**Complexity analysis**

1. For solve function,

* The solve function can iterate for the worst case of an n times of steps for each state move.
* While the maximum number of actions is 6 for the board size of 3\*3, we can consider m as number of legal actions.
* The inner loop (m) is a much smaller scale then the outer loop (n), the time complexity of this function is O(n\*m) = O(n)
* OR if the size of n and m are at similar scale, the time complexity can be considered O(n^2)

2. For the solve function with multiple goals,

* If it has the maximum number of steps, it will have the same time complexity of O(n) as the program ends after n steps.
* If it does not have maximum steps, the worst-case would be O(m\*n^p), where m is the maximum depth of the search tree, n is the number of possible actions at each state, and p is the number of goals to be checked.
* For multiple goals solver, I have decided to implement with a recursive function for checking all the goals as an experiment. This can be less efficient than while loop implementation and uses the stack memory for each successive recursive function call.

3. The use of the priority queue and the list to keep track of visited states could affect the performance of the algorithm, but these data structures generally have good time complexities for their operations.

For example:

* + Priority queue has a O(1) when calling the top element and O(log n) for popping and pushing an element into the priority queue
  + List is used for visited states, the operation for comparing the target state to the states in the list is O(n). All standard operations for list STL has O(1) e.g. push back, push front, pop, insert etc.

Note: This complexity analysis does not include smaller iterator loops.

For example,

* The use of hash table (unordered map STL) for checking for extra goals.
* Heuristic function
* Finding legal actions
* Moving the block
* Pushing down the grid
* Etc.