Search Strategy and Implementation Details

This implementation uses A-star Search to find the path for a single red frog to move from the initial row (near row 0) to the bottom row (row 7). Using Python's ‘heap q’ as the priority queue, where a triplet of (f_value, unique_id, position) is placed, and the smaller the f_value, the more likely it is to be retrieved. We also used close_list which is set to record the node we have visited to prevent visiting existing node. And a dictionary called g_values to record the actual cost form the starting position to current position, and another dictionary called came_from is used to track the predecessor nodes and corresponding movements required for path reconstruction to return as our answer.

Assuming in the worst-case scenario, it may be necessary to traverse the vast majority of feasible chessboard states. If used ‘b’ represents the branching factor (the number of next steps that can be unfolded at each position), ‘d’ represents the search depth, and the theoretical time complexity of A-star is approximately O(b^d). The spatial complexity is usually on the same level, as it requires storing all expanded nodes.

The worst-case time complexity looks the same as Breadth First Search or Iterative Deepening Search, however due to the use of heuristics, A-star can be more efficient in many cases, and can actually significantly reduce the number of nodes that need to be expanded.

Design of Heuristic Function

The goal of heuristic function is to predict the cost is large enough but cannot exceed the actual cost which is admissible. We compute the maximum row that can be reached in a single move from the current state, then estimate how many moves are needed to reach row 7.

In detail, we use get_possible_maves function to obtain all possible positions that can be reached in the next step from the current position, find the position with the largest row coordinate. Finally, take (7 - current_row), divide it by this maximum possible advancement, and apply the ceiling function to determine the estimated number of moves required.

Six Red Frogs Scenario

The time and space complexity grows exponentially. If only one frog can be moved at each step, the branching factor may be approximately "6 (number of selectable frogs) x b (number of ways a single frog can move)", that is 6b. In the worst-case scenario, the search tree depth may approach the total number of steps taken by each of the six frogs.

In the case of multiple red frogs, nodes are no longer just the coordinates of individual frogs, but the state of the entire chessboard. This state will include the positions of all frogs (whether red or blue) on the chessboard, thus fully reflecting all possible movements and their interactions.

Since the goal changes from one red frog reaches row 7 to all six red frogs reach row 7. We can still use A–star search, but we need to redesign our heuristic function by considering the 'collective' cost of multiple red frogs simultaneously.