Heuristic Analysis **Pedro Felipe Gardeazábal Rodríguez**
Project: Build a Game-Playing Agent
Course: AIND - Udacity Date: 01/12/2018

Isolation Game Agent

Methodology

Three heuristic functions were proposed for the Isolation game agent moving as a chess knight playing against an agent with the same moving characteristics. Given that the results of several launches of tournament.py with a value of NUM_MATCHES set to 5 gave a lot of variations, I decided to ran tournament.py with NUM_MATCHES set to 100 to get better statistics for the final analysis while keeping TIME_LIMIT at 150 milliseconds.

Heuristic functions

- Function custom_score:
  I assumed that my agent wants to stay close to its opponent in order to reduce the number of possible positions for its next move. So, this function calculates the inverse of the mean distance between my agent's possible positions in the next move to the actual opponent's position $+\sqrt{5}$, where $\sqrt{5}$ is the distance between the opponent's actual position and its next position. The inverse was chosen because from the tests it gave better results than taking the negative of the mean value.

- Function custom_score_2:
  I considered that my agent wants to have a different evaluation function dependent on the state of the game. At the beginning of the game my agent tries to be as close as possible to the center of the board to increase the number of possible positions for the next move. So, initially this function returns the inverse of the mean distance between the agent's next possible positions and the center of the board plus 1 square. After several turns (estimated to be when the number of occupied squares is equal to the board's height times 2), this function becomes obsolete because the center of the board fills up, then for the rest of the game my agent calculates the difference between the number of its possible valid positions for the next move multiplied by a weight set to 4 and the number of its opponent's possible valid positions for the next move. The weight of 4 gave better results than setting it to 1 or 2.

- Function custom_score 3:
  This function returns the difference between the agent's number of possible positions multiplied by a weight of 4 and the number of possible positions for its opponent. The weight of 4 gave better results than setting it to 1 or 2.

Results

The results of tournament.py are shown in figure 1.

- From the proposed heuristic functions, functions custom_score, custom_score_2 and custom_score_3, identified in the figure by AB_custom, AB_Custom_2 and AB_Custom_3 respectively, performed very well against the Mini-Max players. AB_custom got a winning rate of 77.7% AB_Custom_2 of 81.3% and AB_Custom_3 of 78.0%. Furthermore, all the three functions resulted in a win against the Random player in more than 90% of the matches.
- Considering the overall winning rate, AB_Custom_2 and AB_Custom_3 obtained the best results with a rate of 71.8% and 69.6% respectively (AB_Custom getting just 66.4%). This rate is similar to the overall winning rate of the reference function AB_Improved.
- Considering just the matches against AB opponents, AB_Custom_2 and AB_Custom_3 performed almost identical with a winning rate of 55.0% and 53.2% respectively and outperformed AB_Custom that got a rate of 46.7%. We observe that AB_Custom_2 slightly outperformed AB_Custom_3 and if we take a closer look to the matches against AB_Improved (assuming that an opponent will always use its improved function), AB_Custom_2 performed definitively better with a winning rate of 55.0% against 49.5% for AB_Custom_3.

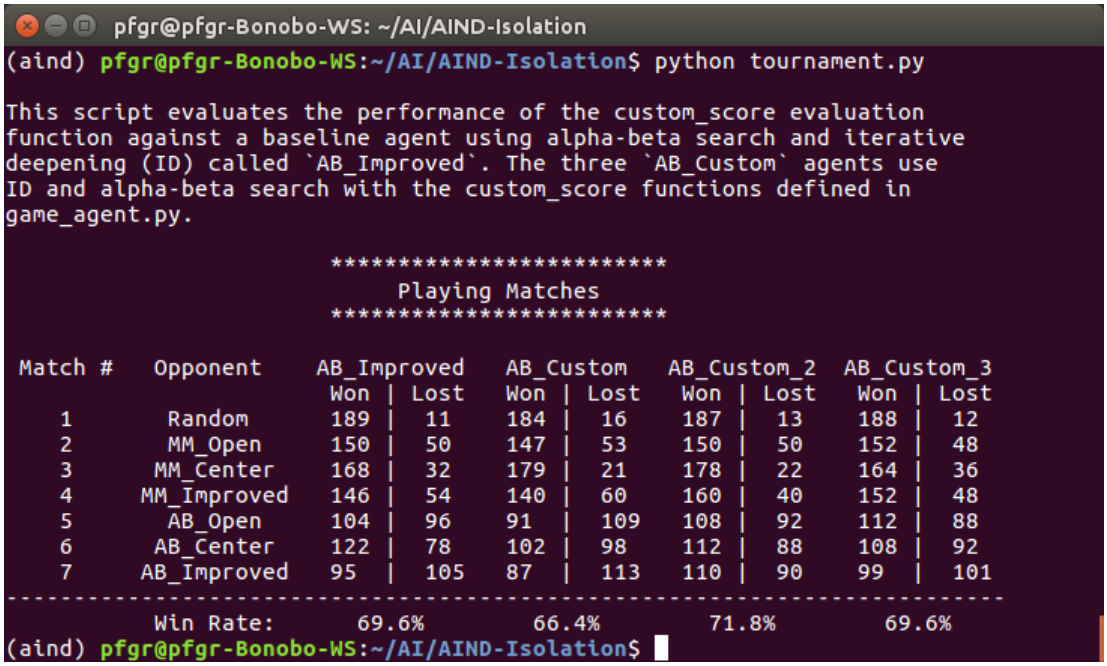

Figure 1: Heuristic analysis results with NUM_MATCHES = 100 and TIME_LIMIT = 150ms

Discussion

Even if the three functions custom_score, custom_score_2 and custom_score_3 are very easy to implement, custom_score_3 stands out for its simplicity and function custom_score performs a more complex calculation than custom_score_3 but gives a higher score to moves landing near to the opponent, then it has the drawback of favoring moves reducing the number of possible positions for the next turn.

Function custom_score_2 combines the simplicity and complexity of custom_score and custom_score_3, however in contrast to custom_score, from the beginning of a game and up to around the middle of a match custom_score_2 favors moves leading to more possible positions for the next turn given that it tries to keep the player close to the center of the board. By the end of a game custom_score_2 turns to the simplicity and reliability of end of match present in custom_score_3.

Of all the three functions custom_score_2 has not only the best overall winning rate but also the best winning rate against AB_Improved winning 55.0% of the times.

Conclusion

For all the functions the proposed 150ms time limit search was not a problem. Of the three proposed functions, custom_score_2 was the best performing one while having at the same time the advantage of combining the simplicity and reliability at end of match of the other two proposed functions depending on the state of the game. Considering this three aspects (good performance on the given time, winning rate and reliability), I recommend the use of custom_score_2 in an isolation match where the pieces moves as chess knights.