Project Part A Report

SEARCH STRATEGY

In our implementation, we use breadth-first search strategy to write a solution to play "single player" version of Infexion. The initial state of the game is a given configuration of the grid with at least one red and blue hex; whereas the goal is to get rid of all the blue hexes on the board. Initially, we begin generating a "state" for every red hex to spread in all valid 6 directions and save those states in a list called potential moves. The list of potential moves will be further pruned down by only keeping the desirable states with **heuristic checking**. We proceed to loop through the pruned list again, repeating the whole loop. When a solution is found, we exit the loop. Below is the data structure used in our implementation:

State = {gridLayout, previousMoves, heuristicResult, gameEnded}

gridLayout: A dictionary consisting of two other dictionaries, each recording locations of blue and red hexes.
previousMoves: A list of previous spread actions that led to the current state from the starting state.
heuristicResult: The current state's heuristic.
gameEnded: A boolean value indicating if the current state is terminal.

TIME / SPACE COMPLEXITY ANALYSIS

The time complexity of a BFS(breadth-first search) is O(***b^d)***, where *b* is the branching factor and *d* is the depth of the least-cost solution. The branching factor is the number of red hexes on the grid multiplied by the number of valid spread directions and an increase in depth by 1 represents an additional move.

Space complexity is also O(***b^d)*** due to the number of nodes we keep at the maximum depth level of the search.

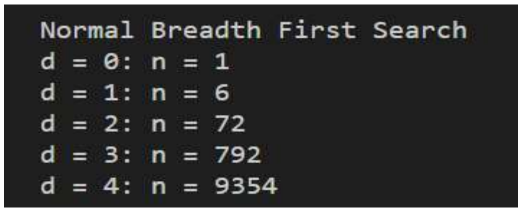
BFS is complete as it would go through every single node of each depth before proceeding to the next depth. It is also optimal as the step cost of Infexion is 1, essentially unweighted. It traverses the nodes in increasing order based on its distance from the root node. Thus, if a solution is found, it would be the shortest path, optimal.

HEURISTIC ANALYSIS

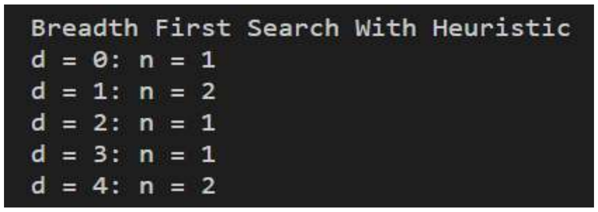
We have incorporated a heuristic in our approach. It consists of two components, the number of blue hexes converted from a previous state to current state and the straight line distance between a red hex and a blue hex on a grid. Only the states in the potential moves with the most number of blue hexes converted, then the shortest straight line distance would be kept. The heuristic managed to cut down a significant amount of less desirable states in potential moves. This reduces the time and cost to search through the number of states generated, thus improving efficiency. Our heuristic is **admissible** as it uses the shortest straight line distance between two hexes which would always be shorter than the actual step distance between two hexes.

EXAMPLE

For the number of nodes expanded (n) at a certain depth (d):



```
Normal Breadth First Search
d = 0: n = 1
d = 1: n = 6
d = 2: n = 72
d = 3: n = 792
d = 4: n = 9354
```



```
Breadth First Search With Heuristic
d = 0: n = 1
d = 1: n = 2
d = 2: n = 1
d = 3: n = 1
d = 4: n = 2
```

With the heuristic applied to our breadth-first search on the sample test case as shown above, it drastically reduced the number of nodes expanded upon in each depth of the search.

IMPACT OF SPAWN ACTIONS

If SPAWN actions were allowed in single player Infexion, the complexity of the search problem would increase significantly. Previously, the search strategy would only need to account for the existing hexes on the grid. The search strategy would need to also factor in every single neutral hex on the grid as it could be a potential place for a hex to spawn, which gets added into the existing hexes.

If we were to just consider SPAWN actions on a given starting board, instead of just spawning a red hex on every possible neutral hex on the board which would greatly increase the number of states for each depth, we could spawn on a valid neutral hex right beside a blue hex. With this, an upper bound would be placed on the number of states generated by our search strategy. And the number of moves required to win the game would be at most double the number of blue hexes.

To accommodate the SPAWN action into our current solution, we would “spawn” a potential red hex on every possible neutral hex surrounding the blue hexes, which are subsequently added into our list of potential moves. Then, we would loop through the list of potential moves to check if the existing red hexes could convert any blue hexes in a SPREAD action. If not, we would SPAWN a red hex beside a blue hex with a high power which is less than 6 as we do not want to get rid of the hex immediately.