Isolation Game Heuristic Analysis

Artificial Intelligence Nanodegree
Irene Sanz Vizcaíno

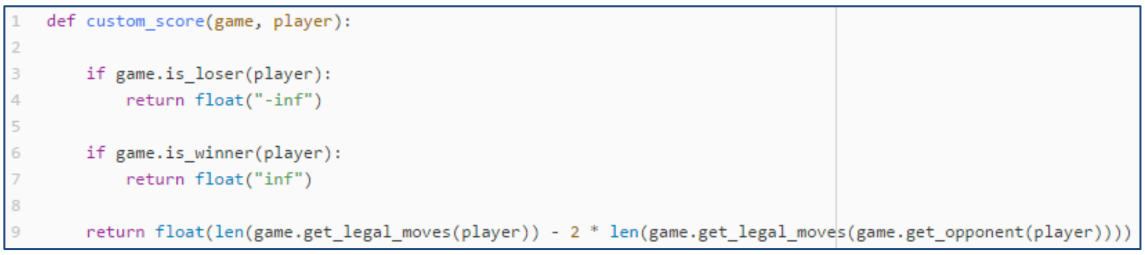
The aim of this project is to develope an adversarial search agent to play the game Isolation using Minimax and Alpha Beta Pruning with three heuristics, functions that assign a value per node in the game tree to be able to compare moves in a depth limited search.

In order to design these heuristics, I considered two points: the aim of the game and which variables can be measured within a unique board state. The aim of the game is not running out of possible moves before the opponent does, which leads us to the first measurable variables in a certain position: my number of available moves and my opponent's moves. Other variables that could be considered are related to the position of the two players on the board, such as the distance between them, the distance to the center, to the walls or the corners and the number of blank spaces.

Hypothesis and implementation

Heuristic 1: *custom_score*

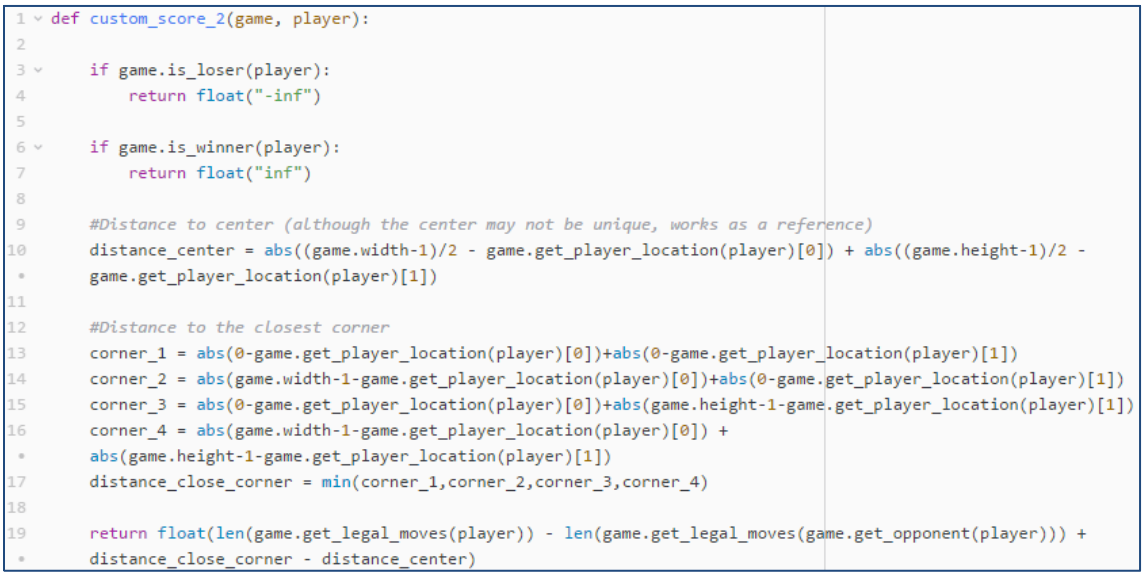
The first heuristic considers the two main variables, current player and opponent's number of movements. It prioritizes attack, reducing the opponent number of moves before increasing current player's by a factor 2.



```python
def custom_score(game, player):

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

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

    return float(len(game.get_legal_moves(player)) - 2 * len(game.get_legal_moves(game.get_opponent(player))))
```

