Heuristic Analysis

Hsin-Wen Chang

Build a Game-playing Agent

In Chess board game there are two players both of them need to conduct strategy to occupy land in the 2D square chess board. Both players have to think about steps to defend their land meanwhile take down the other player's chess and land. I develop three heuristic blend with different weight to be defensive or offensive as follow:

Custom Heuristic #1

```
def weight_heuristic_steps(game, player):
if game.is_loser(player):
        return float("-inf")
    if game.is_winner(player):
        return float("inf")
    moves = len(game.get_legal_moves(player))
    prob_moves =
len(game.get_legal_moves(game.get_opponent(player)))
    return moves * moves - 1.5 * prob_moves * prob_moves
```

This is offenseive heuristic in this way another player's move option should be minimize because I take down more land space. This heuristic performs adequately. The win rate is better than AB_Improved when it plays 20 games. This heuristic is quick to compute and involves the state of the opponent.

Custom Heuristic #2

```
def weight_heuristic_steps2(game, player):
if game.is_loser(player):
        return float("-inf")
    if game.is_winner(player):
        return float("inf")
    moves = len(game.get_legal_moves(player))
    opponent_moves =
len(game.get_legal_moves(game.get_opponent(player)))
    return 1.5 * moves * moves - opponent_moves * opponent_moves
```

This is defensive heuristic in this way player's move option should be maximize. This heuristic performs adequately. The win rate is better than AB_Improved when it plays 20 games. The agent maximizes its own moves.

Custom Heuristic #3

```
def weight_heuristic_steps3(game, player):
opponent = game.get_opponent(player)
    opponent_moves = game.get_legal_moves(opponent)
    p_moves = game.get_legal_moves()
    common_moves = opponent_moves and p_moves
    if not opponent_moves:
        return float("inf")
    if not p_moves:
        return float("-inf")
    move_convergence = 1 / (game.move_count + 1)
    inverse_convergence = 1 / move_convergence
    return float(len(common_moves) * move_convergence +
inverse_convergence * len(game.get_legal_moves()))
```

This heuristic is also blend with different weight to be defensive or offensive. But measure the weight importance as the game ongoing. As the number of each move count the move option will convergence become have less option. This heuristic performs significantly worse than AB_Improved.

Result:

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
**************************
```

Match #	Opponent	AB_Improved		AB_Custom		AB_Custom_2		AB_Custom_3	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	9	1	9	1	8	2	7	3
2	MM_Open	8	2	7	3	8	2	6	4
3	MM_Center	6	4	8	2	8	2	7	3
4	MM_Improved	5	5	7	3	4	6	5	5
5	AB_Open	4	6	5	5	6	4	5	5
6	AB_Center	5	5	5	5	8	2	5	5
7	AB_Improved	4	6	6	4	5	5	2	8
	Win Rate:	58.6%		67.1%		67.1%		52.9%	

There were 5.0 timeouts during the tournament -- make sure your agent handles search timeout correctly, and consider increasing the timeout margin for your agent. Your agents forfeited 153.0 games while there were still legal moves available to play.

Recommandation:

Among the Three custom heuristic AB_Custom and AB_Custom_2 perform adequately but both better than AB_Improved when it plays 20 games. This indicate that measure the weight importance as the game ongoing(AB_Custom_3) isn't better than entirely aggressive or defensive. Despite the number of iterative deepening in recursive may slow down the computation still better than AB_Improved.