Isolation Game Heuristic Analysis

By Yutao Liu

For the project, I experimented different heuristic functions. Compared with limit time and calculation resources, the simpler function allows the search to proceed deeper in the game tree, at first I have tried to implement Open Score and Center Score, but they always perform not very well. I think they only contain very few information about the game. So I tried some complex function to get better result.

Here is a summary of these complex results:
I've compared 4 different heuristic functions, and the first two functions are modified from the improved_score() and center_score.
The last two functions merged the first two functions in different ways.

Customer Player	Opponent Player's Win Rate	Average player's Win Rate
Heuristic 1	61.4%	58.6%
Heuristic 2	60.0%	60.96%
Heuristic 3	61.4%	62.3%
Heuristic 4	55.7%	60.46%

Heuristic 1:
With this heuristic, I got inspiration from improving_score (). Where should the customer_player do when the two players have the common move. I think if the player and the opponent player have the common move, its better to choose the common move and make the opponent player have less move to go.
So I use the length of the player's moves minus the length of opponent's moves and then add the length of common moves.
Results:

```
*************************
    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	9	1	10	0	9	1
2	MM_Open	6	4	5	5	4	6	7	3
3	MM_Center	6	4	7	3	8	2	7	3
4	MM_Improved	6	4	7	3	4	6	6	4
5	AB_Open	4	6	4	6	4	6	4	6
6	AB_Center	5	5	5	5	3	7	4	6
7	AB_Improved	6	4	7	3	4	6	5	5
	Win Rate:	61.4%		62.9%		52.9%		60.0%	

Implementation:
Here is the implementation of the heuristic function:

```python
def custom_score(game, player):
    # TODO: finish this function!
    if game.is_loser(player):
        return float("-inf")

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

    player_moves = game.get_legal_moves()

    opponent_moves = game.get_legal_moves(game.get_opponent(player))

    common_moves = [i for i in player_moves for j in opponent_moves if i == j]
    return float(len(player_moves) - len(opponent_moves)+ len(common_moves))
```

Analysis:

Not a great heuristic function and not performs well every time. I think its not a good choice to occupy the opponent's possible move at first but maybe its useful in the last few moves.

- Compared with the following heuristics, its very easy to compute and search deeper.
- According to the result, it wins once but lose twice, and average win rate is less than the AB_improved heuristic.

Heuristic2:

With this heuristic, I got inspiration from center_score (). I think maybe its better to occupy the center location first then the player may have more possible move to go. So I use the square of the distance between center and the opponent's location minus the square of the distance between center and the player's location.

Results:

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

Implementation:

Here is the implementation of the heuristic function:

```python
def custom_score(game, player):
    # TODO: finish this function!
    if game.is_loser(player):
        return float("-inf")

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

    w, h = game.width / 2., game.height / 2.
    y, x = game.get_player_location(player)
    y1, x1 = game.get_player_location(game.get_opponent(player))
    return float((h - y1)**2 + (w - x1)**2-(h - y)**2 - (w - x)**2)
```

Analysis:

The heuristic preforms a bit better than AB_improved. It seems like that occupy the center location first is a good strategy.

- Its a little more complex than the first heuristic, because it has some multiplication and square algorithms.
- But now it wins AB_improved once and lose once, it's average win rate is a little higher than AB_improved.

Heuristic3:

With this heuristic, I just want to merge the heuristic1 and heuristic2. Maybe this time could perform better.

So I use the first heuristic score plus the second heuristic score.

Results:

****
Playing Matches
****

Match #	Opponent	AB_Improved Won	AB_Improved Lost	AB_Custom Won	AB_Custom Lost	AB_Custom_2 Won	AB_Custom_2 Lost	AB_Custom_3 Won	AB_Custom_3 Lost
1	Random	9	1	10	0	9	1	10	0
2	MM_Open	7	3	5	5	5	5	7	3
3	MM_Center	8	2	8	2	8	2	9	1
4	MM_Improved	4	6	6	4	2	8	7	3
5	AB_Open	4	6	4	6	6	4	5	5
6	AB_Center	5	5	5	5	5	5	6	4
7	AB_Improved	6	4	5	5	5	5	5	5
	Win Rate:	61.4%		61.4%		57.1%		70.0%	

Implementation:

Here is the implementation of the heuristic function:

```python
def custom_score(game, player):
    # TODO: finish this function!
    if game.is_loser(player):
        return float("-inf")

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

    player_moves = game.get_legal_moves()
    opponent_moves = game.get_legal_moves(game.get_opponent(player))
    common_moves = [i for i in player_moves for j in opponent_moves if i == j]

    w, h = game.width / 2., game.height / 2.
    y, x = game.get_player_location(player)
    y1, x1 = game.get_player_location(game.get_opponent(player))
    return float((h - y1)**2 + (w - x1)**2-(h - y)**2 - (w - x)**2) + \
           float(len(player_moves) - len(opponent_moves) + len(common_moves))
```

Analysis:

This is a good strategy.But it doesn’t work very well all the time, because it is complex.

- It is a better strategy but cost too much calculate resource.
- In this heuristic function, we win the AB_improved player once and achieve 70% win rate.

Heuristic4:

With this heuristic, I think the first heuristic function can perform well in the last few moves, and the second heuristic can perform better in the beginning of the game.
So the the heuristic function’s idea is when there is more than 3 moves to occupy, we choose the heuristic2 function, if there is less than 3 moves to occupy, we choose the heuristic1 function.

Results:

```
*************************
     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	8	2	9	1	9	1
2	MM_Open	5	5	4	6	6	4	7	3
3	MM_Center	8	2	8	2	9	1	8	2
4	MM_Improved	5	5	4	6	5	5	4	6
5	AB_Open	4	6	5	5	4	6	6	4
6	AB_Center	4	6	6	4	5	5	3	7
7	AB_Improved	5	5	6	4	5	5	6	4
	Win Rate:	55.7%		58.6%		61.4%		61.4%	

Implementation:

Here is the implementation of the heuristic function:

```python
def custom_score(game, player):
    # TODO: finish this function!
    if game.is_loser(player):
        return float("-inf")

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

    player_moves = game.get_legal_moves()
    opponent_moves = game.get_legal_moves(game.get_opponent(player))
    common_moves = [i for i in player_moves for j in opponent_moves if i == j]

    w, h = game.width / 2., game.height / 2.
    y, x = game.get_player_location(player)
    y1, x1 = game.get_player_location(game.get_opponent(player))
    return float((h - y1)**2 + (w - x1)**2-(h - y)**2 - (w - x)**2) + \
            float(len(player_moves) - len(opponent_moves) + len(common_moves))
```

Analysis:

With different strategy in different state, performs better than the other heuristic function.

- A efficient strategy, which can go deeper than the third heuristic function.
- In this heuristic function, we already won the AB_improved player 3 times.