Heuristic 2: *custom_score_2*

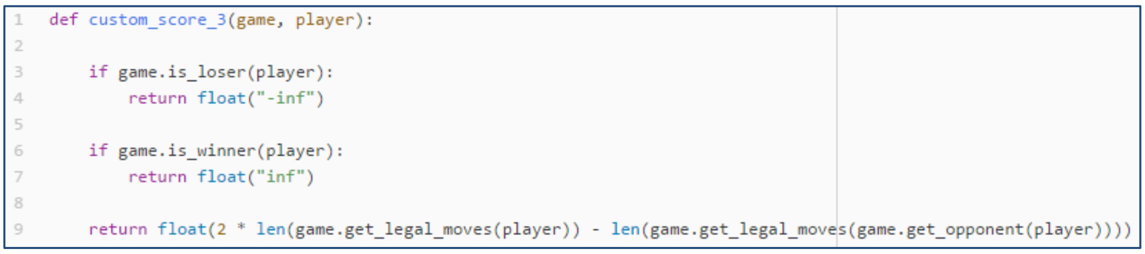
The second heuristic considers again the two main variables, current player and opponent's number of movements, but it also takes into account current player's position, penalizing the corners and favoring the center, as there are more chances to find available moves in a future state from a centered position than near the corners.



```python
def custom_score_2(game, player):

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

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

    #Distance to center (although the center may not be unique, works as a reference)
    distance_center = abs((game.width-1)/2 - game.get_player_location(player)[0]) + abs((game.height-1)/2 - game.get_player_location(player)[1])

    #Distance to the closest corner
    corner_1 = abs(0-game.get_player_location(player)[0])+abs(0-game.get_player_location(player)[1])
    corner_2 = abs(game.width-1-game.get_player_location(player)[0])+abs(0-game.get_player_location(player)[1])
    corner_3 = abs(0-game.get_player_location(player)[0])+abs(game.height-1-game.get_player_location(player)[1])
    corner_4 = abs(game.width-1-game.get_player_location(player)[0]) + abs(game.height-1-game.get_player_location(player)[1])
    distance_close_corner = min(corner_1,corner_2,corner_3,corner_4)

    return float(len(game.get_legal_moves(player)) - len(game.get_legal_moves(game.get_opponent(player))) + distance_close_corner - distance_center)
```

Heuristic 3: *custom_score_3*

The third heuristic considers the two main variables too, current player and opponent's number of movements. It prioritizes defense, increasing current player's number of available moves before reducing the opponent's by a factor 2.

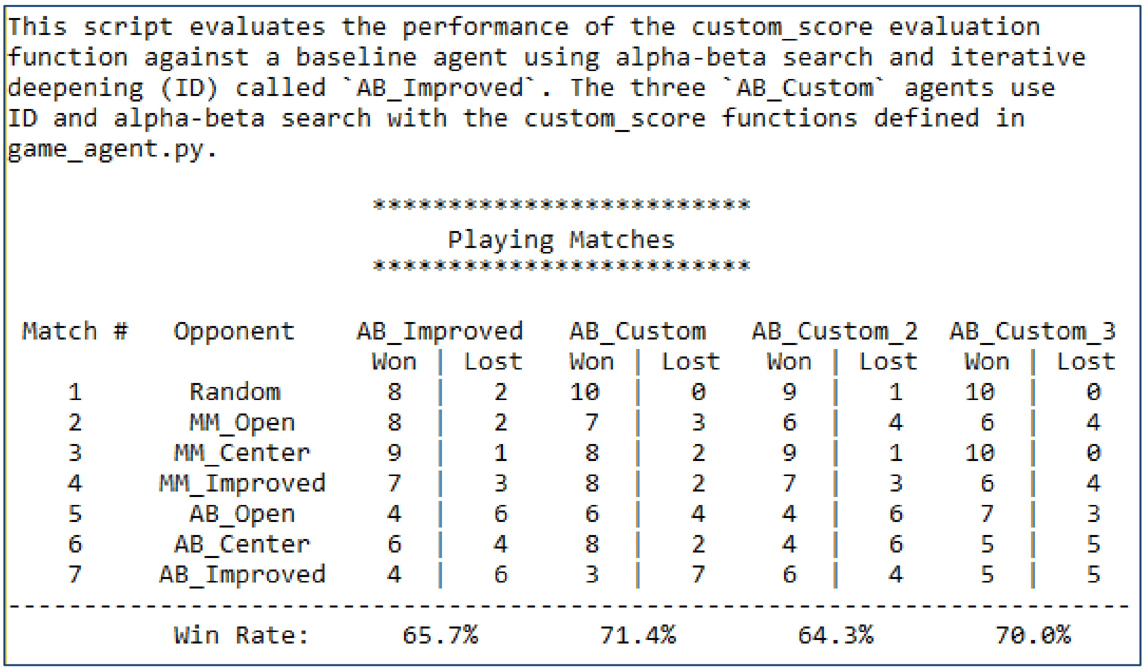


```python
def custom_score_3(game, player):

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

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

    return float(2 * len(game.get_legal_moves(player)) - len(game.get_legal_moves(game.get_opponent(player))))
```

