**Heuristic Analysis for Isolation Game**

By Uttam Panda

Testing the different heuristics was hard and time-consuming. As the starting positions in the tournament are random, it is hard to get reproducible results, even when increasing the number of matches to 100.

Data available to calculate the heuristics

For the heuristics to be fast enough to compute, I’ve ended up using information that was easily available:

- Number of available moves to each player

- Position of each player on the board, especially their distance to the center

- Number of blank spaces on the board: can be used to know if we are closer to the beginning or the end of the game. The strategy can depend on how far we are in the game.

I’ve also tried to calculate the longest available path available to each player when there are few blank spaces left in the game. However, the heuristics was too slow, timing-out and not performing well at all, so I discarded it.

**Heuristics 1 (AB\_Custom)**

**Formula:**

blank\_spaces = len(game.get\_blank\_spaces())

total\_spaces = game.width \* game.height

return float((blank\_spaces/total\_spaces)\*100)

**Result:** 60 % v.s. 69.6 % (- 9.6%)

Heuristics 1 (AB\_Custom) is the worse of the heuristic of all the heuritsics

**Heuristics 2 (AB\_Custom2)**

**Formula:**

my\_moves = len(game.get\_legal\_moves(player))

opponent\_moves = len(game.get\_legal\_moves(game.get\_opponent(player)))

corners\_array = [(0,0), (0, game.height - 1), (game.width - 1, 0), (game.width - 1, game.height - 1)]

if game.get\_player\_location(player) in corners\_array:

# downgrade, because of bad position on the gameboard

my\_moves -= 3

return float(my\_moves - opponent\_moves)

**Result:** 70.1% v.s. 69.6 % (+0.5%)

**Heuristics 3 (AB\_Custom3)**

**Formula:**

my\_moves = len(game.get\_legal\_moves(player)) opponent\_moves = len(game.get\_legal\_moves(game.get\_opponent(player))) return float(my\_moves - 2 \* opponent\_moves)

**Result:** 71.3 % v.s. 69.6 % (+ 1.7%)

**Heuristics recommendation for Heuristics 3 (AB\_Custom3)**:

My recommended heuristics is **Heuristics 3 (AB\_Custom3)** combination of number of available moves and players’ positions on the board.

**Formula:**

my\_moves = len(game.get\_legal\_moves(player)) opponent\_moves = len(game.get\_legal\_moves(game.get\_opponent(player))) return float(my\_moves - 2 \* opponent\_moves)

**Reasons for selecting Heuristics 3 (AB\_Custom3):**

- Best score: even though it was close to other heuristics, **Heuristics 3 (AB\_Custom3)** performed the best in the tournament.

- Short execution time:

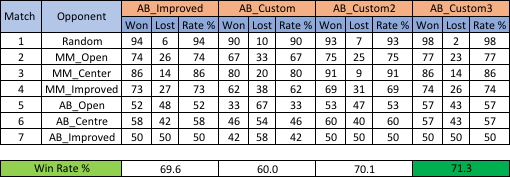
- It uses data that is accessible quickly

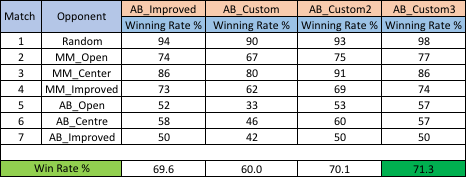
- It uses fast operations

- **Heuristics 3 (AB\_Custom3)** does not timeout and enables the search to go deep in the tree, leading to better results.

- **Heuristics 3 (AB\_Custom3)** uses information about player and opponent: in an adversarial game, it makes sense to use information about us but also about the opponent.

**Summary of the results of all heuristics:**





**Analysis of the tournament agents:**

The rank of the opponents in the tournament is always the same, no matter which heuristics is used:

The y-axis represents the % of wins of the heuristics so the worst agents will have high values.

Consistently we observe that:

Notes : I'm running on a MacBook Pro, 2 GHz Intel Core i5