COMP30024 Project Part A Report

1 How have you formulated the game as a search problem?

A search problem is defined by state, action, goal test and path cost. In our implementation, we consider state as the positions and numbers of both white and black tokens, as well as the positions of the empty squares which are not occupied by tokens. There are two types of actions: move action and boom action. Move action is to move a white stack cardinally and boom action is to explode itself and surrounding 3×3 stacks, resulting in the removal of the white token that did the boom and any black tokens caught in its surroundings, which in turn also explode recursively. The state changes by each action made. The move actions updates the position of a white token from the old state and the boom action causes exploded stacks to be removed from the old state. The goal of the game is to remove all black tokens with 0 or more whites remaining. The path cost is 1 per move action or boom action.

2 What search algorithm does your program use to solve this problem, and why did you choose this algorithm?

The main search algorithm is breadth first search. Starting from each white stack, we use BFS to explore its neighboring squares before moving to the next level of squares. The squares explored are either empty or occupied by a white stack. These are the possible destinations for a white token. At each of these squares, we keep track of the tokens that are exploded due to initial explosion using recursion. We do so for every other white tokens recursively and added up the exploded black tokens. If the sum of exploded black tokens is equal to all black tokens on the current state while the remaining white tokens are at 0 or more, we would conclude that the game can be won by moving the white stacks to these destinations.

We chose this algorithm because in this scenario for a few reasons. Firstly, BFS is complete and it guarantees that we can explore all squares that are reachable. It can also find the optimal or shortest path to any reachable square. Secondly, BFS, in this scenario, has a lower in average time complexity and space complexity as compared to depth first search. If we interpret the game board as a tree structure, there is a small number of branches as the white stack can only move in up to 4 directions depending on how many are in the stack. This could reduce space needed. Thirdly, the tree is very deep as there is up to 64 squares (width of 8 * height of 8) to traverse but the least cost solution, which in this case, is the shortest path from the root to a destination square, is generally short. Lastly, the number of destinations that the white stacks need to move to is relatively small and DFS would take longer time to find solution than BFS.

3 What features of the problem and your program's input impact your program's time and space requirements?

Our breadth first search program requires the use of adjacency list for storing all squares on the board, which equals space complexity of $O(|V|)$. In this case, V equals b^d, where b is the branching factor which is $4 \times N^2$ where N is the number of token in a single white stack, and d is the depth of the least cost solution, which is the shortest path from the root to destinations.

The time complexity for breadth first search with adjacency list is $O(|V| + |E|)$ where V is the number of vertices and E being the number of edges. Since this is done recursively for each white stack, the time complexity becomes $O(|V| + |E|)^n$ where n is the number of stacks and the space complexity is $O(|V|)^n$.