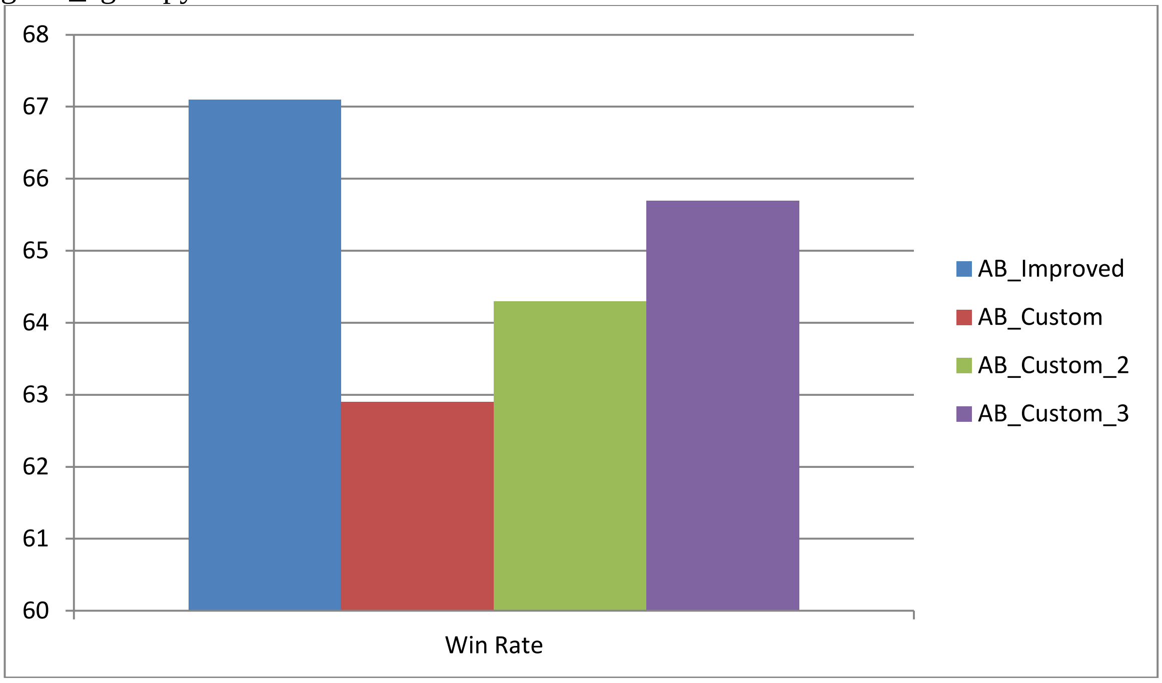
Heuristic Analysis

By Akash Chauhan

This script evaluates the performance of the custom_score evaluation function against a baseline agent using alpha-beta search and iterative deepening (ID) called `AB_Improved`. The three `AB_Custom` agents use ID and alpha-beta search with the custom_score functions defined in game_agent.py.



****

Playing Matches

****

Your ID search forfeited 249.0 games while there were still legal moves available to play.
The games rule is if the player does not have moves he loses, from this, we can infer that more the moves more is the possibility to win. Center heuristic checks for the player near the center position, here the user can make good use of the symmetry and limit the opponents state space to a minimum.

#	Opponent	AB_Improved		AB_Custom		AB_Custom_2		AB_Custom_3	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	7	3	10	0	9	1	8	2
2	MM_Open	6	4	6	4	7	3	7	3
3	MM_Center	9	1	9	1	6	4	8	2
4	MM_Improved	7	3	5	5	6	4	7	3
5	AB_Open	7	3	5	5	6	4	5	5

6	AB_Center	6	4	6	4	8	3	5	5
8	AB_Improved	5	5	3	7	3	7	6	4
	Win Rate:	67.1%		62.9%		64.3%		65.7%	

Report: The best performing student heuristic is AB_Custom_3 which is very close to the AB_Improved heuristic.

Heuristic Custom 3, needs to visit the tile once which can further be optimized by using the symmetry. For heuristic legal moves, it needs to dive a little deep to get results. Heuristic custom 1 is not as good since, the heuristic changing point is not the best one, it is a problem in itself to find the best point along the width of the board where we can change the heuristic in the sense to maximize the winning possibilities.

With the current result i will recommend using the left legal moves heuristic, since it goes for a fair possibility of 200% moves comparing to the opponents moves. Although, a mix of two heuristic is also an good option, given we know or evaluate where to set the change heuristic strategy point. for the initial case, starting of the game, symmetry can be used to make the problem easier and faster.

