COMP30024
PART A REPORT

Zakarya & Kamyar
DOMINATORS

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.

A: In the game of RoPaSci 360, the player needs to aim for the opponent pieces with win chances, for example, if your piece is "Paper" the aim/goal
for that piece should be "Rock". The same strategy applies to all three pieces. Finding the goal (inferior opponent's piece) is where the search algorithm plays a rule.
By applying our search algorithm, we will determine the path cost and the shortest possible path to reach the goal and then start moving all of our pieces towards it
every single turn. Defining the name of opponent pieces (R, P, S) and if there are block(s) in the path are crucially important for the algorithm to understand and
reading the board map. From our perspective, the best game player algorithm is the one who win the opponent in majority of cases with high efficiency and high speed. To achieve this state, we need to develop an efficient algorithm similar to unform cost search for our game. This will help us play the game more rapidly as well as increasing efficiency. This game is game of chance to some extend as you can't predict the player piece in real life examples. However, when the goal and initial states are known to the algorithm, this will be an inform search with increased chance of win for the player (ai agent).

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.

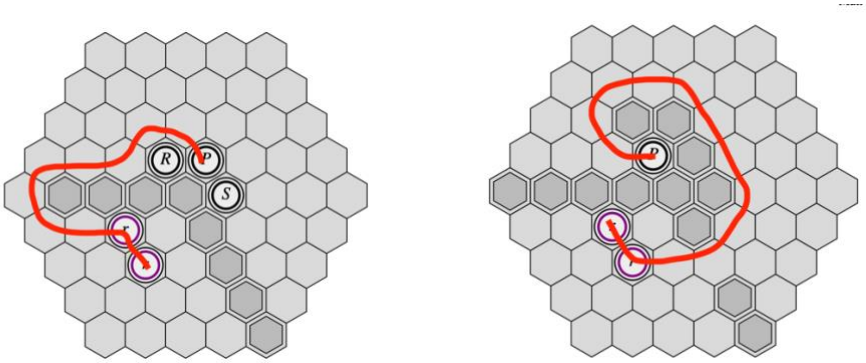
A: We have used uniform cost search (A*) algorithm to solve the search problem. UCS algorithm is the most efficient algorithm for this problem according to our research as it only calculates the cost between current node and the goal while Dijkstra's algorithm calculates the shortest path between the current node and every other node in the path. Uniform Cost Search can also be used as a Breadth First Search. The best algorithm that does not use heuristics is Uniform Cost Search. It will find the cheapest solution to any general graph. BFS will find the shallowest node first which make it different to UCS. Uniform Cost search, as the name implies, looks for branches that are roughly the same in cost. Uniform Cost Search necessitates the use of a preference queue once more. But the Depth First algo used a priority queue with the depth set to a specific value. The expansion according to least-cost and insertion in order of increasing path cost are the advantages of this algorithm. The algorithm is optimal and the time and space complexity is based on a logic of "node.g" < optimal cost. To conclude, UCS has an advantage as it is complete and

optimal although it explores option in every direction and has no information about the goal state (based on prediction).

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.

A: The number of pieces and their positions will of course, impact the complexity of every algorithm. But with UCS, the difference is that it will practice finding the least cost path based on f(g) and f(h) with f(g) playing an important role specially in cases with complicated shortest path (see examples below). With more tokens, the game would take longer while the number of times we need to generate the shortest path increases.
As the algorithm always find the least F = G +H, the number of available adjacent tokens does not effect the cost of the path and space while it may increase the calculation time. For an example, BFS expands states in order of hopes while UCS doing it in order of path cost. As the UCS will only stop if the goal step has been removed from the queue, the factors which effect the number of steps will effect the complexity of the algorithm. However, the goal state would not be expanded which boost efficiency. According to our findings and tesing, the position and number of blocks are vitallu important in changing the behaviour of the algorithm which has been dealt with a separate function for removing them from the shortest path. The more block on the way, the more complicated(longer) the path be. As a result, UCS will again be the most efficient as it will start searching from the closest path including blockage towards the closest path without blockage (the first finding will be the shortest).

- B = branching factor
- C = optimal cost
- M = minimum one-step cost
- Complexity = $= O(B^{\frac{C}{M}})$




*Examples of boards with massive blockage which is extremely delayed by Dijkstra algorithm compare to A* UCS.*