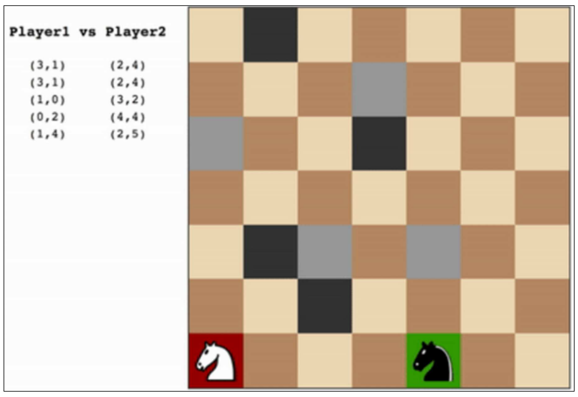
Udacity Artificial Intelligence Nanodegree
Project – Building a Game-Playing Agent
Heuristic Analysis
Kin Ming (Frank) Puk


```
Player1 vs Player2

  (3,1)      (2,4)
  (3,1)      (2,4)
  (1,0)      (3,2)
  (0,2)      (4,4)
  (1,4)      (2,5)
```

The aim of this project is to develop a game agent to play the isolation game against AI players with different design of tree-searching algorithms and evaluation function of game states. Each game state is regarded as a node, and the possibility after making a move can be expanded as a sub-tree. Therefore, each state of the game can indeed be presented as a tree, with the beginning node as the start of the game and end node being the result of the game. This project focuses on two tree-searching algorithms in developing the game-playing agent – minimax and alpha beta pruning algorithms.

There are two parts in this project – 1) implementation of minimax and alpha beta pruning algorithms and 2) design of evaluation function. According to course textbook [1] and textbook github [2,3], the pseudo code for minimax and alpha beta pruning algorithms is as follows:

Minimax

```
function MINIMAX-DECISION(state) returns an action
    return arg max a ∈ ACTIONS(s) MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) the return UTILITY(state)
    v ← −∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(state, a)))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) the return UTILITY(state)
    v ← ∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(state, a)))
    return v
```

Alpha beta pruning

```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state, −∞, +∞)
    return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) the return UTILITY(state)
    v ← −∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(state, a), α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v

function MIN-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) the return UTILITY(state)
    v ← +∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(state, a), α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

The above pseudo code is included for ease of reference. The actual python code can be found in the submission package.

As for the second part, evaluation function is used in this project because it is a good way to quantify or approximate the true utility of a game state without doing a complete search of the entire game tree. They are designed according to the opponent design in “tournament.py”. The three heuristic functions are as follows

Evaluation Function	Definition
custom_score	$m - 2o$
custom_score_2	$m - do$, where $d = b/a$ * constant[1]
custom_score_3	$m * d_{myself} - o * d_{opponent}$

Meaning of variables – number of my moves is m, number of opponent moves is o, number of blank spaces is b, area of game board is a, Euclidean distance of myself from center is d_{myself} and Euclidean distance of opponent from center is $d_{opponent}$.

[1] The constant is tuned to be 3. Win rate for different values of constant is: 75.7% for 1, 81.4% for 2, 85.7% for 3, 85.7% for 4 and 84.3% for 5.

Udacity Artificial Intelligence Nanodegree
Project – Building a Game-Playing Agent
Heuristic Analysis
Kin Ming (Frank) Puk

Heuristic 1: The first heuristic is the evaluation function as stated in the class. This is chosen because it is a good design, is stable across different kinds of game opponents, and can be used as a baseline for comparing heuristic 2 and 3.

Heuristic 2: The second heuristic is similar to heuristic 1. The factor 2 for number of opponents can indeed be regarded as a decay factor and be changed as the game continues. It would make more sense if such decay factor changes accordingly throughout the game, which is similar to the learning rate as tuned in each iteration of optimization algorithms, if any. With such a design, the game will be played more aggressively in earlier moves because the decay factor will be larger and more conservatively as the game proceeds for the same rationale.

Heuristic 3: The third heuristic is inspired from the game opponents (AB_Center, MM_Center) which start at the center of the board. Such evaluation function encourages moves towards the center.

The following shows the performance of different tree-searching algorithms and evaluation functions:

Win rate in 5 matches

Match #	Opponent	AB_Improved		AB_Custom		AB_Custom2		AB_Custom3	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	7	3	10	0	10	0	10	0
2	MM_Open	8	2	8	2	6	4	9	1
3	MM_Center	7	3	10	0	9	1	10	0
4	MM_Improved	4	6	5	5	8	2	7	3
5	AB_Open	5	5	7	3	9	1	9	1
6	AB_Center	7	3	9	1	10	0	10	0
7	AB_Improved	6	4	7	3	8	2	8	2
	Win Rate:	62.9%		80.0%		85.7%		90.0%	

Among the evaluation function, the last one ("AB_Custom3") is the best and recommended for further use and research because of the following:

1) The difference between "AB_Custom3" and "AB_Improved" is significant (27.1%);
2) Performance of "AB_Custom3" is stable across all opponents including the most difficult player "MM_Improved";
3) The rationale behind "AB_Custom3" that encourages the moves to be placed around the center of the board as soon as possible links well to the core strategy of playing the isolation game; and
4) The evaluation function does not require tuning, unlike "AB_Custom2". It is thus ready for deployment and reduces computational time.

Future work includes combining "AB_Custom2" and "AB_Custom3" so that the evaluation function will be more robust.

Reference:
[1] Russell S, Norving P. Artificial Intelligence. 3rd ed. New Jersey: Pearson; 2010.
[2] https://github.com/aimacode/aima-pseudocode/blob/master/md/Minimax-Decision.md
[3] https://github.com/aimacode/aima-pseudocode/blob/master/md/Alpha-Beta-Search.md