Going through the specification, we deduced that this game is fully observable as we can see the whole board, deterministic as the current state and our next action fully determines the next state, static as it is only us (the white player) that will move, discrete as there is only a finite number of moves that can be made (1 boom and up to $4n^2$ moves per stack – this will be explained further on), and sequential as the previous states or actions will determine future states and actions in the game. With that, we decided that this game can be solved offline using a simulation of exploration of the state space since we are the only ones moving in the game. This will be in the form of a search tree where the initial root will be the initial state that we started the game in, the leaf nodes are the potential children created by either moving or booming, and the goal node being a state where there are no more black pieces on the board/state. In our search problem, states were defined as a list of the locations of the white and black piece(s) – a stack was considered a piece in our case – remaining on the board which were stored in the form of a dictionary (the same that was parsed into the function as input, with "white" and "black" as keys and a list of the locations of the remaining pieces as its values. The goal is to remove all the black pieces from the board, and this is tested by checking if the list of the location of the black pieces is empty. The actions make up the path from the initial state to the goal state with each action (defined as either a boom or a valid move) being counted a 1 per action.

To efficiently search for the optimal path, we used the A* search algorithm. We chose this algorithm as this problem was best suited for an informed search algorithm since we know where all the remaining black pieces are and can therefore calculate the distances/number of moves needed to reach it. Because we can only move in the 4 cardinal directions, we used the Manhattan distance as our heuristic function. We also made sure to account for the fact that the white piece cannot be on the same space as the black piece, which mean the closest distance would never be 0, and we also considered the fact that the white piece does not need to be right next to a black piece when booming as it can be at its diagonal. We also prioritised stacking as being in a stack would allow the pieces to move further in less steps. This allows for the algorithm to be admissible as it does not overestimate the true cost. With these rules in place, the algorithm is therefore very efficient. When deciding on which node to expand on, we set our evaluation function to be the sum of the cost to get to the potential node from the initial node and our heuristic value. Hence, our algorithm is optimal. Also, because all the actions cost the same and the way that our heuristic and evaluation function is written, there will not be an infinite number of nodes in our state space as we cannot return to any of our previous states as the evaluation function will deem it to be worse and will therefore not expand on it. Thus, our algorithm is also complete as it will always find a solution if one exists.

That said, there are some downsides to this algorithm. Due to the fact that each piece can have 5 actions (4 moves and 1 boom), that would lead to 5 children states being created. These children states would have to be looped through and evaluated before being added into our list of nodes to expand on. This number is further increased if the piece where to be in a stack as it could then move up to n steps using up to n pieces in the 4 directions (16 moves and 1 boom if in a stack of 2 pieces). This would lead to an exponential growth in the number of children states being created before we evaluate them, and resultantly, the branching factor could be up to $b = 4n^2$ for each stack. Furthermore, because we would keep expanding the nodes until we find a solution (if one exists), the maximum depth of the tree would be the length of the solution, say m in this case. Also, being an informed search algorithm would mean that it would have to store all of its nodes that it will/had expanded on. This would mean that for both the time complexity and space complexity, it would be $O(b^m)$. However, since this particular assignment only has states with up to a maximum of 3 white pieces that we potentially stack, we have decided that the pros outweigh the cons of this algorithm by a huge margin, and hence why we chose to use the A* search algorithm as the search algorithm in our Expendibots games.