Assignment1 part A Group report

Gaoyuan Hao 1245722
Yilin Chen 1239841

Strategy and data structure chosen

In this assignment, our team decided to employ the A* search algorithm and chose node and priority queue as the primary data structures. We utilized nodes to store the current cost of the chess piece, which includes the cost from the initial node to this position and its f(x) value, the current board information, and the action used to reach this position. Priority queues were employed to sort based on f(x) values, allowing us to select the minimum f(x) for expansion until the goal state is reached and the optimal solution is obtained.

Implementation details and complexity

Within the A* algorithm, we designed the heuristic as follows: get the Manhattan distance between the two closest red and blue chess pieces on the current board and divided it by the chess piece's token power (k), then add the rest number of blue chess to derive the h(x) value. Since the spread range of a chess piece dynamically varies between 1 and 6 based on its k value, we divided the Manhattan distance by k to make the heuristic as admissible as possible. We used the cost recorded in the node to represent g(x), which is the number of steps required to move from the initial node of the tree to the current state.

As we used the A* algorithm, it is difficult to obtain specific calculations for time and space complexity, which instead vary dynamically based on our heuristic equation. Broadly speaking, the time complexity going to exponentially increases, while the space complexity is approximately O(b^d), where b is the number of branches and d is the tree depth. In our A* algorithm, the number of branches varies with the heuristic. As we only expand a few branches with the smallest values, the actual number of nodes stored in memory should be less than b^d. In our code testing, using our designed algorithm and heuristic, we only needed to expand 11 times for visible test 1 and 7 times for visible test 2, compared to 10,226 times which use the bfs search algorithm but not our designed heuristic function. This demonstrating that our team's heuristic design is effective and have highly efficient.

Discussion of the use of SPAWN actions

If the spawn action is allowed in single-player Infexion and the spread action remains usable, the complexity of the search problem would increase substantially. This is because the spawn action allows placing chess pieces anywhere on the board, ensuring that a piece can be captured in two steps by placing it next to the target and then using spread. If a spread action is used and no piece can be captured within two steps in all directions, spawn can be employed as an alternative. This increases the algorithm's complexity, as the existence of spawn action increases the number of branches and potential positions on the entire board. This not only results in a significant increase in the number of branches to be predicted but also necessitates the re-optimization of our heuristic function, as the position of the spawned piece itself will impact the efficiency of capturing other pieces after one has been captured. We believe that using the total straight-line distance between the forthcoming red piece and all blue pieces as a heuristic might be an effective design, as it can help spawn action to determine which step is more reasonable.