Some definition:
In report, we define a set of black tokens as a 'group' if any of black token in group explode, then all group explode.
We also define a coordinate as 'explosion point' of group, if one white token explodes at the coordinate, the whole group will explode.

Strategy of the game:
We have two strategies in this game.
Strategy one: this strategy is the main part of the code. In this strategy, all the explosion happens at the end, and there may exists some required moves before the explosion. We divide this strategy into two cases.
Case1: There is no need to form stacks.
Case2: Case with a need to form stacks.
In each case we will use search algorithm to search the route for stacks and tokens. Note that almost all case2 will be developed to case 1 in the end, because after we move stack, we should token from separate it to reach another explosion point.
Strategy two: Tokens needs to be exploded first after serval moves since stagey one is not applicable. After explosion, the situation will become strategy one.

1. **How have you formulated the game as a search problem?**
   We formulate the game as a search problem.
   States: As the environment is fully observable. We obtain the states only by reading at the coordinate of black and white token. State refers to the positions of white and black tokens on the board.
   We will update the states after finding a route from white stacks to a right coordinate.
   Actions: action in search is a route consists set of coordinates, which will lead the stack to the explosion point.
   Path cost: we did not evaluate the path cost of action, as there is no requirement on it, but it may be the total route length (including stack) in each step if we have to evaluate the path cost.
   Goal test: goal test is simply checked if we have reached the right point(explosion_point), its explicit.

2. **What search algorithm does your program use to solve this problem, and why did you choose this algorithm?**
   The core search algorithm in project is a function called 'simple_BFS', It is an offline search algorithm. This algorithm is chosen since it is the first idea we got.
   It has a BFS structure, but we made some improvements on "expanding the node" so that it can handle stack (more than one token) move.
   It will judge if there is a route between two coordinates and return the route.
   This function has multiple uses. Rather than finding the route from a stack to the right coordinate, it can also check if two coordinates are connected. (Like we used in function 'explosion_small_group'). If they are connected, it means we if we can reach one, we can also reach another one.
   Completeness: It is complete if there always exists a path in the input. The algorithm does not

guarantee to find a path between to coordinate as there may be no path at all.
Optimality: It is not optimal, since cost in every step is different. There is no heuristic. The algorithm just goes through the board.

In other functions like explosion group, we also adopt BFS (Starting from a single black token, explore all the surrounding black tokens and their children) It is complete if number of black tokens are finite.

3. **What features of the problem and your program's input impact your program's time and space requirements?**
Time complexity:
For simple_BFS, the maximum branching factor is 4*distance, where 'distance' is the maximum distance a stack can move.
However, since it's a search coordinate algorithm, it will end if all nodes are visited, so the time complexity is O(m-n), where m is number of coordinates on board and n is the number of coordinates occupied by black token.
As for the time complexity for whole algorithm, we run this function on every white stack, so larger numbers of white tokens in the problem could be time-consuming.
Space complexity:
We create a list to store the explosion_point of black group, and the explosion_point of black group may overlap. The space complexity is O(m*j), where m is the coordinate on board (explosion point in each group), j is the group count. So, if there is a large scale of overlapping between each group and a large group count, it would require a significantly larger space to require.

In general, we use BFS algorithm serval times. So as a result, if there exist too many leaf nodes or the tree is much deeper than expected, it can consume a lot of memory spaces.