Heuristic Analysis

Tournament Results:

```
                         *************************
                              Playing Matches
                         *************************

 Match #    Opponent     AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                          Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1        Random        9  |   1     9  |   1    10  |   0    10  |   0
    2        MM_Open       7  |   3    10  |   0     7  |   3     8  |   2
    3       MM_Center      9  |   1     9  |   1     8  |   2    10  |   0
    4      MM_Improved     8  |   2     8  |   2     7  |   3     8  |   2
    5        AB_Open       5  |   5     6  |   4     4  |   6     6  |   4
    6       AB_Center      5  |   5     6  |   4     5  |   5     4  |   6
    7      AB_Improved     5  |   5     4  |   6     4  |   6     5  |   5
--------------------------------------------------------------------------
            Win Rate:      68.6%        74.3%        64.3%        72.9%
```

AB_Custom:

This heuristic used was very similar to the improved_score heuristic, which returns the difference between the # of moves available for the player and the # of moves available for the opponent. The difference with this heuristic is that it also looks at the child nodes of the current game state for both the player and opponent, and factors in the # of moves available in each of those states as well.

This heuristic performed significantly better than AB_Improved, which I think tells us that the results were worth the extra time it took to process the child nodes. This is not extremely surprising, as the logic for each heuristic is pretty much the same – this one just adds more depth to the utility. What *is* surprising is that this heuristic lost 4-6 to regular AB_Improved in their matches. What would be helpful in determining why this is would be to see the depth at which the search timed out for each move.

AB_Custom_2:

This heuristic was pretty experimental. I divided the 7x7 board to several overlapping 3x3 grids that I called “units”, and recorded the number of units the player occupied. I then got each blank space within each unit the player was occupying, and multiplied the number of units occupied with the number of blank spaces within each unit, removing duplicates since some of the units overlapped. My reasoning behind this was that the player could technically have 8 moves within a 3x3 unit, excluding the middle space:

1	4	7
6		2
3	8	5

But, if there any of the spaces occupied, that number is reduced, depending on the location of the occupied space. For example, if #7 is occupied, that's still 6 moves, but if #3 is occupied, we're down to 2 moves within that unit.

I didn't really know what to expect, but it performed slightly worse than AB_Improved. Interestingly, I ran another tournament, with much better results:

Tournament Results (2):

```
                                *************************
                                     Playing Matches
                                *************************

 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random      10  |   0    10  |   0     9  |   1     9  |   1
    2       MM_Open      8  |   2     8  |   2     9  |   1     9  |   1
    3      MM_Center     7  |   3     7  |   3     9  |   1     8  |   2
    4     MM_Improved    7  |   3     8  |   2     9  |   1     8  |   2
    5       AB_Open      5  |   5     4  |   6     3  |   7     5  |   5
    6      AB_Center     4  |   6     5  |   5     6  |   4     6  |   4
    7     AB_Improved    4  |   6     4  |   6     4  |   6     3  |   7
--------------------------------------------------------------------------
           Win Rate:      64.3%        65.7%        70.0%        68.6%
```

In my second run, this heuristic outperformed all the other ones, while it was the worst off in my first run. I think this tells us that while this heuristic has potential to be much better with the correct tweaks, it is still too unpredictable to be used as a reliable evaluation function. I think what would make this heuristic more usable would be to factor in the starting position of the player location, as well as the maximum number of possible moves within that unit (i.e., where that unit cuts off).

AB_Custom_3:

This heuristic combined the improved_score and center_score heuristics given to us in sample_players.py. My thinking here was that both of those heuristics gave pretty solid results by themselves, so they must be even better together. What could go wrong, right? I did decide to weigh improved_score more heavily than center_score, with each at 0.75 and 0.25, respectively. This is due to my perspective that MM_Improved performed better than MM_Center in my tournament test runs.

Performance was consistently better than AB_Improved, but still had a pretty high degree of variability. This isn't too surprising, considering the variability seen in AB_Improved. Again, I think that this heuristic was effective, but was still missing something – just like the other heuristics I used.

Conclusion:

I was curious why there was such a high degree of variability, and looking at tournament.py, the only randomness I could find were the starting location for each player:

```python
# initialize all games with a random move and response
for _ in range(2):
    move = random.choice(games[0].get_legal_moves())
    for game in games:
        game.apply_move(move)
```

My takeaway from this is that the opening of the game is **extremely** important in determining the winner. If I had time to continue tweaking these heuristics, I would definitely do some more research in what are the optimal moves when beginning the game, and possibly create a dictionary to implement them. The videos in lecture suggested the same, but I was unsure of how relevant it was considering our version of Isolation is substantially different. I also assume that end game can be optimized similarly, due to the very limited number of moves available.

My main takeaway is that there are **so many** different factors and variables in putting together an adversarial game agent. I can only imagine how much more this is amplified for increasingly complex games, such as chess. Getting a small taste of how these agents are built makes the fact that IBM's Watson defeated a world-renowned chess player all the more impressive!