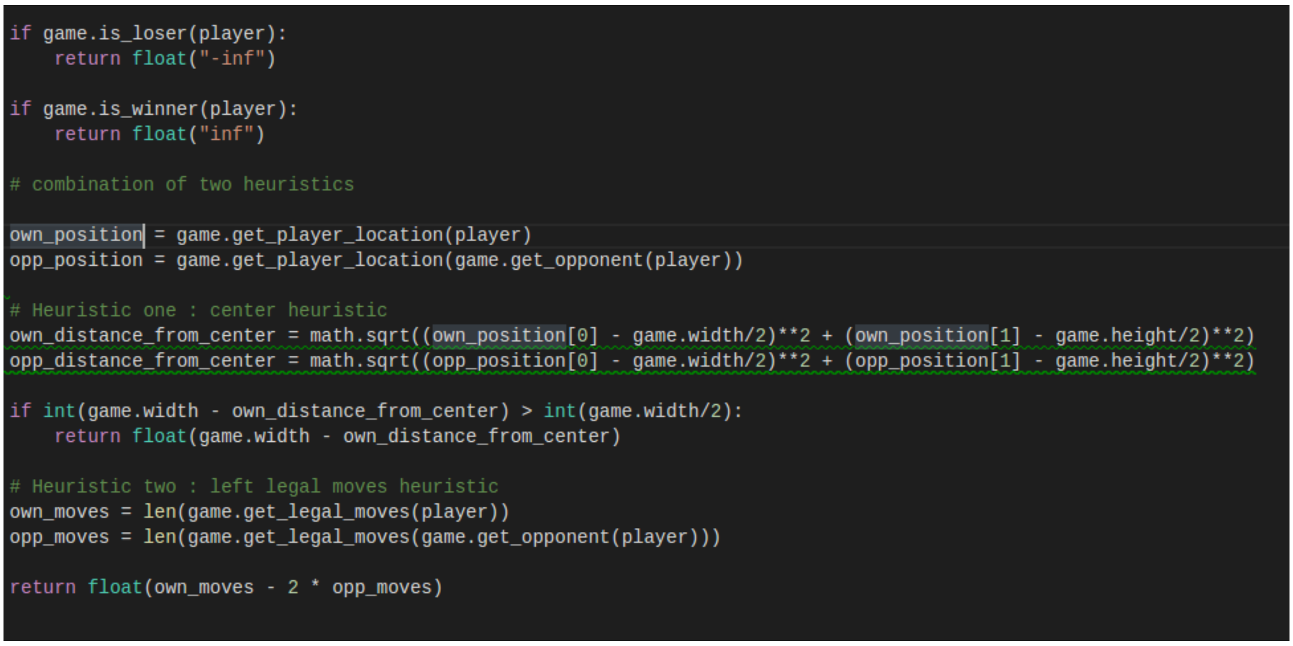
The easiest one is the center heuristic, it just compares the distance from the center of the board.

For minimax, it is complete if tree is finite
Time complexity: O(b^m)
Space complexity: O(bm) (depth-first exploration)
But things can be greatly improved if we use alphabeta pruning and evaluation function

1. custom_score:

This heuristic is a combination of two heuristic function

a. center heuristic the player which is closer to the center can do better comparing to the case when he is far from the center



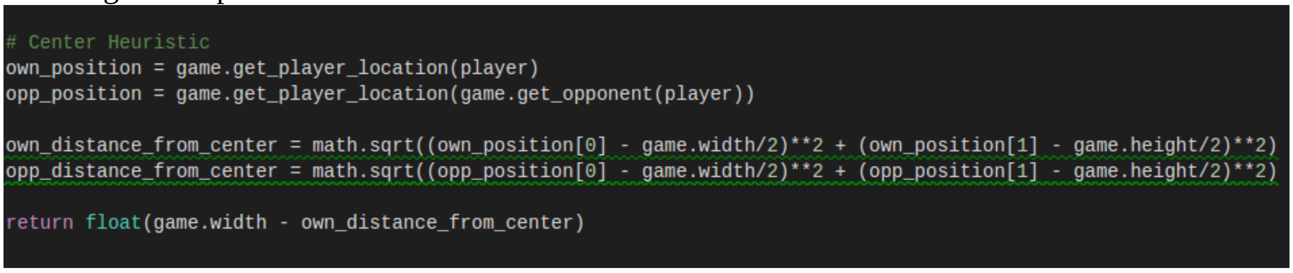
```
if game.is_loser(player):
    return float("-inf")

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

# combination of two heuristics

own_position = game.get_player_location(player)
opp_position = game.get_player_location(game.get_opponent(player))

# Heuristic one : center heuristic
own_distance_from_center = math.sqrt((own_position[0] - game.width/2)**2 + (own_position[1] - game.height/2)**2)
opp_distance_from_center = math.sqrt((opp_position[0] - game.width/2)**2 + (opp_position[1] - game.height/2)**2)

if int(game.width - own_distance_from_center) > int(game.width/2):
    return float(game.width - own_distance_from_center)

# Heuristic two : left legal moves heuristic
own_moves = len(game.get_legal_moves(player))
opp_moves = len(game.get_legal_moves(game.get_opponent(player)))

return float(own_moves - 2 * opp_moves)
```

