AIND-Isolation - Heuristics Analysis ¶

This notebook intends to describe analysis made in Isolation project from Udacity AIND. This project should follow these guidelines to be accepted.

The complete python notebook can be found here.

Baseline ¶

First, I've implemented minimax and alphabeta algorithms.
To give me a baseline to work, I've used:

- `custom_score` function is the same as in `sample_players -> improved_score`.
- The first move was always the first available `legal_move` available (but it could be replaced after running the alphabeta or minimax algorithm)
- NUM_MATCHES is 5 and TIME_LIMIT is 150

The results can have 10% difference between each run as discussed in the forums. But to have a baseline, here's the results after running `tournament.py` with the configurations mentioned:

- ID_Improved: 77.14%
- Student: 81.43%

```
************************
 Evaluating: ID_Improved
************************

Playing Matches:
----------
  Match 1: ID_Improved vs   Random        Result: 16 to 4
  Match 2: ID_Improved vs   MM_Null       Result: 18 to 2
  Match 3: ID_Improved vs   MM_Open       Result: 15 to 5
  Match 4: ID_Improved vs MM_Improved     Result: 15 to 5
  Match 5: ID_Improved vs   AB_Null       Result: 17 to 3
  Match 6: ID_Improved vs   AB_Open       Result: 17 to 3
  Match 7: ID_Improved vs AB_Improved     Result: 10 to 1
0

Results:
----------
ID_Improved         77.14%

************************
   Evaluating: Student
************************

Playing Matches:
----------
  Match 1:   Student   vs   Random        Result: 20 to 0
  Match 2:   Student   vs   MM_Null       Result: 19 to 1
```

```
  Match 3:   Student   vs   MM_Open       Result: 19 to 1
  Match 4:   Student   vs MM_Improved     Result: 12 to 8
  Match 5:   Student   vs   AB_Null       Result: 19 to 1
  Match 6:   Student   vs   AB_Open       Result: 13 to 7
  Match 7:   Student   vs AB_Improved     Result: 12 to 8


Results:
----------
Student             81.43%
```

First heuristics ¶

In order to get started, I've implement two variations from the same idea:

- Get as far as possible
- Get as close as possible

The results didn't seem promissing, but was the first kick-off.

Here's the code for `get_as_far_as_possible`:

```
def get_as_far_as_possible(game, player):
    own_moves = game.get_legal_moves(player)
    opp_moves = game.get_legal_moves(game.get_opponent(pl
ayer))

    maximum_distance = -math.inf
```

```
    for a in own_moves:
        for c in opp_moves:
            dist = distance.euclidean(a,c)
            if dist > maximum_distance:
                maximum_distance = dist

    return float(maximum_distance)
```

The result was **58.57%**:

```
**************************
   Evaluating: Student
**************************

Playing Matches:
----------
  Match 1:   Student   vs   Random        Result: 19 to 1
  Match 2:   Student   vs   MM_Null       Result: 12 to 8
  Match 3:   Student   vs   MM_Open       Result: 11 to 9
  Match 4:   Student   vs MM_Improved     Result: 12 to 8
  Match 5:   Student   vs   AB_Null       Result: 11 to 9
  Match 6:   Student   vs   AB_Open       Result: 10 to 1
0
  Match 7:   Student   vs AB_Improved     Result: 7 to 13
```

```
Results:
----------
Student                 58.57%
```

Here's the code for `get_as_close_as_possible`:

```
def get_as_close_as_possible(game, player):
    own_moves = game.get_legal_moves(player)
    opp_moves = game.get_legal_moves(game.get_opponent(pl
ayer))

    minimum_distance = math.inf

    for a in own_moves:
        for c in opp_moves:
            dist = distance.euclidean(a,c)
            if dist < minimum_distance:
                minimum_distance = dist
    #in order to have better scores for min distances, we
 should multiply by (-1)
    return float(-minimum_distance)
```

The results was **62.86%**:

```
*************************
   Evaluating: Student
```

