# COMP30024 – Project A report

Team members: Connor Cates, Jad Saliba

**How have you formulated the game as a search problem? (You could discuss how you view the problem in terms of states, actions, goal tests, and path costs, for example.)**

States:
A state is represented as a Node that keeps track of the board state and the piece information of the piece we are currently moving. Each potential move we are going to explore creates a new branch of the search tree as a Node reflecting the changes that move would have on the board. For example, if a potential move is moving 1 piece 1 tile vertically, the new Node will represent a board that has executed that move.

Actions:
An action in our algorithm is a potential move for the piece to take. The moves may be move up, down, left, right with a move distance ranging between 1 and the number of white pieces in the stack while constantly checking if the move is valid. A valid move is one that is allowed under the game specifications. As the Node branches into each of the moves it may take, it passes a list of visited locations on the board that the piece has been to, excluding any moves that would place a piece onto any of these visited locations from the search tree.

The other potential action a Node may take is to explode. When a node explodes, it explodes all pieces around it per the game specifications, and afterwards we run our algorithm from the beginning with the new board state post explosion.

Goal tests:
A goal state is one in which there are no black pieces remaining on the board. We run a goal check when we begin solving (to account for an already solved board) and any time a piece takes an "explode" action.

Path costs:
Each move taken by a piece is represented as a branch in the search tree. Each branch is given a heuristic value of the Manhattan distance from the move location to the target location of the piece.

**What search algorithm does your program use to solve this problem, and why did you choose this algorithm? (You could comment on the algorithm's efficiency, completeness, and optimality. You could explain any heuristics you may have developed to inform your search, including commenting on their admissibility.)**

Our program uses a greedy algorithm as it is an efficient way to traverse the search tree moving towards the best solution path. Prior to searching, the program first finds a list of target locations on the board. These targets are the set of all locations on the board neighboring a black tile. The targets are filtered to exclude any locations that would result in a loss of the game. Each target is put into a tuple with each white piece on the board to create a queue of target location / white piece pairs, which is then sorted by distance between the piece and the target. With this formed queue, the search algorithm will try to move the piece closest to the best target location first, which is more likely to result in a final solution.

By keeping track of visited locations, this search is complete in finite spaces, but is not optimal as it does not take into account barriers between the piece and the target location, such as black pieces on the board.

**What features of the problem and your program's input impact your program's time and space requirements? (You might discuss the branching factor and depth of your search tree, and explain any other features of the input which affect the time and space complexity of your algorithm.)**

Since the input may have up to 12 black pieces on the board, this increases the target locations and thus increases the number of calculations in regards to the targets. Similarly, there are up to 12 white pieces which increases the number of start states we have in our queue, which is equal to (number of whites) x (number of targets).

Expected time requirement: $O(b^m)$
Expected space requirement: $O(b^m)$