COMP30024 Artificial Intelligence Part A Report

Overview

For part A of this assignment, our team has used an implementation of A* search algorithm, where the evaluation function $f(x) = g(x) + h(x)$.

Firstly, we set up a priority queue using the Python queue library. With every iteration, we will pop the value with the lowest priority, check whether it contains our goal state (where there are only red cells on the board), else we expand it.

We also define a custom class to hold our nodes called "boardstate", which contains its current board information, the parent node, the last move performed to reach this node, and the number of moves taken to reach this node.

Expansion of the nodes involves a slightly complicated process, where an "optimal cell" is selected out of all available red cells, and that cell is then spread 6 different directions, in total generating 6 possible nodes. To select the "optimal cell", we calculate the total number of blue cells that are within the spreading range of any given red cell. The cell with the highest number of blue cells within its range is then selected as our optimal cell (can be seen in selectOptimalCell function).

After the "optimal cell" is selected, 6 nodes are generated, one for each potential direction the cell is spread in (seen in ExpandNodes function), and each child node is assigned a priority score, and then inserted into the priority queue.

Space Time Complexity

The time complexity is exponential, towards the depth of the solution, $O(b^d)$. In this case, there is a branching factor of 6 since we create 6 nodes from a given node and the maximum depth will be the depth of the solution, as our heuristic will likely guide the search algorithm towards the solution.

The space complexity is $O(b^d)$, as the algorithm keeps all the generated nodes in memory, to be able to backtrack and find the path to the goal state, up until it finds a solution at depth d.

Heuristic

In A* search, the evaluation function is defined as $f(x) = g(x) + h(x)$.

For our heuristic h(x), we went for the Euclidean distance between blue cells and their closest red cell. The minimum value for each pair is 1, whilst the maximum is 12. We take sum of the absolute difference of q and r values between the blue and red cell, and the total sum for

all pairs is assigned as a node's h(X). This heuristic allows us to guide the search towards the branches that SPREAD the red cells towards the blue cells and eventually capture them to form a h(x) of 0.

The g(x), which is the cumulative cost of the path taken, is calculated in the number of moves. To ensure that the h(x) always underestimates the true cost, and that g(x) has the highest weightage to ensure optimality of the solution, we give each move a value of 300. We derive this value from the absolute worst case scenario of the heuristic, where each side has 25/24 cells, and they are all a maximum distance from one another, 25 * 12 = 300. By giving each move a score of 300, we ensure the heuristic is admissible as it will always underestimate the cost of a move. This allows us to find a solution with least possible moves whilst having the heuristic guide us, without compromising optimality.

The logic in choosing this heuristic is that due to the snowball effect, it would be beneficial to take opposing cells as soon as possible, giving us a greater number of cells to work with (enemy cell's power + 1 due to our spread), which overall will likely result in faster searches. This is achieved in the program by prioritising reducing the distance between red and blue cells as much as possible, guided by h(x).

Spawn Action Possibility

In Part A of the project, we are only allowed to use SPREAD. However, if SPAWN was a possibility, it would likely improve the search algorithm greatly. In general, to infect an opposing cell capitalising on SPAWN requires 2 moves, spawning the cell beside our target cell, then spreading in the direction of our target cell. This means that any opposing cells that require more than 2 moves to spread to can be taken using this approach.

The aforementioned snowball effect may also further strengthen with the ability to SPREAD, as a cluster of opposing cells can be quickly infected if a cell is SPAWN'd beside it and starts spreading to it.

In terms of specific modifications of code, we would likely adjust our heuristic, firstly we would expand all the cells that can capture opposing cells in a single move. After which, we would begin spawning cells for opposing cells that require more than 1 move to capture, to limit the number of moves needed to capture to 2.