Tomasz Szyborski
Udacity Artificial Intelligence Nanodegree - Heuristic Analysis

Heuristic Analysis

Consider following functions as heuristic methods to run in built game agent.

Name of the functions below are self-explanatory, but in the code run they will take names of custom_score, custom_score_2, and custom_score_3. It will be indicated which function is bound to which behaviour after experiment is conducted and results presented below.

```
def winner_or_loser(game, player):
    """
    Parameters
    ----------
    game : `isolation.Board`
        An instance of `isolation.Board` encoding the current
state of the
        game (e.g., player locations and blocked cells).

    player : object
        A player instance in the current game (i.e., an object
corresponding to
        one of the player objects `game.__player_1__` or
`game.__player_2__`.)

    Returns
    -------
    float or None
        Floar or None depending if winner/looser is identified
    """
    return float("-inf") if game.is_loser(player) else
float("inf") if game.is_winner(player) else None

def get_own_and_opponent_moves(game, player):
    return (game.get_legal_moves(player),
game.get_legal_moves(game.get_opponent(player)))

def aggressive_heuristic(game, player):
    won_or_lost = winner_or_loser(game, player)

    if won_or_lost is not None:
        return won_or_lost
    else:
        own_moves, opponent_moves = list(map(len,
get_own_and_opponent_moves(game, player)))

    return own_moves - 1.5 * opponent_moves
```

```
def defensive_heuristic(game, player):
    won_or_lost = winner_or_loser(game, player)

    if won_or_lost is not None:
        return won_or_lost
    else:
        own_moves, opponent_moves = list(map(len,
get_own_and_opponent_moves(game, player)))

    return 1.5 * own_moves - opponent_moves


def maximizing_win_chances_heuristic(game, player):
    won_or_lost = winner_or_loser(game, player)

    if won_or_lost is not None:
        return won_or_lost
    else:
        own_moves, opponent_moves = list(map(len,
get_own_and_opponent_moves(game, player)))

    if own_moves == 0:
        to_return = float("-inf")
    elif opponent_moves == 0:
        to_return = float("inf")
    else:
        to_return = float(own_moves)/opponent_moves

    return to_return


def minimizing_losing_chances_heuristic(game, player):
    won_or_lost = winner_or_loser(game, player)

    if won_or_lost is not None:
        return won_or_lost
    else:
        own_moves, opponent_moves =
list(map(len,get_own_and_opponent_moves(game, player)))

    if own_moves == 0:
        to_return = float("-inf")
    elif opponent_moves == 0:
        to_return = float("inf")
    else:
        to_return = -float(opponent_moves)/own_moves

    return to_return
```

```
def chances_heuristic(game, player):
    won_or_lost = winner_or_loser(game, player)

    if won_or_lost is not None:
        return won_or_lost
    else:
        own_moves, opponent_moves = list(map(len,
get_own_and_opponent_moves(game, player)))

    return own_moves * own_moves - opponent_moves *
opponent_moves


def weighted_chances_heuristic(game, player):
    won_or_lost = winner_or_loser(game, player)

    if won_or_lost is not None:
        return won_or_lost
    else:
        own_moves, opponent_moves = list(map(len,
get_own_and_opponent_moves(game, player)))

    return own_moves * own_moves - 1.5 * opponent_moves *
opponent_moves


def weighted_chances_heuristic_v2(game, player):
    won_or_lost = winner_or_loser(game, player)

    if won_or_lost is not None:
        return won_or_lost
    else:
        own_moves, opponent_moves = list(map(len,
get_own_and_opponent_moves(game, player)))

    return 1.5 * own_moves * own_moves - opponent_moves *
opponent_moves
```

The tournament results are presented below:

1st round:

Custom_score -> aggressive_heuristics

Custom_score_2 -> defensive_heurestics

Custom_score_3 -> maximizing_win_chances_heuristic

```
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       6  |   4     8  |   2     6  |   4     4  |   6
    2        MM_Open      6  |   4     6  |   4     7  |   3     7  |   3
    3       MM_Center     4  |   6     4  |   6     5  |   5     2  |   8
    4     MM_Improved     2  |   8     3  |   7     4  |   6     2  |   8
    5        AB_Open      4  |   6     2  |   8     6  |   4     6  |   4
    6       AB_Center     7  |   3     5  |   5     5  |   5     6  |   4
    7     AB_Improved     6  |   4     6  |   4     4  |   6     3  |   7
--------------------------------------------------------------------------
           Win Rate:      50.0%        48.6%        52.9%        42.9%

There were 63.0 timeouts during the tournament -- make sure your agent
handles search timeout correctly, and consider increasing the timeout
margin for your agent.


Your ID search forfeited 193.0 games while there were still legal moves
available to play.
```

