# AI report -- Team: Future_Yoda

## Search problem point of view:

In this board game, our pieces are trying to eliminate all opposed pieces. The current board can be our current state and our goal state is board with only our pieces left. To achieve the goal state(all lower eliminated), there are several actions can take, which is the product of all the possible actions(include swings and slides) of our pieces. The result of action will cause change in state from the current board. The path cost from a board state to next board state will be sum of path cost of all our pieces(move one step accounts for increase in one pathcost). Goal test will be testing for whether there are any opposed pieces left, if not, we reach our goal.

## Search algorithm

The algorithm we choose is A* algorithm, which is an informed search algorithm combines the benefits of both Dijkstra and best first search. The algorithm will evaluate all the possible actions based on heuristic score + path cost. Heuristic score is the max length of path generated by Breadth first search of each upper piece to its target at current state, path cost is the total depth of current state. The algorithm is complete given we have finite state to explores, so we will always find one if exist. Also it is optimal given the heuristic score will only underestimate the true path not overestimate it. We have developed two heuristic function, one is the Euclidean distance, which will guarantee to be optimal as it is always shorter than true path, but it may be inefficient as more nodes need to be expanded to find the optimal path. The other is the length of shortest path of all pieces generated from current state to goal state, it is not complete as sometimes some pieces are temporarily blocked, and hence we can’t get it’s heuristic value. But the heuristic score will be closer to true path cost and reduce the number of nodes we need to generate. We also add an weight on path cost, since for a few complex board, algorithm described above cannot solve in time limit, the weight on path cost make a star behave more like best first search, but also keep the solution quality, and increase the efficiency of the algorithm.

**Feature of initial state**

If the number of our pieces increase, the time required to find the solution increase. This can be clearly shown in generate next actions function, each piece has a possible of 6 slide movement, if we count in the swing action, the max possible movement is 12(a piece surrounded by upper pieces), possible combination of all actions for 2 pieces will be 6x6 = 36 for each stage. For 3 pieces will be 6x6x6 = 216 combinations. So increase upper pieces will result in exponential increase in states to be explored at each stage, due to higher branching factor. Besides, increase in distance between our piece to opponent piece will result in increase in depth. In other word, we need more steps to reach goal state and thus exploring more nodes in between. This can be clearly seen in the time complexity. For best first search, we only explore one path and thus evaluate b nodes for b depth. A* stays in between which need to explore approximately 1+ 6^n + a(6^n) states for each depth given we have n pieces. So increase in depth and initial pieces will result in increase in time complexity. Also as nodes are stored in priority queue, it will also result in increase in space complexity.