The report

Author: Lanruo Su, He Shen

1. With reference to the lectures, which search strategy did you use? Discuss implementation details, including choice of relevant data structures, and analyse the time/space complexity of your solution.

We use A* search algorithm in our project. In the lecture, we learned that A* can generally avoid expensive expanding on future paths, thus saving lots of unnecessary actions, resources, and time to search a path. Meanwhile, A* search is theoretically designed for finding optimal solutions as well. Regarding the specifications of this project, finding the optimal placements and strict time controls are listed under the requirements. Thus, using and implementing A* is a great choice to satisfy the above conditions.

In our A* implementation, we use “PriorityQueue” to store each available node with different game status. In order to expand the node with the least cost, we store its current path cost (g(n)), total cost (g(n) + h(n)), place action list, and the current board itself. The estimated cost to reach the goal, h(n), is designed to calculate the minimal cost from current node to the target goal, using a combination of both Manhattan distance to goal and empty distances in row and column of the goal, with a percentage ratio to adjust the range and size of the minimal cost for a more precise and optimal output. Apart from that, since the board is considered as an infinitely repeating plane, the coordinate is being adjusted using “adjust_coord” if it is coming out of the edges, and there is also a “frozenset” to record an already-accessed board to avoid repeating.

The time complexity of the A* search is depending on the accuracy and values from the heuristic function. In the average case, when heuristic can give a meaningful value, the time and space complexity depend on the total number of nodes that need to be explored and expanded, which might be close to O(n). However, in the worst case, where the heuristic returns the exact same value, A* will try to expand both of these possibilities until reaching the goal. Therefore, the algorithms may eventually traverse through every branch until its depth, resulting in the time complexity of O(b^d). Under this case, A* also needs to store every node in the memory, hence its space complexity will also be O(b^d).

2. If you used a heuristic as part of your approach, clearly state how it is computed and show that it speeds up the search (in general) without compromising optimality. If you did not use a heuristic-based approach, justify this choice.

We have used heuristic function in our A* search algorithm. Generally, the heuristic-based approach estimates the cost of reaching the goal from a given state, which prioritizes exploring paths that are more likely to lead to the goal and speed up the search process. In our implementation, we first count and record the empty cells in the target goal’s row and column, combine them and make it as a percentage ratio to adjust accuracy depending on the board size. Then, we calculate the Manhattan distance from each of the red blocks to the goal, and

find out which one contains the minimal value. After that, we choose the row or the column with the least empty cells, and add its value with the previous Manhattan distance to form a raw heuristic cost. While in many cases the raw value can be seen as admissible for solving the place actions, it does not always help producing the optimal solutions in A* search due to its overloaded size, especially in a 11x11 size board. Hence, we times a ratio to the raw value to help control its size range, and thus maintain heuristic cost's admissibility and accuracy.

With the involving of the heuristic cost, A* search can perform faster and skillfully. Like mentioned above, our heuristic function calculates the minimal Manhattan distance and considers whether the row or the column contains smaller numbers of blank cells. This greatly improves the efficiency of choosing the starting point, and the direction of place actions afterwards. It has fulfilled our expected mission of prioritizing exploration well, while reducing the exploration of irrelevant paths, reducing arbitrary nodes, thus accelerating the entire A* search process.

Upon that, the optimality is also well guaranteed. To make sure the heuristic cost is admissible, where the estimated cost h(n) will always be less or equal to the real cost h*(n), a percentage ratio will participate in the process of calculation. It helps to control the final heuristic cost to a reasonable range, and make sure it will not overestimate the actual cost.

3. Imagine that *all* Blue tokens had to be removed from the board to secure a win (not just one target coordinate). Discuss how this impacts the complexity of the search problem in general, and how your team's solution would need to be modified in order to accommodate it.

The overall complexity will be significantly increased under this circumstance. Since we need to remove all the blue blocks instead of one targeted blue block, it leads to much more possibilities and combinations on different paths to take place actions. Under the worst case, when the heuristic cannot handle such sophisticated situation and the cost gives no useful guidance in A* search process, the time and space complexity will be O(b^d). Even though its Big-O model is the same as the original case, and other parts like generating successor actions or apply place action remain the same complexity, with more branches and depth, the overall complexity grows exponentially. The time and space will both be consumed more in order to keep the optimality.

Some modifications can be made to our program to accommodate this new requirement, while also applying some optimizations to avoid the previous discussed complexity impacts. For the final goal, we need to change the termination condition from detection on removing the target to examine whether there are no further blue blocks remaining on the board. In order to accomplish this new search process, we need to define a new heuristic function which can evaluate the efficiency to get to clear the appointed or all remaining blues instead of simple singleton distance to each of the blue ones. In addition to the adjustments on the heuristic function, we may implement some pruning methods to control the search complexity, one way is to construct a closed list to track the evaluated nodes. This can be helpful in avoiding the future evaluation of these already discovered status, thus reducing time and space complexity.