# Heuristics Analysis for Isolation Game Playing Agent

## Summary

This analysis is based on 200 match simulations per adversarial agent, and 200 millisecond timeout per turn. The simulations are run through *tournaments.py* in a round-robin style.

The adversarial tournament agents are:

- Random: An agent that randomly chooses a move each turn.
- MM_Open: MinimaxPlayer agent using the open_move_score heuristic with search depth 3
- MM_Center: MinimaxPlayer agent using the center_score heuristic with search depth 3
- MM_Improved: MinimaxPlayer agent using the improved_score heuristic with search depth 3
- AB_Open: AlphaBetaPlayer using iterative deepening alpha-beta search and the open_move_score heuristic
- AB_Center: AlphaBetaPlayer using iterative deepening alpha-beta search and the center_score heuristic
- AB_Improved: AlphaBetaPlayer using iterative deepening alpha-beta search and the improved_score heuristic

For the following custom score evaluations we assume the following:

*MM = # of my moves*
*OM = # of opponents moves*

---

## Custom Score 1

*Heuristic = (1.8 * MM $^2$) - OM $^2$*

This heuristic returns the difference between the number of my moves squared and the number of the opponent's moves squared, with a weighting boost of 1.8 for the number of my moves. The logic behind this heuristic is that it maximizes the number of my moves.

---

## Custom Score 2

*Heuristic = $MM^2$ - (1.8 * $OM^2$)*

This heuristic returns the difference between the number of my moves squared and the number of the opponent's moves squared, with a weighting boost of 1.8 for the number of the opponent's moves. The logic behind this heuristic is that it minimizes the number of the opponent's moves.

---

## Custom Score 3

*Heuristic = (1.8 * $MM^2$) / $OM^2$*

This heuristic returns the ratio between the number of my moves squared and the number of the opponent's moves squared, with a weighting boost of 1.8 for the number of my moves. The logic behind this heuristic is that it maximizes the ratio between my moves and the opponent's moves.

## Match Data

| Match # | Opponent | AB_Improved | | AB_Custom | | AB_Custom_2 | | AB_Custom_3 | |
|---|---|---|---|---|---|---|---|---|---|
| | | Won | Lost | Won | Lost | Won | Lost | Won | Lost |
| 1 | Random | 173 | 27 | 174 | 26 | 169 | 31 | 169 | 31 |
| 2 | MM_Open | 67 | 133 | 54 | 146 | 69 | 131 | 66 | 134 |
| 3 | MM_Center | 149 | 51 | 155 | 45 | 145 | 55 | 162 | 38 |
| 4 | MM_Improved | 49 | 151 | 55 | 145 | 54 | 146 | 55 | 145 |
| 5 | AB_Open | 98 | 102 | 98 | 102 | 91 | 109 | 107 | 93 |
| 6 | AB_Center | 171 | 29 | 178 | 22 | 170 | 30 | 179 | 21 |
| 7 | AB_Improved | 104 | 96 | 89 | 111 | 84 | 116 | 93 | 107 |
| | Win Rate: | 57.9% | | 57.4% | | 55.9% | | 59.4% | |

## Conclusion

Based on the simulations of 200 matches per adversarial agent, the conclusion is that Custom Score 3 performs the best with a slight 2% edge over AB_Improved. We could try and further adjust these heuristics by multiplying by more than 1.8 to see if that creates bigger advantages.

In general the alpha-beta algorithms seems much more robust and win more matches when pit against the standard minimax agent. The reason for this is because alpha-beta algorithms are able to search deeper into the search tree and thus discover better moves.