1. We formulated this game as a search problem by taking upper's token as initial position, finding token's enemies by Rock Paper Scissor rule (if upper = rock, enemy = scissor) and making the goal state = lower's enemy token with least distance, and we do this for each of our token.

State: the upper and lower's token's position in hex board
Action: slide (move to the adjacent 6 hexes), swing (swing to adjacent token)
Goal test: check if all lower tokens are defeated
Goal state: all of lower tokens defeated
Path cost: Each step cost 1, so the path cost is the number of steps taken.

2. We decided to implement A* path finding algorithm. We choose this algorithm because it is the most efficient path finding algorithm among other informed search algorithms. Furthermore, it is also a complete and optimal if the heuristic function h(n) is admissible.

In A* Algorithm:
Time Complexity: O(b^d), where b = branching factor = 6
Space Complexity: O(b^d), in which b^d will equal to the number of hexes.
Each hex will only be stored exactly once.
Completeness: Yes, because it will explore all of the nodes
Optimality: Yes, because it will always get the most optimal path

The heuristics function we use is Manhattan distance for 3d space / cube

$$distance = \frac{|(\mathrm{X1} - \mathrm{X2})| + |(Y1 - Y2)| + |(Z1 - Z2|}{2}$$

The function is not admissible, this is because the function does not account for swing action which may result in a cheaper actual cost than heuristic cost.

3. The branching factor is changed when our token is located on the edges of the board. This is because tokens that are on the edges have fewer neighbors than the tokens that are in the middle of the board. In addition, swing action also change the number of branches a token can have by extending the movement of the token.
On the other hand, the depth of the search will vary based on the distance of our node to the enemy node. The longer the distance, the deeper the search.

Both branching factor and depth have positive correlation with the time and space complexity of the algorithm. With more branching factor or depth, the algorithm will take more time and space.