, we viewed the intermediary states as also board configurations. Actions are slide/swing actions taken by Upper tokens to move from one board configuration to another. Path costs are the same for both actions.

2. 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.

We used a breadth-first search which goes through the coordinates of each Upper token every turn to generate a path towards their target (Lower tokens), following the RoPaSci360 mechanic. We picked this algorithm because Lower tokens(targets) do not move, and the path is only generated once when a target is specified or when the path is obstructed (by other Upper-class tokens) only. We pick the nearest Lower-class token for each Upper-class token choosing the token with the lowest Euclidean distance.

Efficiency :	Time and space efficiency is good for small numbers of tokens. Even though the time and space needed to run the algorithm increases exponentially with the length of path from a token to their target, the algorithm is run only once each time upon target initialization or when there is obstruction on the generated path.
Completeness :	The algorithm is complete since the board space is finite, and the branching factor is finite, fixed at a maximum of 8 per hex (considering valid slides and swings).
Optimality :	The algorithm is optimal because all actions have the same cost. A slide/swing action both cost the same. As total path cost never decreases as depth of the search increases, optimality is assured.

3. 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 algorithm has to run each time a new Target is found for an Upper-class token, or when the path is obstructed (by other Upper-class tokens), for each Upper-class token. Time and space needed to run 1 instance of the algorithm increases exponentially with the length of path from a token to their target, but the algorithm is only run once upon target initialization or when there is obstruction on the generated path.

Branching factor, b :	6 for each node for each adjacent hex of current hex, 8 if a swing action is possible
Depth of least cost solution, d :	length of shortest valid path from current token to the target token

Overall algorithm space complexity: O(b^d)

Overall algorithm time complexity: O(b^d)

This algorithm works fine for the scope of this project, where certain conditions like a limit of 3 Upper tokens and the fact that Lower tokens do not move, help to keep the search space small. If the number of Upper tokens increases substantially, or Lower tokens can move to evade Upper tokens, the search space becomes much bigger, and space may very quickly become an issue, as the paths stored have to now consider more tokens/more possibilities.