The report for project part A
Shizhan Xu 771900
Li Shangqian 908462

The first part of the project is quite simple comparatively. Since the blocks can't move, all we need to consider is finding a currently shortest path towards the exits. Hence this falls into a search problem where the states are the spaces on the board. The actions are either jump or move where both with uniform cost 1. The goal test is to check whether a piece reaches any exit space. The path cost is not considered since we move 1 step a time.

Strategy for single piece:

The branching factor of this problem is 6 since from 1 space we can access to at most 6 other spaces. Let b be the maximum branching factor, d be the depth of the least-cost path, m be the maximum depth of the state stage.

There are totally 37 spaces on the board, which means m of this problem is 36 if we never visit a visited node. Much higher than the depth of the shortest path d even if there are many blocks on the board. Since m is much greater than d, depth first search doesn't suit this problem. Breadth first search and iterative deepening search are the better choices in terms of the completeness and runtime. These two strategies are both complete for this problem. As the cost of each move are uniform cost 1, both algorithms are optimal. Furthermore, the time and space are O(b^d) and O(bd) for both algorithms, which are more efficienct than the depth first search. Since we have never tried the iterative deepening search before, we chose iterative deepening search as our strategy for this problem.

The space requirement of my strategy is minimized since it's based on depth first search. The time complexity is branching factor to the power of depth of closest route, so if the blocks are placed intentionally to have some deep dead-ends as well as a deeper closest solution, the time spent could be potentially high. However, it's hard to make both d and b high in this game, since we only have 37 blocks and 6 maximum branches, so I believe the average time spent would be within the requirement.

Strategy for multiple pieces:

Iterative deepening search is extraordinarily suitable for solving shortest path for a single piece. However, it is not suitable for multiple pieces. We attempted to solve this issue by using A* algorithm. We used iterative deepening search as the heuristic for A*. It appears that the space and time requirements have been a big issue.

Let n be the number of pieces on board.

In our design, we run the heuristic to find the best move at the current stage after every movement so that we can find the overall optimal solution. This require huge run time and space. The next move is found by calculating the least sequence of movement. For example, find least cost among move sequence “A,B,C”, “B,C,A”… The reason to do this is because moving one piece to a new space may bring a chance for another piece to jump, which is fairly unpredictable based on our design. The resulting time will roughly equal to O((nb^d)^d) and the space is O((nbd)^d), which are huge. The code does not work when dealing with extremely complex board.

One possible solution is to not use iterative deepening search as our heuristic function but choose a more suitable and less time-consuming algorithm.