In this review I will describe the algorithms for the game search and compare the different custom score functions from the result of tournament.

1. Iterative deepening search for Alpha-Beta pruning algorithm

For the max layer -
Test if search depth is zero: if yes, return score function from the point view of current player.
Pre-assign the current value so that any thing is greater than it.
Loop through game's available subsequent moves from current player:
a) apply the move and create the next min layer with (search depth -1),
update current maximum value if it is less than next value;
b) pruning: if current maximum is greater than or equal to beta – the upper bound of search on max layer, then return current maximum.
c) update alpha if it is less than current maximum.
Return current maximum.

For the min layer -
Test if search depth is zero: if yes, return score function from the point view of current player.
Pre-assign the current value so that any thing is less than it.
Loop through game's available subsequent moves from current player:
a) apply the move and create the next max layer with (search depth -1),
update current minimum value if it is greater than next value;
b) pruning: if current minimum is less than or equal to alpha – the lower bound of search on min layer, then return current minimum.
c) update beta if it is greater than current minimum.
Return current minimum.

2. Custom score function

1) (number of legal moves for current player – number of legal moves for its opponent)* number of filled spaces*0.5. It favors more moves for current player than its opponent. In addition, it favors more search into deeper depth.

2) (number of legal moves for current player – number of legal moves for its opponent)* number of filled spaces. Similar to 1), but has a higher weight of favoring search depth.

3) Using (number of legal moves for current player – number of legal moves for its opponent) at first, then in later stage of game – 80% of board is filled, use (number of legal moves for current player). It is a slightly more conservative version of AB_Improved.

Match #	Opponent	AB_Improved	AB_Custom	AB_Custom_2	AB_Custom_3
		Won | Lost	Won | Lost	Won | Lost	Won | Lost
1	Random	8 | 2	8 | 2	5 | 5	6 | 4
2	MM_Open	3 | 7	5 | 5	6 | 4	4 | 6
3	MM_Center	6 | 4	7 | 3	7 | 3	6 | 4
4	MM_Improved	6 | 4	5 | 5	3 | 7	6 | 4
5	AB_Open	4 | 6	6 | 4	8 | 2	5 | 5
6	AB_Center	8 | 2	7 | 3	5 | 5	6 | 4
7	AB_Improved	4 | 6	5 | 5	5 | 5	4 | 6
	Win Rate:	55.7%	61.4%	55.7%	52.9%

AB_Custum has a overall win rate of 61.4%. Its performance against MM or AB is quite balanced and shows a clear advantage. AB_Custom_2 has a win rate of 55.7%, not as good as Custum. It shows the weight of 1 instead of 0.5 in front of number of filled space is too greedy. AB_Custom_3 has a win rate of 52.9%, not as good as AB_Improved. It shows even in the later stage of game, it is still more favorable to

use heuristic to suppress opponent's choice, in addition to increasing its own legal moves.

3. Recommendation

Overall, it is recommended to use AB_Custom. It has the highest win rate against all agents. The reason is that it not only favors large difference between legal moves of player and its opponent, it also encourage explore searching into deeper trees. In term of complexity, it counts moves for player and opponent, as well as the filled spaces on the board. It is still order (size of board positions).