b. left legal moves: that states the condition of player comparing to the opponents 100% more moves. Heuristic change point: when player is closer to the center than the opponent, if the player is far then the b heuristic comes into play.

Different heuristics have different pros and cons, in this i added two heuristic with 50% weight to each and decided the heuristic changing point when the player is far from the center, because near the center the player has better moves.

2. custom_score_2:

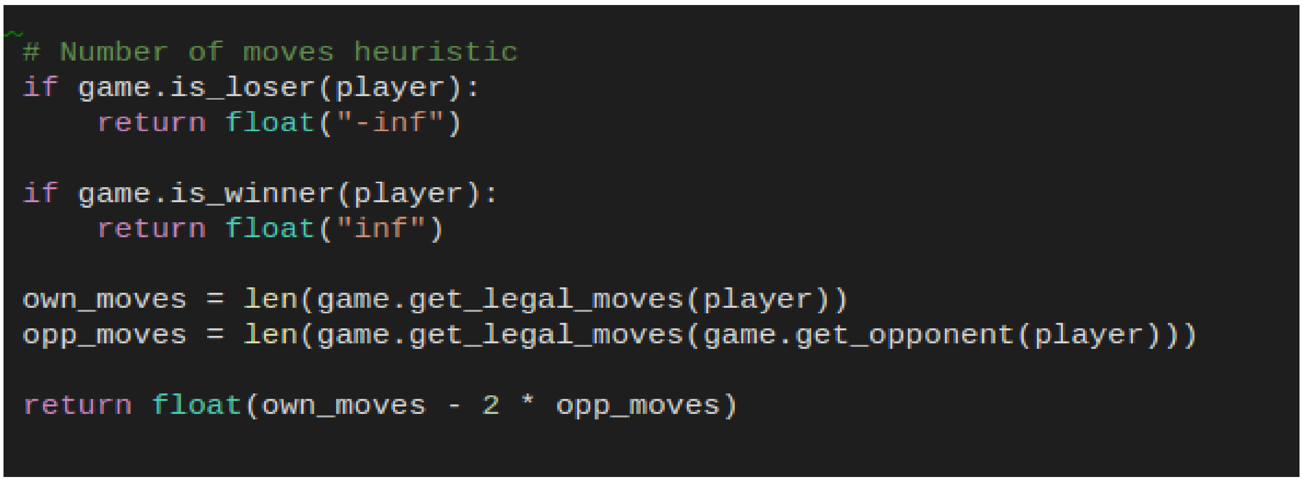
Center Heuristic: it states that the player near to the center has better chances to win the game. The center position is the a good starting position, because occupying it decreases the available moves for the opponent, even if it is not possible to occupy, the positions near to the center are having advantages than positions far from the center.



```
# Center Heuristic
own_position = game.get_player_location(player)
opp_position = game.get_player_location(game.get_opponent(player))

own_distance_from_center = math.sqrt((own_position[0] - game.width/2)**2 + (own_position[1] - game.height/2)**2)
opp_distance_from_center = math.sqrt((opp_position[0] - game.width/2)**2 + (opp_position[1] - game.height/2)**2)

return float(game.width - own_distance_from_center)
```

custom_score_3:

Legal moves left heuristic: this one is intuitive, one with the more moves available has more chances to win. Available moves is space in which the player can move, more the available space



```
# Number of moves heuristic
if game.is_loser(player):
    return float("-inf")

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

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

return float(own_moves - 2 * opp_moves)
```

more the chances the player has to win the game. Definitely, if player has twice the legal moves left than his opponent, he is having a good chance to win the game.