# Heuristic Analysis

*by Salih Ergut*

## Introduction

In this project we are creating an AI agent to play isolation game.

I first implemented the minimax algorithm for a fixed depth, and then alpha-beta pruning algorithm, which is a more efficient algorithm as it eliminates (or prunes) branches that do not have any possibility to affect the result. Finally I implemented iterative deepening algorithm so that deeper tree is search when there is enough time.

## Heuristic functions

After successfully implementing the alpha-beta pruning I chose several heuristic functions to use as an evaluation function for depth-limited tree search algorithms.

Since we are using iterative deepening, the simpler the algorithm the more time left for going deeper in the tree search. I checked mainly the effects of centrality and commonality of available legal moves.

### Heuristic 1: custom_score

The following function computes inverse of the Eucledian distance for each next legal move to the center of the board and sums them to provide as score. Center locations provide more options to move around for the following rounds. For this reason this heuristic favors locations close to the origin.

```
def custom_score(game, player):
    if game.is_loser(player):
        return float("-inf")

    if game.is_winner(player):
        return float("inf")

    def calculate_score(moves):
        if len(moves) == 0:
            return 0
        centered_locs = np.array(moves) - (game.height/2.0, game.width/2.0)
        return np.sum(1/LA.norm(centered_locs,ord=2,axis=1))

    own_moves = game.get_legal_moves(player)
    opp_moves = game.get_legal_moves(game.get_opponent(player))

    return float(calculate_score(own_moves)-calculate_score(opp_moves))
```

### Heuristic 2: custom_score_2

The following function computes the difference between the number of legal moves between players. If opponents legal move overlaps with the player's at leaast one legal move, "1" is subtracted from the score. This is very similar to improved score except for the exclusion of one opponent move on the player legal moves.

```
def custom_score(game, player):
    if game.is_loser(player):
        return float("-inf")

    if game.is_winner(player):
```

```
        return float("inf")

    own_moves = game.get_legal_moves(player)
    opp_moves = game.get_legal_moves(game.get_opponent(player))
    common_moves = [x for x in own_moves if x in opp_moves]
    # Opponent can block at least one of the common moves
    return float(len(own_moves) - len(opp_moves) - (1 if len(common_moves)>0 else 0))
```

### Heuristic 3: custom_score_3

This function computes one step and two steps available moves and scores the game using these functions. Four main metrics are calculated:

1. The difference between one-step available moves
2. The difference between two-step available moves
3. Number of common moves in one step
4. Number of common moves in two-steps

First two metrics are indicative of opportunity, while the last two metrics are indicative of aggressiveness. We can weight the importance of each metric in the final score calculation. A positive large coefficient for the last two puts more emphasis on having more common moves with your opponent, hence it can be considered as an aggresive strategy.

```
def get_2step_moves(game, player):
    """ Returns one step and two step legal moves for a given player

    :param game: Board class
    :param player: Player (own or opponent)
    :return: one_step_moves, two_step_moves
    """
    one_step_moves = game.get_legal_moves(player)
    two_step_moves = []
    for move in one_step_moves:
        game._active_player = player
        two_step_moves.append(game.forecast_move(move).get_legal_moves(player))

    # Remove repeating ones (flatten using sum)
    two_step_moves = list(set(sum(two_step_moves,[])))

    return one_step_moves, two_step_moves

def get_score(game, player, k1, k2, k3, k4):
    opp = game.get_opponent(player)

    own_one_step_moves, own_two_step_moves = get_2step_moves(game.copy(), player)
    opp_one_step_moves, opp_two_step_moves = get_2step_moves(game.copy(), opp)

    common_one_step_moves = [x for x in own_one_step_moves if x in
opp_one_step_moves]
    common_two_step_moves = [x for x in own_two_step_moves if x in
opp_two_step_moves]

    score = k1 * (len(own_one_step_moves) - len(opp_one_step_moves)) + \
            k2 * (len(own_two_step_moves) - len(opp_two_step_moves)) + \
            k3 * len(common_one_step_moves) + \
            k4 * len(common_two_step_moves)
```

```
    return score

def custom_score_3(game, player):
      if game.is_loser(player):
        return float("-inf")

    if game.is_winner(player):
        return float("inf")


    return float(get_score(game, player, k1=1, k2=0.6, k3=0.8, k4=0.5))
```

## Analyzing Simulation Results

I ran the tournament for 40 times with 200ms timeout threshold. The results are shown below:

| Match # | Opponent | AB_Improved | | AB_Custom | | AB_Custom_2 | | AB_Custom_3 | |
|---|---|---|---|---|---|---|---|---|---|
| | | Won | Lost | Won | Lost | Won | Lost | Won | Lost |
| 1 | Random | 26 | 14 | 29 | 11 | 28 | 12 | 28 | 12 |
| 2 | MM_Open | 27 | 13 | 29 | 11 | 24 | 16 | 24 | 16 |
| 3 | MM_Center | 30 | 10 | 30 | 10 | 30 | 10 | 28 | 12 |
| 4 | MM_Improved | 23 | 17 | 26 | 14 | 24 | 16 | 30 | 10 |
| 5 | AB_Open | 20 | 20 | 21 | 19 | 19 | 21 | 21 | 19 |
| 6 | AB_Center | 23 | 17 | 20 | 20 | 24 | 16 | 22 | 18 |
| 7 | AB_Improved | 21 | 19 | 20 | 20 | 16 | 24 | 21 | 19 |
| | Win Rate: | 60.7% | | 62.5% | | 58.9% | | 62.1% | |

Heuristic 1 (AB_Custom), the heuristic that favors central locations, and Heuristic 3 (AB_Custom_3), the heuristic that uses one-step and two-step legal moves perform the better than the others. Heuristic 3 is the most complex heuristic when compare to the rest, as it first computes first step legal moves and for each one-step legal move computes the second level legal moves. Heuristic 1 is much simpler on the other hand, it just computes the eucledian distance to the center for each legal move. The performance is slightly better than Heuristic 3 but not that much better. I would recommed Heuristic 1 due to its simplistic approach and good performance.