2nd round:
Custom_score -> minimizing_losing_chances_heuristic
Custom_score_2 -> chances_heuristic
Custom_score_3 -> weighted_chances_heuristic

```
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       5  |   5     9  |   1     9  |   1     8  |   2
    2        MM_Open      7  |   3     8  |   2     7  |   3     4  |   6
    3       MM_Center     7  |   3     6  |   4     8  |   2     6  |   4
    4     MM_Improved     3  |   7     3  |   7     4  |   6     5  |   5
    5        AB_Open      4  |   6     7  |   3     6  |   4     7  |   3
    6       AB_Center     7  |   3     5  |   5     8  |   2     4  |   6
    7     AB_Improved     5  |   5     5  |   5     6  |   4     5  |   5
--------------------------------------------------------------------------
           Win Rate:      54.3%        61.4%        68.6%        55.7%

There were 12.0 timeouts during the tournament -- make sure your agent
```

```
handles search timeout correctly, and consider increasing the timeout
margin for your agent.


Your ID search forfeited 237.0 games while there were still legal moves
available to play.
```

3rd round:
Changed weights from 1.5 to 2 in aggressive and defensive heuristics, left chances_heuristics to compare different runs:
Custom_score -> aggressive_heuristis
Custom_score_2 -> defensive_heuristics
Custom_score_3 -> chances_heuristic

```
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     8  |   2    10  |   0     5  |   5
    2       MM_Open      5  |   5     4  |   6     7  |   3     6  |   4
    3      MM_Center     8  |   2     8  |   2     7  |   3     7  |   3
    4     MM_Improved    4  |   6     7  |   3     6  |   4     5  |   5
    5       AB_Open      4  |   6     7  |   3     4  |   6     4  |   6
    6      AB_Center     7  |   3     8  |   2     5  |   5     6  |   4
    7     AB_Improved    8  |   2     6  |   4     7  |   3     7  |   3
--------------------------------------------------------------------------
           Win Rate:      64.3%        68.6%        65.7%        57.1%

There were 29.0 timeouts during the tournament -- make sure your agent
handles search timeout correctly, and consider increasing the timeout
margin for your agent.


Your ID search forfeited 219.0 games while there were still legal moves
available to play.
```

4th round:
Changed weights from 1.5 to 2 in weighted_chances_heuristic and weighted_chances_heuristic_2, left chances_heuristics to compare different runs:
Custom_score -> weighted_chances_heuristic
Custom_score_2 -> weighted_chances_heuristic_v2
Custom_score_3 -> chances_heuristic

```
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       7  |   3     9  |   1     7  |   3     5  |   5
    2       MM_Open      6  |   4     5  |   5     5  |   5     4  |   6
    3      MM_Center     7  |   3     8  |   2     8  |   2     9  |   1
    4     MM_Improved    7  |   3     8  |   2     8  |   2     6  |   4
    5       AB_Open      5  |   5     7  |   3     4  |   6     3  |   7
    6      AB_Center     6  |   4     4  |   6     7  |   3     6  |   4
    7     AB_Improved    6  |   4     6  |   4     5  |   5     4  |   6
--------------------------------------------------------------------------
           Win Rate:      62.9%        67.1%        62.9%        52.9%

There were 41.0 timeouts during the tournament -- make sure your agent
handles search timeout correctly, and consider increasing the timeout
margin for your agent.


Your ID search forfeited 211.0 games while there were still legal moves
available to play.
```

