THIS IS A EARLY INCOMPLETE DRAFT, I CAN STILL WRITE MORE

1. 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.

RoPaSci360 singleplayer is a fully observable, deterministics, episodic, static, discrete environment for an AI to process. The problem could be broken down into states including information regarding the tokens on the board, their symbol, position and player. In this singleplayer version of the game; our goal as the player or AI is to play as the Upper and defeat all lower by moving all tokens simultaneously per turn. The initial state of the game is given as the input and the goal is a state where all of the Lower tokens are defeated. The path cost of a certain solution is the number of turns taken in order to achieve the goal state if it is possible.

To formulate a solution to the game each Upper token controlled by the AI would need to find and follow a path whereby it defeats all defeatable tokens corresponding to its own symbol. This is how the game can be abstracted into a search problem, where from our current state we must search for opposing tokens from each of our tokens in order to know what action to take each turn.

1. 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.

Our program uses breadth-first search to solve the problem. We chose this algorithm as it is known to be complete and optimal for finding a path from one token to the nearest possible defeatable token while being uninformed. It’s time complexity and space complexity can be considered to be high however the maximum depth is limited to 9 and branching factor to 5 (assuming we store all previously visited nodes). This is due to the limited size of the playing board.

1. 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.

For every turn, each token has a BFS search performed. Therefore the number of upper tokens and lower tokens will increase the runtime linearly theoretically however their distances apart will mean more searches will have to be done as more turns would need to be taken. In the worst case of the given speculation, there would be 3 Upper tokens searching for 9 Lower tokens. For breadth-first search in the game, it would be searching from an Upper token in a corner of the board to a Lower token on the opposite end of the board, going to a depth of 9 with a maximum of 5 branches per node (this should be uncommon due to nodes on edges and neighbouring nodes being walked due to the .