Data:

```
 Match #    Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                         Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1        Random      46  |   4    49  |   1    46  |   4    44  |   6
    2        MM_Open     40  |  10    33  |  17    38  |  12    35  |  15
    3       MM_Center    43  |   7    42  |   8    47  |   3    48  |   2
    4      MM_Improved   38  |  12    36  |  14    39  |  11    39  |  11
    5        AB_Open     24  |  26    29  |  21    22  |  28    25  |  25
    6       AB_Center    26  |  24    34  |  16    31  |  19    25  |  25
    7      AB_Improved   22  |  28    23  |  27    27  |  23    19  |  31
--------------------------------------------------------------------------
            Win Rate:      68.3%        70.3%        71.4%        67.1%
```

Note: Modified the tournament script to run 50 games instead of 10 to get more reliable stats.

AB_Custom:

The custom score heuristic gets the proportion of the player's moves vs that of the opponent (own legal moves/total legal moves). Wanted to see how the the proportion of player vs opponent legal compare to stand heuristic where the number of legal moves by players minus the number of legal moves by opponent.

Figure this algorithm captures the agent’s knowledge that a move that heavily restricts the opponent while keeping its own options open is a good rule of thumb. In addition the values are all scaled between zero and one where high values near one are proportionally better than lower values.

AB_Custom heuristic is a very simple function that calls game.get_legal_moves() twice to get the own player’s moves vs that of the opponent without any additional loops or datastructures therefore the algorithm runs in constant time of O(1).

The heuristic does very well against the random opponent with 49 wins against random's 1 (49/1) which seems to indicate it is doing much better than random guessing as a standard benchmark.

AB_Custom still performs better than the three MiniMax algorithms with scores of 33/17, 42/8, 36/14 respectively which indicate that the algorithm is effectively pruning large sections of the game tree against its opponents.

When compared to the three AlphaBeta algorithms the results were mixed. AB_Custom outperformed AB_Open & AB_Center with 29/21 & 34/16 respectively, and slightly underperformed against AB_Improved with 23/27.

Total results for AB_Custom against all 7 players is 70.3% with 246 wins and 104 losses.

AB_Custom_2:

Gets own players legal moves subtracted from twice the opponent's legal moves. Tested several implementations as suggestion on the forums that would incentive looking for more conservative legal moves. Because the algorithm looks at legal moves from both players the values are not scaled like AB_Custom.

AB_Custom_2 also runs in constant O(1) time since also calls game.get_legal_moves() twice to get number of legal moves from each player and simply subtracts opponents moves twice from own moves, no additional computation required.

Just as our previous heuristic, AB_Custom_2 performs well against random with a score of 46/4. Next we compare the heuristic function with our MiniMax algorithms with scores of 38/12, 47/3, and 39/11 respectively which show again that the algorithm is searching deeper than the simpler MiniMax functions (expected since our heuristics all utilize the same AlphaBeta pruning algorithm).

Finally we test our heuristic against several AlphaBeta algorithms with mixed results of 22/28, 31/19, 27/23 scores respectively.

Total results for AB_Custom_2 against all 7 players is 71.4% with 250 wins and 103 losses.

AB_Custom_3:

Get the proportion of own moves vs that of the opponent and combines with the location. Wanted to develop a heuristic that takes into account the number of player and opponent legal moves as well as their location from the center. In this way we can incentive moves that favor the maximizing the players moves that are near the center at the opponents expense.

While AB_Custom_3 is slightly more complex (twice the code length) than the previous two heuristic it also runs in constant O(1) since it also does not require any additional data structures, loops, etc.

As with our other heuristics AB_Custom_3 did well against random with 44 wins and 6 losses.

AB_Custom_3 is than compared to and consistently beat our three MiniMax opponents with scores of 35/15, 48/2, 39/11 respectively.

Finally we test our heuristic against our three AlphaBeta opponents. Unfortunately our results were mixed with results of 25/25, 25/25, and 19/31.

Total results for AB_Custom_3 against all 7 players is 67.1% with 235 wins and 115 losses.

Summary:

Based on our preliminary data, it seems to indicate that our AB_Custom_2 as having the best overall performance of 71.4%. Outperforming most of its opponents except for AB_Open and performing favorably to the baseline AB_Improved. In terms of algorithm complexity and resource constraints neither seemed to have any real advantage over the others since they all run in constant O(1) time and are simple heuristics to execute.

In regards to which agents here consistently beating all the others, both AB_Custom and AB_Custom_2 were tied both beating six out of the seven agents consistently. However I believe more important is how the heuristic performs against all three baseline AlphaBeta algorithms.

Since MiniMax is less efficient than AlphaBeta algorithm, this would suggests the AlphaBeta agents should be a better gauge how the custom agent is at consistently evaluating the state of the game rather than the efficiency of the search algorithm. Upon closer inspection AB_Custom outperforms AB_Custom_2 with 86 wins/64 losses of 57.33% compared to 80 wins/70 losses of 53.33% for AB_Custom_2 (if we just compare the AlphaBeta agents stats instead of the grand total). Also note, AB_Custom has a total win rate of 70.3% (second only to AB_Custom_2) and the best win rates for AB agents; therefor my preliminary choice would be AB_Custom over AB_Custom_2 (however would suggest running much longer set of games to get more statistically reliable data).