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.

I used Breadth-First Search (BFS). This guarantees the shortest path in an unweighted grid-based problem. BFS explores each possible state once, marking visited positions to prevent cycles. My solution uses the data structure of a double-ended queue which allows me to add elements to the end of the queue and remove and return from the front. The worst case time and space complexity is $O(N^2)$. I considered this feasible for such a small board size, however as the board grew, BFS would become too slow and memory-intensive.

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.

I did not use a heuristic because I considered that for a simple problem like this, with such a small grid size, it was not necessary. BFS results in some inefficiencies due to the fact that it stores and expands all possible paths at the same rate, even if obstacles or a more direct path exists. If the grid was larger, I might have implemented A-star search, using either Manhattan distance as a heuristic or a more informed combined heuristic. Manhattan distance is admissible but not very informed, as it does not consider obstacles, or the fact that paths that include a string of hops outperform a series of adjacent steps.

3. Imagine that all six Red frogs had to be moved to the other side of the board to secure a win (not just one). Discuss how this impacts the nature of the search problem, and how your team's solution would need to be modified in order to accommodate it.

If all six frogs had to move to the other size, the problem would become a multi-agent coordination problem, rather than a single-agent search problem. The space needed to track state would become exponentially larger, meaning that breadth first search would no longer be practical. The solution would also need to be modified to consider path dependencies and avoid invalid moves that would cause frog collision. I could still plan each frog individually using A* but would need to add a constraint and replan recursively any time two frogs' paths conflicted, building a constraint tree that branches when conflicts occur. A conflict-based search would provide a complete and optimal solution, but would still be exponential complexity in the worst case.