COMP30024 Artificial Intelligence Project 1 Report

team member: XuLin Yang(904904), Liguo Chen(851090)

Search Problem Formulation

- ***State***: player pieces and obstacle pieces' location on board
- ***Action***: player can move, jump or exit one player piece per turn defined in specification
- ***Goal Test***: no player's piece on board
- ***Path Cost***: 1 cost per action

Search Algorithms

Terminology

- ***b***: branching factor for search tree
- ***d***: length for the solution path in search tree
- ***δ***: relative error in heuristic = |h*(s) - h(s)|
- ***$dist_{SLD}$()***: straight line distance in hexe
- ***h()***: heuristic function

A* search

- This is the search algorithm used in our program. It's a simple but efficient search algorithm. Not only is it complete and optimal, but also optimally efficient, meaning among similar algorithms(ones that expands paths using heuristic as a guide), which use the same heuristic, A* expands the least(or as least as others) number of nodes(states in this project).
- Time Complexity
    - best case ∈ O(d) if we disregard the complexity of the heuristic calculation
    - average case ∈ $O(b^{\delta d})$ (from lecture)
    - worst case ∈ $O(b^{d})$ (because it is uniform cost search now)
- Space Complexity ∈ $O(b^{\delta d})$ (because "keep all nodes in memory")
- Completeness
    - A* search is guaranteed to find a solution if one exists. As it is guaranteed in the specification that there is at least one solution, A* search is complete in the project.
- Optimality
    - Yes, as long as h(s) ≤ h*(s) ∀ s ∈ state space*
      (A search is optimal if the heuristic is admissible(required in tree search) and consistent(required in graph search))

Heuristic Function

- h(state) = $\sum{piece ∈ player} (\lceil \frac{dist{SLD} (piece)}{2} \rceil + 1)$
- Admissibility:
    - Discussing Red player is similar for Green and Blue player as they are parallel cases. So we can only discuss Red player case at here.
      Fastest path for a single piece on board to reach in goal hexe is that the piece can jump to goal hexe as much as possible (optionally plus one move if next to the goal hexe) and then exit.

i.e. h(piece to goal hexe) = $\lceil{\frac{number \, of \, move \, action}{2}}\rceil$ as one jump is considered as two move actions. Where #move action = SLD distance
∵# jump action (optionally plus one move if next the goal hexe) is the ideal(lower bound of) length of path for the piece to reach the goal hexe as described above
∴h*(piece) ≥ h(piece to goal hexe) + 1. Note: plus 1 for exit action
∴ h*(state) ≥ $\sum*{piece ∈ player}$h(piece) = $\sum*{piece ∈ player} (\lceil \frac{dist_{SLD} (piece)}{2} \rceil + 1)$

(The heuristic we chose is half of the hex distance(which is the straight line distance in hex coordinate).It is derived using relaxed problem method. In the original rules of game, a piece can move to a new location if it is not occupied by other pieces or blocks, and a piece can jump to a new location if it is not occupied by other pieces or blocks and there is a piece or block in between. In the relaxed problem, a piece can move or jump as it wishes. So, the heuristic we chose can be interpreted as the piece jumps all the way to the destination without thinking the availability of new location and if there is piece/block helping for jumping. This is the shortest path as in real scenario, jump action is not always valid. Therefore, the heuristic is admissible.)

Problem Feature Impact

Search Tree

- branching factor
    - In this project, the board is hexagonal, which has a higher branching factor, compared with a square board. According to the the analysis above, both time and space complexity will be higher.
- depth

Other features of the input which have impact on program

- Input Method:
    - The input (initial information) of our program comes from a json file in the disk. Another way of giving input (initial information) is through command line argument (CLA).
      The time complexity of our program is affected by different input methods. There is one I/O operation for the first method, whereas using CLA, the information goes directly into the memory, enabling CPU to fetch data more quickly.
- time complexity
- space complexity