Report

Min Se Chang

For the third project of AI nanodegree, I implemented an adversarial game playing agent which plays Knights Isolation game. I used minimax search algorithm with alpha-beta pruning and iterative deepening from depths 2 to 5. The agent was tested by playing against random, greedy, and minimax agents and consistently outperformed these sample agents. To get baseline, I played against minimax agent (which is the hardest agent to beat among the three) with “fair_matches” flag, for 100 times.

In order to outperform a vanilla Minimax agent, I implemented a custom heuristic score function. It essentially penalizes moves that are going towards edges of the board, by returning penalty of “2” for the farthest edges (any cells with x or y value of 0) and “1” for cells with x or y value of 1. These penalty values are increased when the agent is in later phase of the game; when ply-count is greater than 70, penalty value is multiplied by 3, and when ply-count is between 40 and 70, it is multiplied by 2.

As can be seen from Table 1, alpha-beta search agent with custom heuristic consistently outperforms vanilla Minimax agent. Although not included in the table, the difference in winning probability is significantly higher for greedy and random agents. This means the custom heuristic is doing a reasonably good job in prioritizing moves that are farther from the far edges and this strategy is consistently contributing to higher winning rate.

Table 1: Match Result (with “fair_matches” flag enabled)

	Iterative Deepening (depth 2~5), 100 runs	**Fixed Depth of 2, 20 runs**	**Fixed Depth of 5, 20 runs**
Heuristic with Edge Penalty	71.2%	23.8%	72.5%
Simple Heuristic (difference in liberties)	63.2%	27.5%	62.5%

I had a choice of implementing the heuristic function in two different ways. If I prioritize accuracy over speed, I could add more complicated logic to the penalty calculation. For

example, I could penalize moves to the 4 corners very severely, extend the penalized edges to 4 rightmost/leftmost cells or add more granularity to the scaling factors for multiplying penalty value by adding more case statements. On the other hand, it's beneficial to keep the heuristic calculation function as simple as possible since complicated logic in heuristic calculation function leads to longer execution time, which means actions with longer depth will never have chance to run (given the search will be terminated after a certain amount of time). I decided to go with the latter approach since I thought it's more advantageous to leverage power of iterative deepening than being limited to shallow search depths.

For this reason, search speed matters more than accuracy to the performance of my heuristic. I believe this was the right choice to make since heuristic function will be called for every single move from the beginning of the game and it's very hard to get "accurate" estimate at early stage of the game anyway. This was confirmed by running the algorithm with two extreme fixed search depths; when search depth was fixed to 2, the search algorithm performed very poorly and resulted in 23.8% of winning rate (as opposed to 27.5% without the edge penalty heuristic). On the other hand, when search depth was fixed to 5, search with edge penalty heuristic resulted in 75% and search with simple heuristic ("my_liberties - opponent_liberties") resulted in 65%. This clearly shows that higher search depth consistently leads to bigger difference in terms of winning rate.