Heuristic Analysis

tournament.py modifications

The file `tournament2.py` is copy of `tournament.py`, and is changed in following ways:

- Increase NUM_MATCHES from 5 to 25, so more matches will be done.
- Remove `ID_Improved` from test_agents, since `ID_Improved` is only needed to run once to get it's performance value.

Heuristic functions

ID_Improved (given)

`ID_Improved` calculate the number of available move of 2 players, and minus them to get the score.

The `tournament2.py` result is as follow:

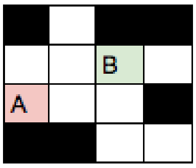
```
*************************
 Evaluating: ID_Improved
*************************

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 93 to 7
  Match 2: ID_Improved vs   MM_Null     Result: 77 to 23
  Match 3: ID_Improved vs   MM_Open     Result: 54 to 46
  Match 4: ID_Improved vs MM_Improved   Result: 44 to 56
  Match 5: ID_Improved vs   AB_Null     Result: 57 to 43
  Match 6: ID_Improved vs   AB_Open     Result: 34 to 66
  Match 7: ID_Improved vs AB_Improved   Result: 50 to 50


Results:
----------
ID_Improved         58.43%
```

	Random	MM_Null	MM_Open	MM_Improved	AB_Null	AB_Open	AB_Improved	Result
ID_Improved	93	77	54	44	57	34	50	58.43%

custom_score_0 (Improved ID_Improved)

The `custom_score_0` put n moves forward into consideration.

For example, in a 4x4 game:



We first do breadth first search to find number of move required to go to each cell, in empty board.

Since the calculation is based on empty board, the result can be cached to save CPU.

2	1	4	3
3	2	1	2
0	3	2	3
3	2	1	2

1	2	3	4
2	3	0	3
1	2	3	2
4	1	2	1

Then, for each cell which require n moves, we score it r_0^n.

For r_0 = 1/8:

0.0156	0.1250	0.0002	0.0020
0.0020	0.0156	0.1250	0.0156
1.0000	0.0020	0.0156	0.0020
0.0020	0.0156	0.1250	0.0156

0.1250	0.0156	0.0020	0.0002
0.0156	0.0020	1.0000	0.0020
0.1250	0.0156	0.0020	0.0156
0.0002	0.1250	0.0156	0.1250

Finally, we sum up all cell which is not blocked, and subtract each other.

- Player A: value = 0.316
- Player B: value = 0.193
- score = 0.316 - 0.193 = 0.123

The `tournament2.py` result is as follow:

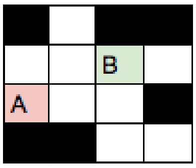
r_0	Random	MM_Null	MM_Open	MM_Improved	AB_Null	AB_Open	AB_Improved	Result
1/2	95	86	57	47	75	50	56	66.57%
1/3	94	80	44	50	70	60	42	62.86%
1/4	93	82	51	53	65	51	46	63.00%
1/5	95	88	54	41	72	58	61	67.00%
1/6	86	83	62	54	72	58	58	67.57%
1/7	91	83	48	53	73	54	53	65.00%
1/8	98	79	51	47	77	55	51	65.43%
1/9	94	82	54	52	72	53	56	66.14%
1/10	92	77	54	57	68	56	57	65.86%

It is sightly better than `ID_Improved`. The differences between each r values are too small to make any conclusion.

custom_score_1 (Neural network)

The `custom_score_1` is based on neural network and minimax Q learning.

First, we convert the 7x7 game data into 7x7x3 = 147 boolean value. The first 7x7 boolean value represent which cell is not blocked. The second 7x7 boolean value represent the location of active player. The third 7x7 boolean value represent the location of inactive player.



becomes

0	1	0	0
1	1	0	1
0	1	1	0
0	0	1	1

0	0	0	0
0	0	0	0
1	0	0	0
0	0	0	0

0	0	0	0
0	0	1	0
0	0	0	0
0	0	0	0

Then we put the values into 3 hidden layers neural network, which represent the score of 8 move. We apply boolean mask to filter out impossible move.

The score of state s would be:

```
max( Q(s0,a0) for all a0 )
```

The neural network is trained by following equation:

```
Q(s0,a0) = -1                                   if loss
         = +1                                   if win
         = - gamma * max( Q(s1,a1) for all a1 ) otherwise
```

Since the Q function return the score of the active player, the right hand side of the equation should be negative, since the second move is made by opponent.

