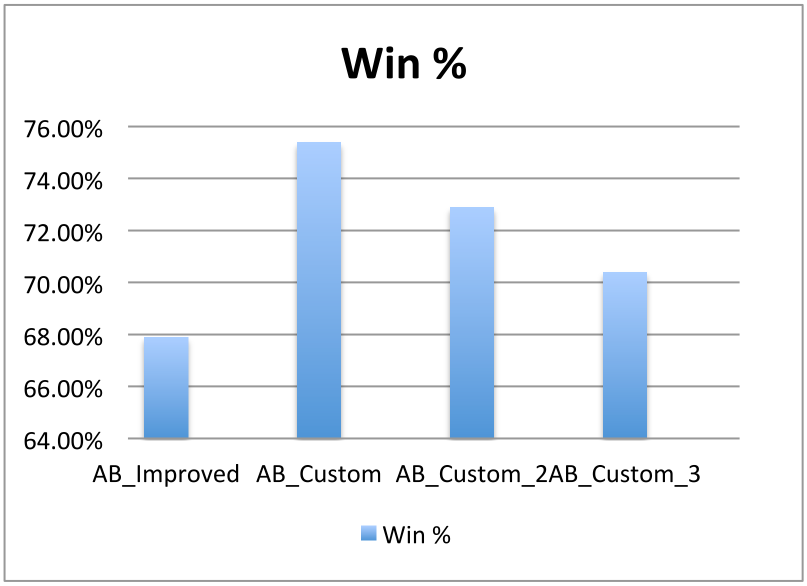
Adam Main

Udacity AIND - Project #2 - Build a Game Playing Agent

Opponent	AB_Improved		AB_Custom		AB_Custom_2		AB_Custom_3	
	Won	Lost	Won	Lost	Won	Lost	Won	Lost
Random	36	4	39	1	38	2	39	1
MM_Open	32	8	34	6	30	10	26	14
MM_Center	36	4	37	3	37	3	35	5
MM_Improved	30	10	28	12	26	14	29	11
AB_Open	18	22	25	15	23	17	24	16
AB_Center	19	21	23	17	27	13	24	16
AB_Improved	19	21	25	15	23	17	20	20
Win %	67.90%		75.40%		72.90%		70.40%	
	(s) = own_mv - opp_m		f(s) = now+future_moves / opp_moves**w		f(s) = own_mv/(1/corners)		f(s) = own_mv / opp_mv*2	



Recommendation: AB_Custom

1) Consistently won the most games compared to other heuristics.
2) Best cost (processing time) to game state information (look-ahead) ratio.
3) Adding look-ahead adds depth of search.

AB_Custom ** Chosen Heuristic

* NOTE: Created "get_moves(loc)" function in game_agent.py for the look-ahead functionality. This function was intended for use only within the

Board class, but creating a local function for access within game_agent.py resulted in a 5% increase in wins on average. The alternative would be to call 2 functions to get the active players step-ahead moves (forecast_moves(move), .get_legal_moves(game)) before a secondary call would be made to "__get_moves(move)". To get the opponents looh-ahead moves, 3 function calls would need to be made (forecast_move(), get_opponent(), get_legal_moves()) before an secondary call to "__get_moves" is made. Evidently, the processing time to do these extra function calls is costly.

Returned score is the ratio between the number of available legal moves for the computer player plus a look-ahead count to the computer players available future moves to the weighted number of available legal moves for the opponenent.

The weighted ratio encourages the computer player to both maximize the breadth of its available moves while also minimizing the opponenets available moves.

The look-ahead step rewards moves that lead to a larger breadth of available moves even if the current number of available moves is low.

The highest returnable score available would be achieved by the computer player having a high number of available legal moves for its current turn, a high number of available moves for at least one of it's available moves for its next turn, as well as the opponent having a low number of available moves for their current turn. A low score would be achieved if the reverse was true.

AB_Custom_2

Score returned is the determined by the total number of legal moves available negatively weighted if the player by player's position is in the corner of the board.

The highest available score other than winning would be achieved by choosing a move that maximizes the players available legal moves and is not in the corner of the board. A low score would be achieved by choosing a move that minimizes the players available legal moves and is in the corner position on the board.

Less sophisticated than custom_score resulted in lower percentage of wins. Won a similar amount of games to custom_score_3 in most tests.

AB_Custom_3

Compares computer player's available legal moves to a opponent player's weighted available legal moves. The return score is the ratio of the number of available legal moves for the computer player to the weighted number of available legal moves the opponent.

The weighted ratio encourages the computer player to both maximize the breadth of its available moves while minimizing the opponenets available moves.

The highest score available to return would be achieved by the computer player choosing a move that maximizes its current and future available legal moves while minimizing the opponent available legal moves for their current turn. A low score would be achieved if the reverse was true.

The weighted ratio performed better than AB_Improved in some tests, but not all. Contains less board state information than custom_score, and the faster processing time did not overcome this information deficit.