Go-Moku and A Heuristic search

1st Ziyang Wei
Electronics and Computer Science
University of Southampton
Southampton, the United Kingdom
zw1y18@soton.ac.uk

***Abstract*—This article will explore the mechanism of the Gomoku's AI, and demonstrate how it finds the optimal solution based on millions of possible nodes. The project has been displayed on the website server and can be access from http://106.15.227.178,since the server has a limited CPU, so search depth be set to the 4 depth, but it's still challenging for ordinary player.**

***Index Terms*—GoMoku, Evaluate function, MiniMax Search, Alpha-Beta pruning, Heuristic Search**

I. Introduction

Although the Gomoku has been solved in 1992 by Allis with the threat space search since the black stone(first player) has more advantages than white stone(second player) [1]., it is still challenging for the machine to win the player when the rules has been changed. Besides, it is a decent example to demonstrate the searching algorithm behind the game since the number of possible nodes expanded in each layer is large enough and it is not easy to solve it in an violent way(Tic tac toe is extremely easy to get a draw without any optimization).

A. Rule

The rules of Go-Moku is easy to be understood. There are 225 stones distributed on a 15 * 15 board.Two players, black and white, put their stones on the board in turn, and the black goes first normally in the center of the board. The player who puts a consecutive five stones horizontally, vertically, diagonally) will win the game. If a player has put his stone one the board, it has been locked and can't be moved nor can be captured. If the board is filled with stones, the Go-moku game will be drawn.To simplify this game, any variants are depreciated in the rules, which means the black stone can be more aggressive and has more advantages than the white.

B. Definations

In this section, I will give some definitions to some threat sequences which will be widely used in the later section.

As the Diagram.1 shows, the threats has been divided into several types, each type has its own threaten value that represents how danger it is.

- The "Five is defined as a sequence that five stones have been placed in the same line, which means that the attacker has won, and the game has ended.The sequenced has been marked with "A" in the Diagram. 1 .
- The "Direct Four" is defined as a situation where there are four stones lying on a line and the two position square near it is empty.When this threat has shown on the board, the attacker will be unstoppable and is about to win in the next turn. "B" has demonstrated this threat clearly.
- The "Dead Four" is a situation where four squares has been occupied in a line while the end or the front has been blocked,which means that there is still a chance to defense the attack from the enemy.The "E" situation has demonstrated it below.
- The "Direct Three" is the most common threat in this game.It denotes a case where three stones of the same colour are in an unbroken row, or with one-intersection gap between the stones that is about to become a straight four on the next move.The "C" has marked this case.
- Compared with the "Direct Three", "Dead Three" is much less aggressive, which is a case when the "Direct Three" has been blocked by the opponent. Even though the player let it go, it still can only form a "Dead Four" in the next round. This case has been marked by "F" in the graph below.
- The "Two" is the least dangerous among these threats, in which a stone lies close to another same stone and there is no enemy adjacent to them in a same line.

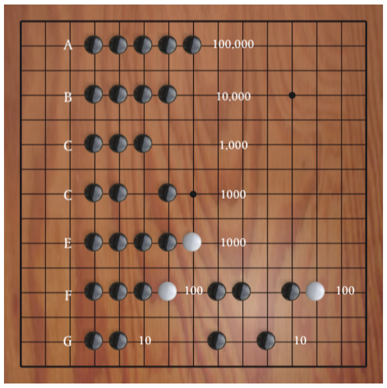

Fig. 1. Typical Threats

C. Application

The Application is developed in a Browser-Server architecture, and has been deployed on the http://106.15.227.178.The search depth has been set to 4ply(Note that since the server I brought has the lowest configuration,it may lead to memory leak). Obviously, The front-end part is written in Javascript and the back-end part is written in Python and C.The interface between front-end and computing part is coded in Python based on the Flask, while the core code is written in C programming language to guarantee it has a good performance in computing speed. and I will compare the difference in running speed between the C extension and without the extension.

D. *Method Overview*

The board can be represented as a nested list in Python or a 2-D array in C. the top-left one is represented as (0,0) and the bottom-right one is represented as (14,14).And the black stone is represented as the integer 1, and we uses 2 for the white.

The robot uses evaluation function to assess the current situation, and the Minimax algorithm gives the robot a long-term eyesight.Alpha-beta pruning used to reduce the number of nodes that it need to traverse and it has been optimized by the heuristic function.

II. Approach

A. *Extend successors*

When the opponent places his stone on the board, the robot should consider every possible empty position that can be placed.But it's impossible to consider every empty position on the board, so here we need to define a function to generate all the possible position.So basic idea of it is consider every neighbour of the current occupied square.The definition of neighbour is the distance between two position less or equal than 2.