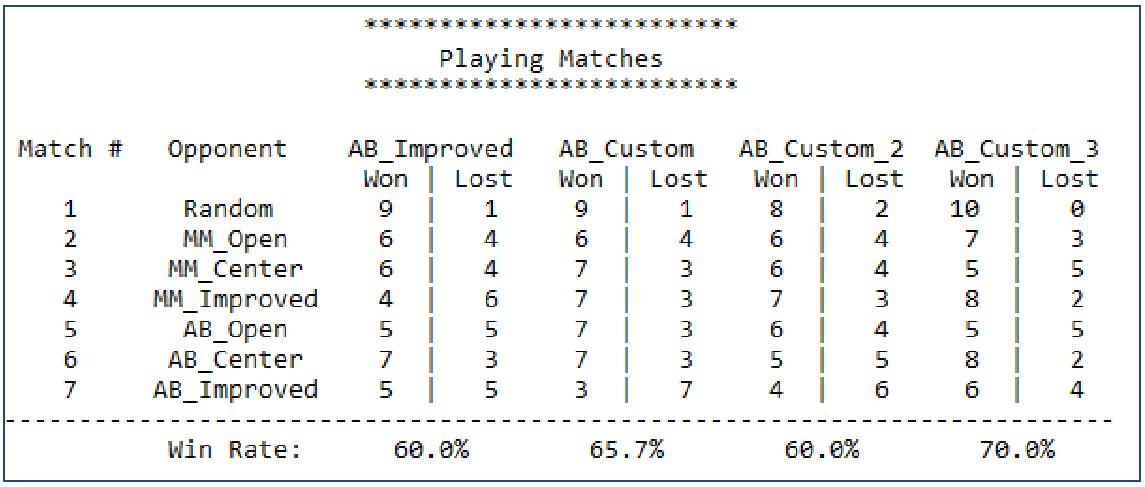
Conclusions

Initially, I expected the second heuristic to have the best performance, as it considers more variables on the board. I also expected defensive behavior to be safer than aggressive behavior. I guess there is where testing plays his role, because none of my assumptions were met.