Step of training:

1. Make 100000 moves
2. Train upon 1-100000th move
3. Make 100001st move
4. Train upon 2-100001th move
5. Make 100002st move
6. Train upon 3-100002th move
7. continue...

The `tournament2.py` result is as follow:

moves	Random	MM_Null	MM_Open	MM_Improved	AB_Null	AB_Open	AB_Improved	Result
200000	86	57	36	37	58	31	36	48.71%
300000	87	72	44	36	59	37	44	54.14%
400000	86	70	32	38	54	37	36	50.43%
500000	85	69	35	34	62	40	44	52.71%
600000	94	67	34	28	56	37	49	52.14%
700000	89	71	40	29	53	32	43	51.00%

The result is disappointing. Here are possible reasons:

- Not enough sample
- The neural network is too simple

Possible improvement:

- Increase the number of sample window size and sample number. Require more training time.
- Increase the complexity of neural network. Apply convolution layer, add more feature to input data set. Require more CPU / GPU time.

Moreover, the trained neural network can be used only in game same as training game. For games which have difference size, it is necessary to train another neural network.

The `custom_score_1` require TensorFlow to run.

custom_score_2 (Distance from center)

The output is the distance between player location to the center. Since we are not sure it is better to stay at center or edge, so we make 2 opposite functions.

- version 2a: Self distance - Opponent distance. Stay on edge will get higher score.
- version 2b: Opponent distance - Self distance. Stay in center will get higher score.

	Random	MM_Null	MM_Open	MM_Improved	AB_Null	AB_Open	AB_Improved	Result
2a	86	38	16	18	25	22	17	31.71%
2b	82	71	37	37	54	45	42	52.57%

Even though the result is disappointing, the function is too simple and it achieve 52.57%. Moreover the difference of 2a and 2b is high, we may assume that it is better to be in center. The discovery can help to develop other score function.

custom_score_3 (custom_score_0 + custom_score_2)

Since we know `custom_score_0` and `custom_score_2b` are good heuristic functions, so we combine those functions to make a new function.

```
custom_score_3b = custom_score_0(r=1/6) + r3 * custom_score_2b
```

r_{3b}	Random	MM_Null	MM_Open	MM_Improved	AB_Null	AB_Open	AB_Improved	Result
0	86	83	62	54	72	58	58	67.57%
0.02	93	79	48	53	72	55	55	65.00%
0.04	92	81	50	51	65	56	51	63.71%
0.06	92	77	50	57	66	55	46	63.29%
0.08	90	75	56	56	64	50	35	60.86%
0.10	94	74	56	54	74	48	54	64.86%
0.12	95	76	56	58	69	51	45	64.29%
0.14	90	73	58	49	68	55	44	62.43%
0.16	92	76	47	37	67	51	47	59.57%
0.18	96	73	50	51	66	55	51	63.14%
0.20	96	83	50	45	64	49	45	61.71%

It seems `custom_score_2b` bring negative effect to `custom_score_0`.

Here is the test to combine `custom_score_0` and `custom_score_2a`

```
custom_score_3a = custom_score_0(r=1/6) + r3 * custom_score_2a
```

r_{3a}	Random	MM_Null	MM_Open	MM_Improved	AB_Null	AB_Open	AB_Improved	Result
0	86	83	62	54	72	58	58	67.57%
0.02	90	77	53	54	61	52	44	61.57%
0.04	93	77	56	54	76	53	48	65.29%
0.06	93	80	57	56	67	54	53	65.71%
0.08	98	81	60	51	73	57	51	67.29%
0.10	90	76	56	50	67	49	45	61.86%
0.12	95	83	46	47	74	53	46	63.43%
0.14	91	91	55	46	60	47	44	62.00%
0.16	93	86	50	48	71	52	45	63.57%
0.18	95	75	50	43	76	45	41	60.71%
0.20	97	80	51	40	78	49	45	62.86%

No improvement found also.

custom_score_4 (Simulation)

`custom_score_4` run simulation play to end game, and output the win/loss of the result.

In simulation, the game will choose the move closest to the board center. We choose this feature since it is less CPU consumption and effective.

For the game which win in n step, the function would output r_4^n ($0 < r_4 < 1$). For the game which loss in n step, the function would output $-r_4^n$. So the AI would prefer early win than later win, and prefer later loss than early loss.