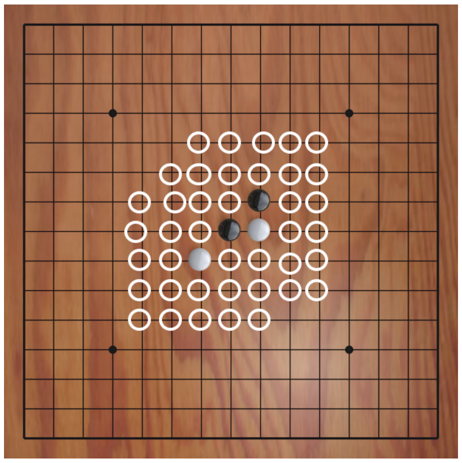
Fig. 2. Extend successors

Like the Figure.2 shown above, the open circles are those that need to be considered. There are approximately 50 successors needs to be considered in each turn.

B. *Evaluation Function*

It is not exaggerating to say that the evaluation function is the most important part in this game, since it's the foundation of all the methods that will be mentioned later. This function is used to evaluate who is more advantageous based on the current situation.It will traverse every stones on the board and returns its overall score. The more score it has, the more advantage it owns.So if the evaluation function is good enough, it will have a good performance even we don't apply other methods.In other words, while if the evaluation function is set wrongly at the beginning, no matter how good the approach it takes later,the robot will still perform very unsatisfactory.

TABLE I
Threat Mark Table

Threats	Mark
Five	100,000
Direct Four	10,000
Direct Three	1,000
Dead Four	1,000
Dead Three	100
Two	10

The evaluation function starts with traversing every position in the board. If the square is occupied by the robot, the programme will check if it is a threat or not in four direction(as the Figure.3.If it is, it will added the corresponding mark.Besides, if the threat is caused by the enemy, then the mark will be deducted by the corresponding mark.The codes in appendix has shown the detail.

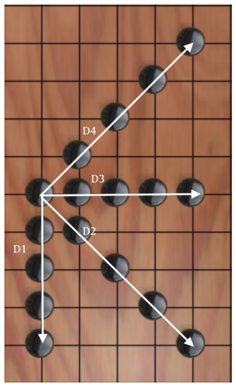

Fig. 3. Check if it is a threat

C. *Minimax Algorithm*

The reason why we need Minimax algorithm is easy to explain.Suppose a situation like the Figure.4 shows below, where it is the turn of white to put stone. For the position of $(4, 8)$, it can create two "Direct Three" for the white, although there is also exist a "Direct Three" lies below , it can still get 1000 scores in the end.While if the white square put his stone in the position of $(6, 6)$, it can only can 200 scores. And it is obviously that if the white stone does not put his stone in the position of $(6, 6)$, it will lose the game immediately. By the way, before the minimax algorithm is implemented, I set a ”defensive ratio” to decrease the score when the robot is being threaten. Although this method can solve this problem in a tricky way, it has an obviously drawback that the AI will almost only focus on the defense and has no intention to make an aggressive move. To solve this problem, an algorithm considers future situation should be implemented.

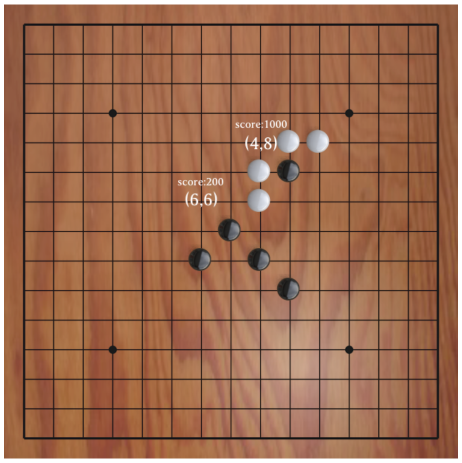

Fig. 4. A case where robot is shortsighted

Minimax algorithm is a kind of depth-first tree search that simulates the process of the two sides playing alternately.The ”Max” function can be viewed the AI, it will choose the best move based on the current situation, while the ”Min” function is the opponent, who will choose the lowest score based on the situation.With the minimax algorithm, the AI will have a long-term eyesight and make a better decision and the more depth it reaches, the more intelligent it is. [3]The process of Minimax can be visualized as the Figure.minimax shows below.Since the number of nodes expanded in each layer is very large and wide, so I cannot display the process of Minimax in this game.Circles represent the robot who wants to maximize his score, and squares represent the moves of the opponent (the player). Inside the circle is the value of the node.In this case, it is the score calculated from the evaluation function.The choice they made has been marked with the red arrow.