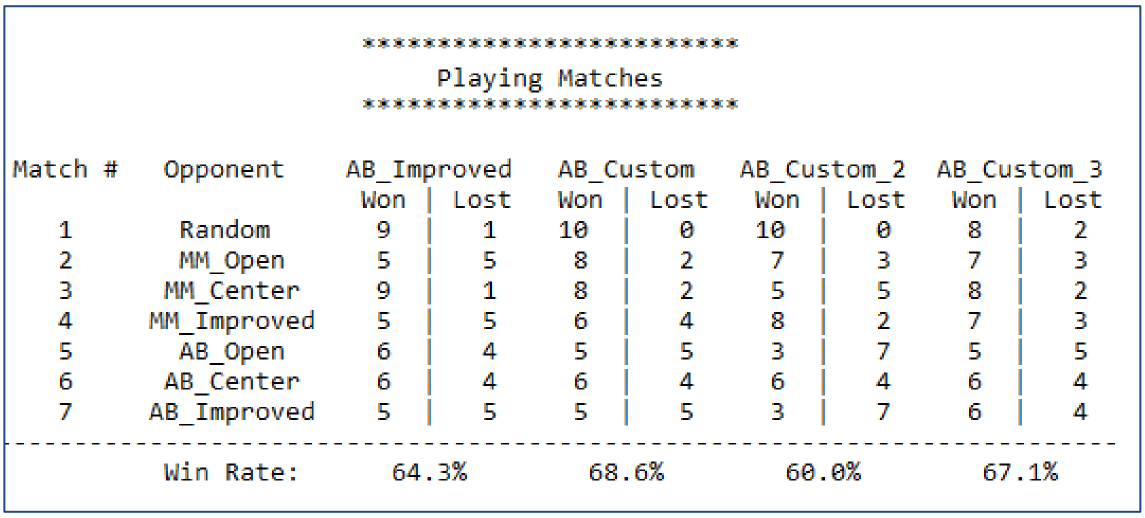
The best heuristic was the first one, *AB_Custom*, considering the number of movements of every player but prioritizing the attack by multiplying by 2 the number of available numbers. The third heuristic, *AB_Custom*_3, which is similar but prioritizes defense, had a similar although slightly worse performance. Actually, when the test was repeated to check how the results were influenced by randomness, the third heuristic performed better. Finally, the second heuristic, *AB_Custom_2*, which considered also player's distance to center and corners, performed even worse than *AB_Improved*, the example provided.



```
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        8  |   2    10  |   0     9  |   1    10  |   0
    2       MM_Open       8  |   2     7  |   3     6  |   4     6  |   4
    3      MM_Center      9  |   1     8  |   2     9  |   1    10  |   0
    4     MM_Improved     7  |   3     8  |   2     7  |   3     6  |   4
    5       AB_Open       4  |   6     6  |   4     4  |   6     7  |   3
    6      AB_Center      6  |   4     8  |   2     4  |   6     5  |   5
    7     AB_Improved     4  |   6     3  |   7     6  |   4     5  |   5
--------------------------------------------------------------------------
          Win Rate:      65.7%        71.4%        64.3%        70.0%
```



```
                        *************************
                             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    10  |   0
    2       MM_Open       6  |   4     6  |   4     6  |   4     7  |   3
    3      MM_Center      6  |   4     7  |   3     6  |   4     5  |   5
    4     MM_Improved     4  |   6     7  |   3     7  |   3     8  |   2
    5       AB_Open       5  |   5     7  |   3     6  |   4     5  |   5
    6      AB_Center      7  |   3     7  |   3     5  |   5     8  |   2
    7     AB_Improved     5  |   5     3  |   7     4  |   6     6  |   4
--------------------------------------------------------------------------
          Win Rate:      60.0%        65.7%        60.0%        70.0%
```



```
                        *************************
                             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    10  |   0    10  |   0     8  |   2
    2       MM_Open       5  |   5     8  |   2     7  |   3     7  |   3
    3      MM_Center      9  |   1     8  |   2     5  |   5     8  |   2
    4     MM_Improved     5  |   5     6  |   4     8  |   2     7  |   3
    5       AB_Open       6  |   4     5  |   5     3  |   7     5  |   5
    6      AB_Center      6  |   4     6  |   4     6  |   4     6  |   4
    7     AB_Improved     5  |   5     5  |   5     3  |   7     6  |   4
--------------------------------------------------------------------------
          Win Rate:      64.3%        68.6%        60.0%        67.1%
```

Future work

For this project, as the results obtained improved the heuristic provided as example *AB_Improved*, I considered them enough. Never though, if I had to improve this agent, next step to consider would be applying the same heuristics but analyzing not only the current board state, but at least another movement in advance. The computational cost of this movement would be probably reducing the depth explored, but it may be worth it.

Another possibility could be considering specific scenarios that may be common in this game, such as the creation of partitions in the board.

Some other ideas explained in the course include considering specularity and symmetry in the first steps of the game to speed up the search and be able to explore the tree deeper.