Artificial Intelligence Project A Report

Strategy

Referenced to week 3's first lecture. We used A* search algorithm in the search.py for its guaranteed optimality with the choice of using Manhattan Distance as our heuristic function's metric. The h(n) we chose is admissible leading to A* search being optimal as when it progresses it is only expanding and exploring on better or cheaper solutions.

Implementation

Start by finding the red frog's position and identifying valid ending positions where there exists a lily pad at the cell at the last row. Then we create a priority queue and insert the start node (red frog's cell).

In the main A* search (pathfinding) loop, next nodes are explored depending on their f value which is h(n) + g(n), h(n) being the heuristic cost we chose and g(n) being the actual cost so far. For each node, we check possible moves in possible directions (red_directions) with other checks like valid_landing_spot and can_jump as listed in the project rules. Once a path to the end row is found, then we use retrace_path to reconstruct the solution by tracing backward from the goal node we reached.

Data Structures

- A Node class: to keep track of the data of each node, storing the coord, parent node, g(n), h(n), f(n) costs, move direction and a flag to check for jumps.
- Linked List: To trace back, collect and reconstruct the optimal path obtained.
- Priority Queue: implemented using heapq to facilitate fast ordered insertion and finding the node with lowest f_cost to explore.
- Set: used in closed_list to track visited node in O(1) time for best efficiency.
- Dictionary: board cells and its state, use hash map for O (1) data access, while minimising unnecessary space allocation.
- Also with existing structs implemented in core.py.

Heuristic Function

We use Manhattan distance as the metric because of red frog's directional movements. The heuristic function computes the minimum Manhattan distance from the red frog's position to the destination. Admissible because it is always the shortest distance possible for the red frog to reach a cell allowed at the final row with a minimum number of grid movements required.

It never overestimates the true cost of the actual path (h(n) <= h*(n)) thus being admissible and lead to the A* search algorithm in this case being optimal.

Time and Space Complexity

A* algorithm's time complexity is O(b^d) where b is the branching factor and d is the depth of the solution.

In project A, the branching factor is bounded by the number of possible moves which is at most 5 from red_directions in part a. For the 8x8 board we have, the depth could be O(n^2) where n is the number of rows or columns and equals 8 in this case, in the worst case where the red frog possibly needs to traverse almost all cells to find a path.

For other operations:

- Checks for next moves like is_on_board, can_jump etc., they all have O(1) complexity which is constant.
- Computing the heuristic (Manhattan Distance): can treat as O(1) constant.
- Heap operations in action_list: O(logm) with m being the size of the priority queue.
- Set operations in closed_list: O(1) average.

Overall, the **Time Complexity** would be O(n^2 logn).

For **Space Complexity**, the action_list can have at most O(n^2) nodes, the closed_list can also grow to O(n^2), the path reconstruction space is at most O(n^2) in the worst case. Therefore, the overall space complexity is O(n^2).

Imagine all six red frogs need to move to the other side to win the game. Discuss the impacts and modifications needed.

- In this case, moving one red frog could block possible paths for other frogs and we will need to consider all six frogs' potential paths to determine one that is going to be optimal and the best to move all frogs across the board.
- Branching factor will grow largely, b = 5 when there is one red frog present on the board, and it has 5 possible movement directions. With all six frogs, b = 30 at max.
- Red frogs could potentially run into a deadlock situation where one blocks another's path and could not explore any further.
- The Manhattan Distance Heuristic we used might not be sufficient or useful in this scenario.