The runtime and performance of this function depends on number of blank cell in the board. With more blank cell, simulation require more time, and the result is less accurate. As step count increase, $+/-r_4^n$ would be close to zero. So we may put a cutdown to save CPU. But now we just simulate to endgame.

It is meaningless to tune r_4. We just need to ensure $0 < r_4 < 1$.

type	Random	MM_Null	MM_Open	MM_Improved	AB_Null	AB_Open	AB_Improved	Result
cs4	95	81	47	51	71	53	52	64.29%

It's strength is close to `ID_Improved`, and weaker than `custom_score_0`, even though it consume high CPU.

custom_score_5 (custom_score_0 + custom_score_4)

`custom_score_5` combine `custom_score_0` and `custom_score_4` by adding them together.

`custom_score_5 = (1-r5) * custom_score_0 + r5 * custom_score_4`

To get the best performance of the function, we need to tune r_0, r_4 and r_5 together. For simplicity we take $r_0 = 1/6$, $r_4 = 0.99$, and tune r_5.

Result: ($r_0 = 1/6$, $r_4 = 0.99$, NUM_MATCHES = 250)

r_5	Random	MM_Null	MM_Open	MM_Improved	AB_Null	AB_Open	AB_Improved	Result
0.1	968	868	652	627	765	619	603	72.89%
0.2	958	879	672	613	790	645	606	73.76%
0.3	974	905	671	669	808	625	617	75.27%
0.4	964	886	674	627	787	630	587	73.64%
0.5	965	877	668	602	794	611	619	73.37%
0.6	958	890	686	630	782	637	609	74.17%
0.7	965	867	681	620	820	617	599	73.84%
0.8	958	887	647	590	801	642	592	73.10%
0.9	968	894	644	614	795	633	589	73.39%

The performance of `custom_score_5` is far better than `custom_score_0` and `custom_score_4`. r_0, r_4, and r_5 can be further tuned to achieve better performance.

Conclusion

Here is the summary of the heuristic functions:

type	Random	MM_Null	MM_Open	MM_Imp	AB_Null	AB_Open	AB_Imp	Result
ID_Improved	93	77	54	44	57	34	50	58.43%
custom_0	86	83	62	54	72	58	58	67.57%
custom_1	89	71	40	29	53	32	43	51.00%
custom_2b	82	71	37	37	54	45	42	52.57%
custom_3b	93	79	48	53	72	55	55	65.00%
custom_4	95	81	47	51	71	53	52	64.29%
custom_5	97.4	90.5	67.1	66.9	80.8	62.5	61.7	75.27%

We recommend `custom_score_5`:

- It achieve best result when matching with random opponent, with chance 97.4%.
- It achieve best result when matching with intelligent opponents such as `open_move_score` and `improved_score`, with higher than 60% chance.
- It's performance is the best among all other custom score.

In CPU limited environment, we recommend `custom_score_0`, as it can run in constant time in larger board.

In extreme limited environment, `custom_score_2b` would be a choice as it does not require much calculation.

tournament.py output

```
This script evaluates the performance of the custom heuristic function by
comparing the strength of an agent using iterative deepening (ID) search with
alpha-beta pruning against the strength rating of agents using other heuristic
functions.  The `ID_Improved` agent provides a baseline by measuring the
performance of a basic agent using Iterative Deepening and the "improved"
heuristic (from lecture) on your hardware.  The `Student` agent then measures
the performance of Iterative Deepening and the custom heuristic against the
same opponents.


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

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 18 to 2
  Match 2: ID_Improved vs   MM_Null     Result: 14 to 6
  Match 3: ID_Improved vs   MM_Open     Result: 7 to 13
  Match 4: ID_Improved vs MM_Improved   Result: 10 to 10
  Match 5: ID_Improved vs   AB_Null     Result: 12 to 8
  Match 6: ID_Improved vs   AB_Open     Result: 9 to 11
  Match 7: ID_Improved vs AB_Improved   Result: 11 to 9


Results:
----------
ID_Improved         57.86%

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

Playing Matches:
----------
  Match 1:   Student   vs   Random      Result: 19 to 1
  Match 2:   Student   vs   MM_Null     Result: 17 to 3
  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: 17 to 3
  Match 6:   Student   vs   AB_Open     Result: 15 to 5
  Match 7:   Student   vs AB_Improved   Result: 13 to 7


Results:
----------
Student             74.29%
```