COMP30024 – Project Part A Report

Formulate the problem

The game

- The board is a list of lists (matrix) of size 8x8.
- On a tile, a piece is present if there exists a 2-tuple (color, quantity). For example, if two black pieces is on the tile (5,4), then board[5][4] == ('black', 2).
- An empty tile has the value None, so if board[0][1] == None, then the tile (0,1) is clear.

How we see the game

- State: one state is exactly the list of lists. Obviously, any move changes the list, and produces a new signature. We convert the whole list of lists into a string, and hash that string. These take O(1) time only, but allow us to compare between states, or put states a set (Python data structure) – a convenient way to ensure identical states.
- Actions: (1) move up, down, left, right, and we can enumerate it based on the rule of the game, and (2) is the boom action.
- Goal tests: if all black pieces are completed eradicated from the board, then it is a win. Therefore, we turn our list representation to a dictionary, having two keys: 'white', and 'black'. This is similar the format available in the json file. The win state is reach, whenever we check 'black' key and it returns an empty list.
- Path cost: each move costs exactly ONE, plus the Manhattan distance from within blacks and white pieces (the further they are apart, the harder to BOOM, hence more cost).

The algorithm

We choose Iterative-deepening search, with a simple heuristic calculation. First, we thought a breadth first search is reasonable. However, the number of next nodes from a current node is large (we must enumerate next nodes by applying all possible moves and booms on a state). This could increase space complexity. Also, doing depth first search requires memories for visited nodes. We then chose iterative deepening search, with some heuristic to eliminate the states where blacks and whites are too far apart. This is pretty close to ida*, but we do not think we have successfully achieved ida* yet.

Strategy of this algorithm

- Completeness: yes – if the heuristic is calculated correctly (which is hopefully what we did). Iterative deepening search eventually visits all possible nodes. However, if the heuristic is done incorrectly, it will ignore correct nodes, hence we will never find the goal.
- Optimality: no. Since we treat each move to be a uniform cost and have not furtherly developed cost calculation. This would not guarantee optimality.
- Time complexity: $O(b^d)$ based on iterative deepening search.

Some calculations

- Maximum branching factor b: if a white stack has n elements, then the next possible moves, in 4 directions, should yield n^2. If there are k white stacks, then it should be k * n^2.
- Maximum depth should be all possible way to position n white pieces on a 8*8 table, meaning select n from 64, or $\sum_{k=1}^{n} 64Ck$. Now, 64C3 = 41164, so our algorithm quickly breaks once it reaches 3 pieces.

- There are numerous nested for-loops to convert to different data formats, however, most of them run on range(0,8) – a constant value. A 2-nested for loop generally takes O(64), which is O(1) (negligible).
- Therefore, the significant factor, is the number of white pieces supplied. Since, in the worst case, we must enumerate all possibilities of on the configurations of white pieces on the board. Our algorithm runs effectively when supplying 1 white piece (128 scenarios total: maximum 64 possible tiles on the board + 64 booms on each tile), and quickly falls out on larger numbers of white pieces.

The heuristic

Given

- n: the number of white pieces on that stack.
- k(a): the Manhattan distance from a white tile to all black stacks on the board.

The heuristic is calculated: sum(n * k(all_white_stacks)).