Starting out, I compared the basic “improved” heuristic (player moves - opponent moves) and center heuristic (squares of distances to center of x and y coordinates) with a heuristic I found online to be a solution to the solo knight tour problem, which is to pick the square with the fewest options. If the least moves heuristic could apply to essentially two-player knight’s-tour, it could do well by the fact that it would prune branches with a higher branching factor. Here are the winrate results of running these three through tournament.py 3 times, along with the default AB_Improved:

AB Custom is (player moves - opponent moves), AB_Custom_2 is (distances to center squared), AB_Custom_3 is (least moves as long as there is one)

AB_improved	AB_Custom	AB_custom_2	AB_custom_3
50%	44%	54.3%	47.1%
44.3%	40%	64.3%	40%
47.1%	42.9%	45.7%	44.7%

The least moves heuristic didn’t perform too well. In hindsight, the fewest moves heuristic does well for single player knight’s tour because it inadvertently avoids situations where you create an unreachable space. Creating unreachable spaces is fine in this isolation game.

For my second run, I decided to try a closest to center approach that didn’t square the distances to the center, instead just taking the absolute value, and an approach that combined all three heuristics. The combined approach gave a score expressed as ((distance_from_center)/2 + player_move_count - opponent_move_count).

AB Custom is ((distances to center squared)/2 + player_move_count - opponent_move_count), AB_Custom_2 is (distances to center squared), AB_Custom_3 (absolute value of distances to center)

AB_improved	AB_Custom	AB_custom_2	AB_custom_3
47.1%	44.3%	48.6%	45.7%
48.6%	51.4%	50.0%	37.1%
47.1%	54.3%	41.4%	47.1%

Here the combined heuristic seemed to do best. After some thinking, I realized that the variance between opponent move count could only ever be one for scores in the same depth. So I decided to switch to an implementation where I did (8-opponent moves /2)^2, which essentially squared a value from 0 to 3.5, giving greater returns from limiting an opponent from 2 to 1 than from 8 to 7, which usually doesn’t lead to any sort of end game scenario.
Playing through some games of isolation myself, I found myself often looking at specific critical squares that split the board and would cut my opponent off from many squares, and figuring out who could get to the square first, me or my opponent. This often lead to a win, so I tried to come up with a heuristic that would be able to discover moves that grant access to more of the board for one player. I came up with a method that creates coordinate sets from each player, and then alternates between players, picking out all the available moves picks out moves from

the list of empty spaces that could be reached from spaces that the players have traversed. It essentially plays out the game as if each player could make all moves simultaneously each move. This is expensive, but not as expensive as calculating every possible move to the endgame. Essentially, this heuristic has the effect of determining which player has more board control based on the fact that if they can move to a space before the opponent, they can potentially block off the opponent's moves. This heuristic is really good at finding critical moves that will block the opponent from a portion of the board, and rewards being closer to the center than the opponent.

AB Custom is (distances to center squared)/2 + player move count - ((8-opponent_move_count))/2)^2,
AB_Custom_2 is player_move_count - distance_from_center/2 - opponent_move_count,
AB_Custom_3 is the board control algorithm

AB_improved	AB_Custom	AB_custom_2	AB_custom_3
42.9%	42.9%	41.4%	60.0%
41.4%	58.6%	45.7%	50.0%
47.1%	51.4%	45.7%	51.4%

From this I conclude that the final set of 3 heuristics is the best 3, with the slight edge going to the board control algorithm as it won by such a vast margin in the first round.