Project Part A – Report

COMP30024

Sounak Bhandari

Zuhair Mobasshar

I.

In our solution for the single player Freckers game, we have implemented an A* search algorithm. We've used this because it is a heuristic-based search algorithm that prioritizes exploration of the most promising states. The frontier is implemented as a priority queue using the *heapq* module in Python, with each element in the queue being a tuple containing *f(n)*, *g(n)*, a counter, the current position of the frog, the board state, and the path taken. In this case, *f(n) = g(n) + h(n)* is the total estimated cost, where *g(n)* is the number of moves so far (current cost), and *h(n)* is the heuristic function which estimates the cost to reach the goal. The counter ensures tie-breaking for states with the same *f(n)*.

The heuristic function that we've chosen is one that calculates the number of rows remaining to reach the last row. This is simple and admissible because it never overestimates the actual cost.

We've also implemented a visited set, which is used to avoid revisiting the same board state from the same position by combining the frog's position and the board state, as a *frozenset* of coordinate mappings, into the *board_key* to uniquely identify a state. Using a *frozenset* allows us to represent the board state while ensuring immutability and efficient hashing.

The *get_moves* function evaluates single step moves as well as legal jump sequences generated by the *jump_sequences* function and adds them all to a list of moves. All these moves are simulated and added as possible options along with their costs to the frontier priority queue. The search then terminates when we either reach the last row (solution) or exhaust the search space.

The priority queue implemented using the *heapq* module allows us to ensure that the state with the lowest estimated total cost is always expanded first. It also allows for efficient insertion and extraction of the minimum element in *O(log n)* time complexity.

In the worst case, the A* search algorithm will explore all possible states, which will lead to a time complexity of *O(b^d)*, where *b* is the branching factor and *d* is the depth of the solution. Our heuristic will reduce the effective branching factor by prioritizing more promising states, but different board configurations will lead to different levels of improvement.

Each state is a copy of the board, which will require *~O(64)* space. The priority queue stores all states that are yet to be explored, which means that the size of the frontier is proportional to the number of states in the search tree. On the other hand, the visited set will have a size proportional to the number of unique states explored. This means that the main factors in the space complexity here are the frontier and the visited set, and thus the space complexity will be *O(b^d)*.

II.

The heuristic that our team decided to use computes the number of rows left to traverse until the final row. This is calculated by subtracting the current row coordinate from the final row coordinate. It is the minimum number of moves required assuming no obstacles, so it does not overestimate and is thus admissible. The total estimated cost is calculated by adding the heuristic to the current cost so far, and states with lower estimated costs are searched first. This reduces the explored states and speeds up the search because it guides the search toward the goal instead of having it explore all possible states in a breadth-first manner, which would be very time and space consuming. We also know that the heuristic is consistent because it always gives the same final cost once a state is expanded. Therefore, we can say that our heuristic does not compromise optimality and ensures that the search always finds the shortest sequence of moves to achieve our goal.

III.

If all six frogs had to be moved to the other side, the nature of the problem would change drastically and become significantly more complex. The state space would grow exponentially to keep track of all frogs, the frogs would have to have coordinated movements to avoid blocks or deadlocks, the goal condition would have to be changed and certain frogs might have to be prioritized over others at different stages, the branching factor would become much larger because each frog can generate its own set of moves, some frogs might be dependent on other frogs making their moves first, and the time complexity would increase significantly due to the large branching factor and state space.

In our solution, we would have to change our state representation to include the positions of all frogs, update the move generation to generate moves for all six frogs, update the state transitions to handle multiple frogs moving in one transition, modify the frontier to track all six frogs' positions, and change the heuristic to consider the progress of all frogs together. We would also need to add new logic that deals with deadlocks.

References

Leckie, Chris. 2025. “Problem Solving and Search – COMP30024.” Lecture, University of Melbourne, Parkville VIC, March 12.

Leckie, Chris. 2025. “Informed Search Algorithms – COMP30024.” Lecture, University of Melbourne, Parkville VIC, March 19.