5th round:
Changed weights from 1.5 to 3 in aggressive and defensive heuristics, left chances_heuristics to compare different runs:
Custom_score -> aggressive_heuristis
Custom_score_2 -> defensive_heuristics
Custom_score_3 -> chances_heuristic

```
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       7  |   3     5  |   5     6  |   4     9  |   1
    2       MM_Open      7  |   3     4  |   6     7  |   3     7  |   3
    3      MM_Center     8  |   2     7  |   3     7  |   3     6  |   4
    4     MM_Improved    7  |   3     6  |   4     5  |   5     5  |   5
    5       AB_Open      4  |   6     3  |   7     4  |   6     8  |   2
```

```
    6      AB_Center     6  |   4     7  |   3     6  |   4     6  |   4
    7     AB_Improved    4  |   6     4  |   6     7  |   3     6  |   4
--------------------------------------------------------------------------
           Win Rate:      61.4%        51.4%        60.0%        67.1%

There were 21.0 timeouts during the tournament -- make sure your agent
handles search timeout correctly, and consider increasing the timeout
margin for your agent.


Your ID search forfeited 232.0 games while there were still legal moves
available to play.
```

6th round:
Changed weights from 1.5 to 3 in weighted_chances_heuristic and weighted_chances_heuristic_v2, left chances_heuristics to compare different runs:
Custom_score -> weighted_chances_heuristic
Custom_score_2 -> weighted_chances_heuristic_v2
Custom_score_3 -> chances_heuristic

```
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       7  |   3     5  |   5     5  |   5     8  |   2
    2       MM_Open      8  |   2     6  |   4     7  |   3     6  |   4
    3      MM_Center     8  |   2     8  |   2     5  |   5     8  |   2
    4     MM_Improved    4  |   6     6  |   4     7  |   3     4  |   6
    5       AB_Open      5  |   5     4  |   6     5  |   5     6  |   4
    6      AB_Center     6  |   4     6  |   4     7  |   3     6  |   4
    7     AB_Improved    7  |   3     3  |   7     3  |   7     4  |   6
--------------------------------------------------------------------------
           Win Rate:      64.3%        54.3%        55.7%        60.0%

There were 31.0 timeouts during the tournament -- make sure your agent
handles search timeout correctly, and consider increasing the timeout
margin for your agent.


Your ID search forfeited 224.0 games while there were still legal moves
available to play.
```

Final choice:
Custom_score -> aggresive_heuristic with multiplier changed to 2.0
Custom_score_2 -> weighted_chances_heuristic
Custom_score_3 -> weighted_chances_heuristic with weight changed to 2.0

The final choice is made as above, because of outcome roughly 67% in previous games won per choice, and this is best score I could achieve among other choices (see 6 rounds of tournaments).

Below the final tournament results are presented:

```
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       7  |   3     7  |   3     8  |   2     8  |   2
    2       MM_Open      5  |   5     7  |   3     8  |   2     8  |   2
    3      MM_Center     9  |   1     8  |   2     7  |   3     5  |   5
    4     MM_Improved    6  |   4     6  |   4     4  |   6     4  |   6
    5       AB_Open      5  |   5     5  |   5     6  |   4     8  |   2
    6      AB_Center     4  |   6     8  |   2     5  |   5     3  |   7
    7     AB_Improved    5  |   5     4  |   6     5  |   5     6  |   4
--------------------------------------------------------------------------
           Win Rate:      58.6%        64.3%        61.4%        60.0%

There were 62.0 timeouts during the tournament -- make sure your agent
handles search timeout correctly, and consider increasing the timeout
margin for your agent.


Your ID search forfeited 188.0 games while there were still legal moves
available to play.
```

We can see that the most efficient heuristics are aggresive_heuristics which is chasing the opponent to effectively block all of its moves and weighted_chances_heuristics with two different weights - these are returning a difference between own (player) moves to the power of two and opponent moves to the power of two (multiplied by weight) to determine next move. What is clearly visible that AB_Improved algorithm resulted in a draw when confronted by itself, then the custom choice by the author resulted in 4 to 6 loss, 5 to 5 draw and 6 to 4 win respectively.

To summarize, author would choose aggressive_heuristic with multiplier raised to 2.0 as best evaluation function as:

1. It scored overall best score in the last tournament
2. It is simple in terms of logic, behaviour and code

3. It is simple from calculation point of view, yet contains all the information about own and opponent moves
4. Taking psychological aspect into consideration - when playing with another human, there is more to emotions than to calculation. Aggressive play can induce panic into opponent causing errors in the decisions.