Heuristic report

Analysis of coding a Game-Playing agent using Heuristic Methods
Student: Martijn de Boer
Program: Udacity Artificial Intelligence Nano Degree course
Project: Project two: Building a alpha go agent!
Language: Python + GIT

Heuristic research:

I downloaded the project's files from github and imported the needed python packages in my Anaconda environment and created the search functions, I included Iterative Deepening in my code. When the coding was done I tweaked the model's parameters to build the best possible agent. Then I runned 13 test runs with different settings to define the ideal settings for the agent.

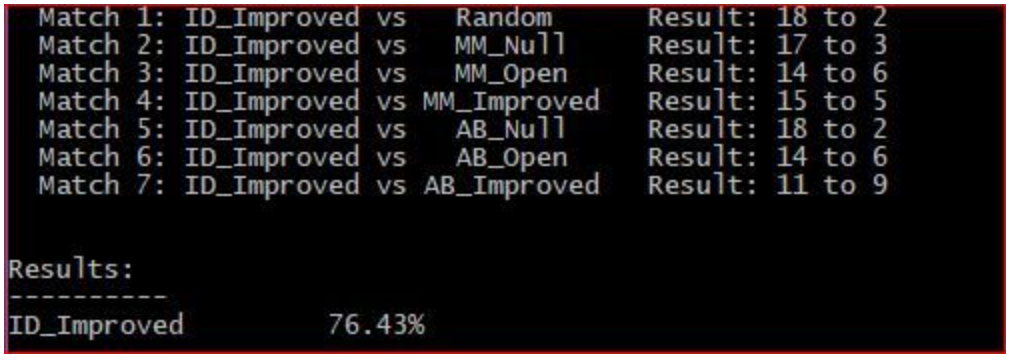
Steps taken:

Depth: I noticed that Iterative Deepening with time out takes way more time, I also tried to implement multiple different depths for the model: 5, 10 and running until the models timeout. in Iterative deepening with alpha-beta pruning to see if a custom function consistently outperforms the "Improved" scoring function

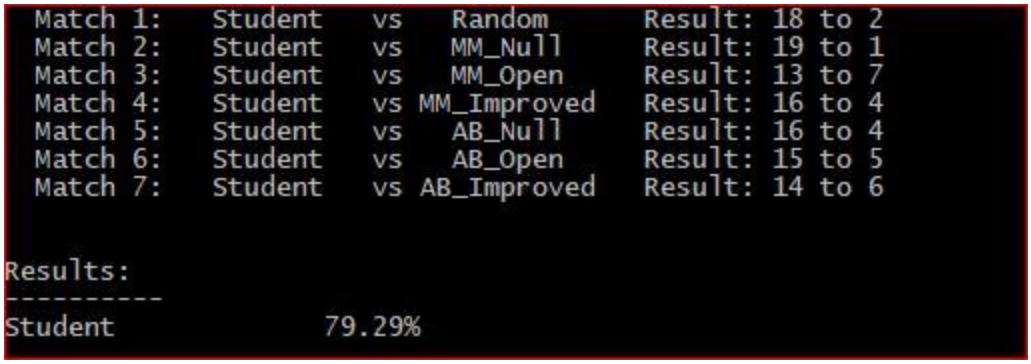
Random: I noticed that the ID_Improved agent provides different results due to randomness of initial moves, I tried to use a constant seed (21) to see if I could try multiple test cases to compare the results with each other.

Eval Function: I implemented a few different evaluation function mostly based on own_moves left and opponent player's moves left. I also experimented with implementing more weights for both own_player and opponent_player to see if it can make a difference and I also implemented 'log' values to normalize the scores in my model.

ID_Improved matches:



Student matches:



Visualization of the test runs plus the performance of the agents:

Run	Used technique	Depth	Function	Evaluation function	Id Improved score	Student score
1	Iterative deepening	Untill time out	Random	float(np.log(1+own_moves) - np.log(1+(2*opp_moves)))	72,14%	70,71%
2	Iterative deepening	Untill time out	Random	float(np.log(1+(2*own_moves) - np.log(1+(opp_moves)))	75,71%	71,43%
3	Iterative deepening	Untill till depth = 5	Random	float(own_moves / opp_moves)	68,57%	67.86%
4	Iterative deepening	Untill till depth = 5	Random	float(own_moves - opp_moves)	71.40%	71.43%
5	Iterative deepening	Untill till depth = 5	Random	float(own_moves)	71,4%	61.43%
6	Iterative deepening	Untill till depth = 5	Random	float(own_moves - (2*opp_moves))	73.57%	69.29%
7	Iterative deepening	Untill till depth = 5	Random	float(blank_space)	71.43%	48.57%
8	Iterative deepening	Untill till depth = 5	Random	float((blank_space/own_moves) - (blank_space/opp_moves))	69.29%	46.43%
9	Iterative deepening	Untill till depth = 5	Random	float(np.log(1+own_moves) - np.log(1+opp_moves))	68,57%	77.60%
10	Iterative deepening	Untill till depth = 10	Random	float(np.log(1+own_moves) - np.log(1+opp_moves))	75.00%	72.80%
11	Iterative deepening	Untill till depth = 5	Random	float(np.log(1+own_moves)-np.log(1+(2*opp_moves)))	66.43%	73.50%
12	Iterative deepening	Untill till depth = 5	21	float(np.log(1+own_moves) - np.log(1+(opp_moves)))	66.43%	65.71%
13	Iterative deepening	Untill till depth = 5	Random	float(np.log(1+own_moves) - np.log(1+opp_moves))	76.43%	79.29%

Recommendations based on the data:

Based on the results of all 13 test runs I decided to go for the “Iterative deepening till depth = 5 and float(np.log(1+own_moves) - np.log(1+opp_moves))” settings, thus this settings reached the best results in the matches for both the ID_improved (76,43%) and the Student (79,29%) matches