Heuristic Analysis

Introduction and Game Rules

In scope of this project was implemented an adversarial search agent to play the game "**Isolation**".

Isolation is a *deterministic*, two-player *game of perfect information* in which the players alternate turns moving a single piece from one cell to another on a board. Whenever either player occupies a cell, that cell becomes blocked for the remainder of the game. The first player with no remaining legal moves loses, and the opponent is declared the winner. These rules are implemented in the isolation.

This project uses a version of Isolation where each agent is restricted to *L-shaped movements* (like a knight in chess) on a rectangular 7 x 7 grid.

Additionally, agents will have a *fixed time limit* each turn to search for the best move and respond. If the time limit expires during a player's turn, that player forfeits the match, and the opponent wins.

Solution and Heuristics Implemented

The solution was programmed in Python based on the framework provided here https://github.com/almazurenkin/AIND-Isolation. The framework defines class Game (`isolation.py`) that implements a model for the game Isolation assuming each player moves like a knight in chess.

Implemented two game-playing AI agents (`game_gent.py`) based on *minimax* and *alpha-beta pruning* algorithms for searching the *game tree*. In addition, alpha-beta pruning -based agent uses *iterative deepening* technic in order to reach maximum search depth within a set time constraint. .

Three heuristic evaluation functions were defined, each of them implemnts a certain (basic) game strategy.

- `custom_score(...)`

Implements modified `#my_moves`, simple evaluation logic that prioritizes branches with maximum number of possible moves, squared.

- `custom_score_2(...)`

Implements modified `#my_moves-#opponent_moves` heuristics. This evaluation function takes into account number of opponent's moves left and prioritizes moves with a higher difference. Similar to the baseline `improved_score(...)` heuristic enhanced with an `AGGRESSION` parameter.

• `custom_score_3(...)`

Implements `blocking strategy`: prioritize moves that overlap with possible opponent's moves.

Comparison and Conclusion

Script `tournament.py` was used to benchmark performance of different heuristics described above against reference agents defined in `sample_players.py`. The three `AB_Custom` agents use ID and alpha-beta search with the custom_score functions from `game_agent.py`.

Results are shown below.

```
                                *************************
                                     Playing Matches
                                *************************

 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random      10  |   0     7  |   3    10  |   0     9  |   1
    2       MM_Open      7  |   3     7  |   3     9  |   1     6  |   4
    3      MM_Center     8  |   2     7  |   3     9  |   1     8  |   2
    4     MM_Improved    7  |   3     6  |   4     6  |   4     6  |   4
    5       AB_Open      8  |   2     6  |   4     4  |   6     5  |   5
    6      AB_Center     6  |   4     4  |   6     8  |   2     5  |   5
    7     AB_Improved    5  |   5     5  |   5     5  |   5     4  |   6
--------------------------------------------------------------------------
           Win Rate:      72.9%        60.0%        72.9%        61.4%
```

We see that players that use defined heuristics outperform reference opponents (prefix in the agent's name is MM for minimax and AB for alpha-beta pruning).

As expected, performance of "AB_Custom_2" player is close to the one of "AB_Improved". Clearly, heuristics it uses is the best out of the three tested. Hence, "AB_Custom_2" is recommended due to its superior performance (72.9% against 60% and 61.4% of "AB_Custom" and "AB_Custom_3", correspondingly). The function is relatively simple that allows deeper search, yet efficient in predicting game's outcome even if not (always) reaching end-game leafs.

This brief experimentation shows that selection of a good playing strategy implemented in a form of a board evaluation function, is an important factor of game-playing agent performance.

More complex heuristics may improve performance of our game-playing agent, however, it's important to remember that simple and faster heuristic function will allow deeper search in the game tree, which can produce a better player than a shallow search with better evaluation.