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

Playing Matches:
----------
  Match 1:   Student   vs   Random        Result: 19 to 1
  Match 2:   Student   vs   MM_Null       Result: 16 to 4
  Match 3:   Student   vs   MM_Open       Result: 9 to 11
  Match 4:   Student   vs MM_Improved     Result: 13 to 7
  Match 5:   Student   vs   AB_Null       Result: 12 to 8
  Match 6:   Student   vs   AB_Open       Result: 11 to 9
  Match 7:   Student   vs AB_Improved     Result: 8 to 12

Results:
----------
Student             62.86%
```

Second heuristics ¶

As `get_as_close_as_possible` perfomed a slightly better than other, I've tried to consider blank spaces as well in order to increase the score.

Intuitively I thought increasing blank spaces to final score would perform better, but the results showed that I should decrease blank spaces.

Here's the final code:

```
def get_as_close_as_possible_with_blank_spaces(game, play
er):
    own_moves = game.get_legal_moves(player)
    opp_moves = game.get_legal_moves(game.get_opponent(pl
ayer))
    blank_spaces = len(game.get_blank_spaces())

    minimum_distance = math.inf

    for a in own_moves:
        for c in opp_moves:
            dist = distance.euclidean(a,c)
            if dist < minimum_distance:
                minimum_distance = dist

    return float(-minimum_distance - blank_spaces)
```

The result subtracting `blank_spaces` was **72.86%**:

```
*************************
   Evaluating: Student
*************************

  Match 1:   Student   vs   Random        Result: 19 to 1
  Match 2:   Student   vs   MM_Null       Result: 19 to 1
  Match 3:   Student   vs   MM_Open       Result: 11 to 9
```

```
  Match 4:   Student   vs MM_Improved     Result: 14 to 6
  Match 5:   Student   vs   AB_Null       Result: 15 to 5
  Match 6:   Student   vs   AB_Open       Result: 13 to 7
  Match 7:   Student   vs AB_Improved     Result: 11 to 9


Results:
----------
Student              72.86%
```

The result adding `blank_spaces` was **50%**:

```
*************************
   Evaluating: Student
*************************


  Match 1:   Student   vs   Random        Result: 17 to 3
  Match 2:   Student   vs   MM_Null       Result: 11 to 9
  Match 3:   Student   vs   MM_Open       Result: 9 to 11
  Match 4:   Student   vs MM_Improved     Result: 7 to 13
  Match 5:   Student   vs   AB_Null       Result: 9 to 11
  Match 6:   Student   vs   AB_Open       Result: 9 to 11
  Match 7:   Student   vs AB_Improved     Result: 8 to 12


Results:
----------
Student              50.00%
```

Third heuristics ¶

Since combining “get close” and “blank spaces” give me some results, I’ve added weight 2 to blank results, which started to give interesting results.

Here’s the code:

```
def get_as_close_as_possible_with_blank_spaces_and_weight
s(game, player):
    own_moves = game.get_legal_moves(player)
    opp_moves = game.get_legal_moves(game.get_opponent(pl
ayer))
    blank_spaces = len(game.get_blank_spaces())

    minimum_distance = math.inf

    for a in own_moves:
        for c in opp_moves:
            dist = distance.euclidean(a,c)
            if dist < minimum_distance:
                minimum_distance = dist

    return float(-minimum_distance - (2*blank_spaces))
```

The results was **85.71%**:

```
Match 1:   Student   vs   Random        Result: 20 to 0
  Match 2:   Student   vs   MM_Null       Result: 16 to 4
  Match 3:   Student   vs   MM_Open       Result: 19 to 1
  Match 4:   Student   vs MM_Improved     Result: 16 to 4
  Match 5:   Student   vs   AB_Null       Result: 19 to 1
  Match 6:   Student   vs   AB_Open       Result: 15 to 5
  Match 7:   Student   vs AB_Improved     Result: 15 to 5


Results:
----------
Student             85.71%
```

Changing the weight to **10** have the result of **75.00%**:

```
*************************
   Evaluating: Student
*************************


Playing Matches:
----------
  Match 1:   Student   vs   Random        Result: 16 to 4
  Match 2:   Student   vs   MM_Null       Result: 16 to 4
  Match 3:   Student   vs   MM_Open       Result: 14 to 6
  Match 4:   Student   vs MM_Improved     Result: 15 to 5
  Match 5:   Student   vs   AB_Null       Result: 17 to 3
  Match 6:   Student   vs   AB_Open       Result: 14 to 6
  Match 7:   Student   vs AB_Improved     Result: 13 to 7
```

