Brief Introduction

Since this assignment is focusing on the shortest path in a Chexer game, it is not useful to consider this problem as a point-to-point issue, especially for multiple pieces. Therefore, in the A* search, each state of the Chexer game is regarded as a whole problem to solve Eventually, it supposes to gain the shortest path to our goal state by using Dijkstra and A* search algorithms.

Before the implementation, the establishment of single agent program, which is used to figure out the shortest path to the goal state (the smallest number of actions to exit all pieces), needs to consider the following four attributes:

- *Percept* : in the gameboard, each movement is considered by detecting the coordinates of blocks, the current and next coordinates of a piece.
- *Actions* : before the piece can exit, the possible actions for it are moving and jumping. In addition, all actions need to happen in the board and also include exiting action, because the piece has to exit from board hexes.
  - For the moving action, as long as the next hex is in board and is not the block, the piece can move successfully to that nearby hexes.
  - For the jumping action, the piece could jump only over one piece or one block to a empty hex which means there is not any other piece and piece in the linear distance. Additionally, the piece cannot jump out of the board directly to exit.
  - For the exiting action, the past cost for the exiting is also 1 like other movements. When the piece exits, it cannot come back to the board again and the total number of pieces will be minus 1.
- *Environment* : in the board, there are pieces might in different colors and numbers, empty hexes, block hexes and exiting hexes.
- *Performance measure* : since each board is considered as a state, the agent needs to find the least number of changed states to the goal state. In other words, the agent needs to find the least number of actions for all pieces to exit the board.

The main algorithm used is the A* in agent. There are several key steps to implement the A* search:

1. The A* search mainly focus on the formula f(n) = h(n) + g(n), which h(n) is the heuristic distance and g(n) represents the path cost. In this Chexer's state, each action only costs 1 without any other special condition, thus, the calculation of heuristic distances are the only various variables and play an important role on the algorithm's efficiency.
2. The introduction of Dijkstra search can help a lot to support an admissible heuristic evaluation to implement A* search in this Chexer game. It computes the smallest number of actions to the closest exit for each hex, and then, returns a dictionary with all coordinates and corresponding smallest numbers of movements for coordinates.
3. Eventually, the preprocessed dictionary which gets from Dijkstra search is utilized as the referenced h(n) in the final A* search.

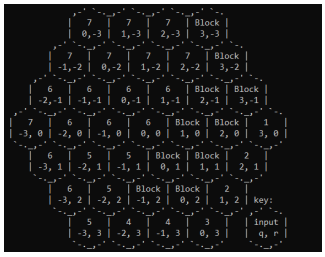
The Implementation of Algorithms

This assignment's target is to find a shortest path, therefore, the using search algorithm must be completed, optimal and as efficient as possible. According to all factors above, A star search is the best choice, because, except iterative deepening search, breath-first search and uniform-cost search, other algorithms which mentioned in subject are not complete or not optimal.

- For the uniform-cost search, it has no idea where the goal state is so that the search space and the time complexity will be extremely large for the Chexer game. To some extent, the iterative deepening search can be modified by uniform search tree, the iterative deepening search has the same drawbacks with uniform search.

- For the breath-first search, the space problem is an unignored issue during the implementation.
- For A star search, if h(n)<=h* (h* is the minimum true cost to the goal state) the optimal path must be found and the efficiency of this search could be best above all algorithms as long as the proper h(n) is established.

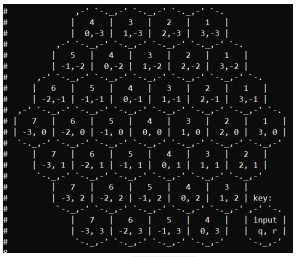
In the agent, the priority queue method is mainly used in the A star search. In the A* search's PQ, it begins with putting the start state in and pop the queue, then get a list of that popped state's next possible states by the 'result' which is a return value from 'successors' function. In Dijkstra search, it returns "cost_so_far" as a dictionary which records minimum cost In order to make sure the minimum cost from that hex, jump action should be firstly considered, after all, jumping can move two hexes but moving is only one. What is talk above is showed on picture1.1 as an instance (all numbers in hexes are the min cost for themselves).



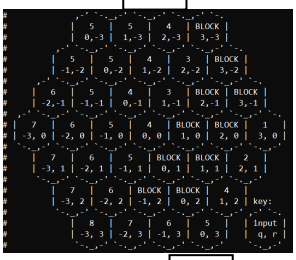
1.1

Admissibility

For the admissibility, when the heuristic evaluation is equal or smaller than the true cost (h(n)<=h*), it is admissible. For the min cost that returns by the D search, it is truly admissible, because the jump action is always prior considered and there is not any other action could be better saving past cost.



2.1



2.2

Further Discussion

Before using the heuristic values from D search, there were also other two attempts :

1. Linear distance between one hex to exit is admissible, however, the h(n) cannot ensure the optimal path. For the linear distance, take picture 2.1 as an example, the h(n) along the curve side is relatively smaller than the optimal path's h(n) so that the piece will firstly consider the path along the curve in picture 2.2 instead of the optimal one
2. For the Manhattan distance of hex, it can be considered as moving first so that the h(n), sometimes, would be larger than the true cost and cannot get the optimal path (picture 2.2).

Memory space and time complexity

For the last part of this report, it is going to demonstrate other factors that affect the memory space and time complexity for the Chexer game itself. There are two possibilities to impact the result efficiency, which are the number of pieces and the number of blocks.

1. The increasing number of using pieces will raise a huge number of states to access in the priority queue so that the more space and the more time complexity would be used. The increasing number of pieces, in general, may cause:
   a) the increasing number of branch factors, because the large number of pieces means the more states there would be so that there should be more possible actions.
   b) no absolute effect on the depth of search tree, because the cooperation (jumping) among pieces may also save path cost.

2. The more blocks there are, the less memory space and time complexity will be, because the block means the hex cannot move or even jump. Therefore, there should be fewer possible successors to consider about. However, the number of blocks will affect the branching factor and the depth of the search tree in general. The increasing number of blocks, in general, may cause:
   a) the increasing number of possible actions, because of the indirect path to exits, so that the depth of search will be larger.
   b) the decreasing number of branch factors, because there might be fewer next possible hexes to expend, especially for two consecutive blocks.