Heuristic analysis

Yun Ling

This article is my review for AIND-Isolation project. Below the content, there are three evaluation functions I developed for my game agent.

I chose three of the best function from my work. The three custom functions are similar with each other. Each of them are based on a previous thinking before.

AB_custom

The first function is based on the code given in the sample_players.py. It is the combination of center_score and improved_score. The center_score evaluate from how long the player away from the center. The nearer player in center, the more chances can it move. Then, the imporved_score is based on how many opening moves the player have against opponent. I think both them are good evaluation function. So, I combine them together, that is the AB_custom.

AB_custom2

The second evaluation function is the reform for AB_custom. In my opinion, if it is good for the player keeping himself around center, keeping the opponent away from the center is always a good decision. So, I added the feature to encourage the player not only stay at center but also tend to force the opponent away from the center. But in testing, it performs a little bit worse than the AB_custom.

AB_custom3

In searching function, the time is always limited. As quicker algorithm can spend less time. And then, the function will iterate more times and may find a better solution. Reducing the time usage per calculation is what the third evaluation function to do. Following this viewpoint, I tried to simplify the algorithm in third evaluation to let the CPU do fewer calculations. I remove the square and root extracting in distance calculation functions in order to save more time. To make this work, I replace Euclid distance with Manhattan distance

Match #	Opponent	AB_Improved		AB_Custom		AB_Custom_2		AB_Custom_3	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	35	5	40	0	38	2	33	7
2	MM_Open	30	10	32	8	25	15	22	18
3	MM_Center	30	10	37	3	29	11	31	9
4	MM_Improved	23	17	29	11	23	17	24	16
5	AB_Open	19	21	25	15	21	19	22	18
6	AB_Center	23	17	19	21	24	16	24	16
7	AB_Improved	19	21	17	23	19	21	14	26
	Win Rate:	63.9%		71.1%		63.9%		60.7%	

Figure 1

The graph (Figure 1) shows about each opponent verses different game agents. We can see that all of the agents can defeat the random-choice agent very well. Furthermore, the game agents using Alpha-Beta pruning get a better winning rate against the player just using the Minimax tree. But it appeared fierce competitions on alpha-beta pruning functions. So I make another 100 rounds for each of my evaluation functions against AB_improved function (Figure 2).

Match #	Opponent	AB_Custom	AB_Custom_2	AB_Custom_3
		Won | Lost	Won | Lost	Won | Lost
1	AB_Improved	53 | 47	52 | 48	49 | 51
	Win Rate:	53.0%	52.0%	49.0%

Figure 2

As we can see, it is still hard to guarantee which one is better because my custom agent just has a slightly winning rate. But we can say all of them should be a good algorithm in this game.

To compete with other agents, if player evaluate the winning rate just according to the result upon. AB_custom agent will be the best one. But in realistic competition, each side will not just use min/max strategy, so I decide to use AB_custom2 due to its better performance against all alpha-beta pruning strategy. If the player concentrates on the implement difficulty, AB_custom3 will be a good choice because of the single structure. It can be easily to compute using some optimize hardware structures. AB_custom2 is the best potential strategy to improve, because it considers more than other two. In further research, I will choose it to be my base thinking to develop.

To draw a conclusion, it is still hard to defeat just a simple evaluation function in real applications. We need an algorithm, to avoid pruning good potential result. In further research, choose cross-validation to weight each part of algorithm will be a better choice.