```
Results:
----------
Student             75.00%
```

Changing the weight to **5** have the result of **75.71%**:

```
*************************
   Evaluating: Student
*************************

Playing Matches:
----------
  Match 1:   Student   vs   Random        Result: 19 to 1
  Match 2:   Student   vs   MM_Null       Result: 19 to 1
  Match 3:   Student   vs   MM_Open       Result: 15 to 5
  Match 4:   Student   vs MM_Improved     Result: 13 to 7
  Match 5:   Student   vs   AB_Null       Result: 16 to 4
  Match 6:   Student   vs   AB_Open       Result: 10 to 1
0
  Match 7:   Student   vs AB_Improved     Result: 14 to 6

Results:
----------
Student             75.71%
```

Since it didn’t improved the first result, I’ve decided to remain 2 as a

final weight.

In order to have one more test, I've added a weight of 2 to `minimum_distance` as well. Here's the final return:

```
return float(-(2*minimum_distance) - (2*blank_spaces))
```

The result was **75.00%**:

```
*************************
   Evaluating: Student
*************************

Playing Matches:
----------
  Match 1:   Student   vs   Random        Result: 19 to 1
  Match 2:   Student   vs   MM_Null       Result: 15 to 5
  Match 3:   Student   vs   MM_Open       Result: 14 to 6
  Match 4:   Student   vs MM_Improved     Result: 14 to 6
  Match 5:   Student   vs   AB_Null       Result: 16 to 4
  Match 6:   Student   vs   AB_Open       Result: 13 to 7
  Match 7:   Student   vs AB_Improved     Result: 14 to 6

Results:
----------
Student             75.00%
```

Conclusion and results ¶

My approach was based in getting closer to or further from the opponent. Besides, I've tried to consider remaining blank spaces to have better scores.

Here's the results:

Baseline	Getting further	Getting close to	Getting close to + blank spaces	Getting close to + weightned blank spaces
81.43%	58.57%	62.86%	72.86%	**85.71%**

More results ¶

As discussed in this post, in order to have more accurate results, I've changed `NUM_MATCHES` to 20 and here's the results:

- ID_Improved: 76.61%
- Student: 76.96%

It got **0.456%** better than ID_improved :

```
*************************
 Evaluating: ID_Improved
*************************
```

```
Playing Matches:
----------
  Match 1: ID_Improved vs   Random        Result: 78 to 2
  Match 2: ID_Improved vs   MM_Null       Result: 76 to 4
  Match 3: ID_Improved vs   MM_Open       Result: 58 to 2
2
  Match 4: ID_Improved vs MM_Improved     Result: 57 to 2
3
  Match 5: ID_Improved vs   AB_Null       Result: 64 to 1
6
  Match 6: ID_Improved vs   AB_Open       Result: 47 to 3
3
  Match 7: ID_Improved vs AB_Improved     Result: 49 to 3
1


Results:
----------
ID_Improved         76.61%


*************************
   Evaluating: Student
*************************


Playing Matches:
----------
  Match 1:   Student   vs   Random        Result: 78 to 2
  Match 2:   Student   vs   MM_Null       Result: 73 to 7
  Match 3:   Student   vs   MM_Open       Result: 61 to 1
```

```
9
  Match 4:   Student   vs MM_Improved     Result: 54 to 2
6
  Match 5:   Student   vs   AB_Null       Result: 64 to 1
6
  Match 6:   Student   vs   AB_Open       Result: 54 to 2
6
  Match 7:   Student   vs AB_Improved     Result: 47 to 3
3


Results:
----------
Student             76.96%
```

So, the best tested heuristic was the `get_as_close_as_possible_with_blank_spa` method with 2 as weight. ¶

I've decided to chose this evaluation function because:

- It's simple to implement and fast to run. Each for loop takes less time mainly because everytime the number of `legal_moves` decreases.
- In my hardware, this helped to never have a timeout.
- Its results were consistent and as discussed above, it beated the ID_Improved in 0.456% by playing 20 matches
- Intuitively by playing games, I thought it would be better if I increase blank spaces, but data showed that I should decrease

in order to get better results