1. For exact solution, the DFS method is used. For each board and each red token, branch all possible moves of it. The DFS tree max height is bounded by the size of the greedy solution. The details are listed as code comments. A list of tuples is used to store the best and intermediate solutions.

1. We use a greedy solution to upper bound the search tree height. For the greedy solution, in each round, we find the blue and red token pairs that are closest to each other (use the spread action to cover some distance) and move the red token towards the blue one. For the test.csv, with greedy solution as upper bound, DFS solution terminates in 1s. Without it, the DFS does not finish in 1min.

3. With spawn action enabled, the complexity of the algorithm is drastically increased since possible moves are increased by the number of all empty cells. Considering spawn a token next to blue token with high power could be an interesting strategy. Particularly, for our A\* heuristic, we can modify our cost function. Without spawn action, the cost of empty cell is treated as infinity since it is impossible to spawn the token on empty cell. Now, for each empty cell, we can also estimate its cost and increase the cost by the minimum cost of all non-empty cell. The rationale behind is that for a newly spawned token, to reach the target destination, is has to go through two stages: 1. spawn. 2.move to destination. The spawn cost could be covered by the minimum cost of all non-empty cell since we give up a chance to move and choose to spawn. If an empty cell has the lowest cost, it could indicate that all other existing tokens are far away for the target while there is an empty cell that could potential be spawned at with even lower cost to the target.