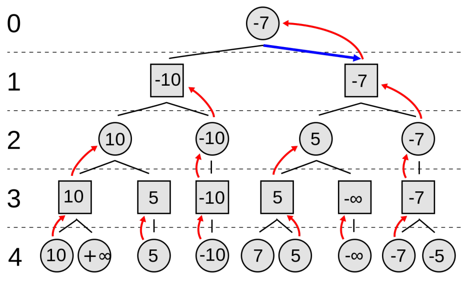

Fig. 5. The process of Minimax algorithm (source:http://en.wikipedia.org/wiki/Image:Minimax.svg)

D. Alpha-beta pruning

Suppose that in each layer, there will be 40 possible nodes to explore, and the time complexity for the tree search is $O(b^d)$, which means if the AI wants to do search for 4ply, then it needs to traverse 2560000 nodes. For my own Mac, the processor is 2.2 GHz Intel Core i7 that is able to compute approximately 10000 nodes within a second, it needs takes 4 minute to finish this search, which is unacceptable for the game.As for the alpha-beta pruning, when it searches trees, It stops immediately when a stone has been found that proves the move to be worse than a previously visited move. [4]

The process of alpha-beta pruning has been illustrated below.The node ”8” that lies in the third layer should be pruned since it has find the node values 5. Because it is the ”min” node, it will only choose the minimal value of all the successors, so when it find the ”5”, it will only yield a result that is less than it.But the parent node that wants to maximize his value has found the value of ”6”, so it will definitely not consider this sub-tree.This is the principle of alpha pruning, the beta pruning has a similar idea with it.

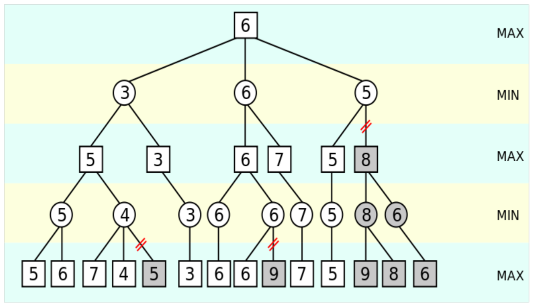

Fig. 6. The process of Alpha-beta pruning, (source:https://upload.wikimedia.org/wikipedia/commons/9/91/AB_pruning.svg)

To implement alpha-beta pruning, every nodes should have its alpha and beta value, and the initial value is $-\infty$ and $+\infty$ respectively.When it does the ”Max” search,it should updates its alpha value and if the successor’s value larger than the beta, while when it does the ”Min” search, the beta value should be updated and check whether the sub-node’s value smaller than the alpha.If it is, break the loop.

Theoretically speaking, the best-case performance of time complexity for alpha-beta pruning is $O(\sqrt{b^d})$, and the worst case for it is $O(b^d)$.In practical, it normally can reduce half of the nodes.

E. Heuristic Search

As I mentioned before, the time complexity for alpha-beta pruning ranges from $O(\sqrt{b^d})$ to $O(b^d)$ and it is mainly depends on the order of the nodes.

Take the second layer nodes in Figure.6 as an example, if we can change the order of the nodes from ”3,6,5” to ”6,5,3”, then when we check the node of ”5, it can do a pruning operation, and the another node ”3” is the same.In this case, the alpha-beta pruning performs much better.

But the problem is that, the minimax search is a kind of deep search, so we can't know the value of nodes in the second layer before we explore the fifth layer. Therefore, a rough estimate should be performed before the search, and do a sort based on the estimated value.

Here, the heuristic function be applied to do the rough estimation, but it calculates the score based on the position of the stone, rather than the global situation, so it is much more faster than the previous evaluation function.

The Threat Mark Table still be used in the heuristic function, when we do the estimation for a position in the current board.It will check four direction of the position whether there is a chance to attack and whether there is a risk of being threatened.If it exists, the point will be added with the corresponding mark.When the check finished, it will return a value that can be used to estimate the point roughly.For the max function, it will sort these numbers from the largest to the least, while for the min function, it will sort them from the least to the largest.

III. The Result

A. *Difference in running speed between applying C extension and without it*

The response time of the robot is mainly depends on the speed of evaluating the current board since the evaluation function and heuristic function run more than $100,000$ times every time.So, I decided to rewrite them in a compiled language C.And rewriting Python code in C language greatly improves the speed of the program.To compared the speed of them, I saved the consuming time of evaluation function when it do the search for 31948nodes, the I average them in the end to get a number which can show the difference in running speed significantly.Below table has shown the difference.On average, C runs 221 times faster than Python.

TABLE II
Difference in Elapsed time for evaluation function between Python and C

C(s)	Python(s)
9.596250156504319e-06	2.128892e-03

B. *The intelligence of robot*

Since the intelligence of robot is hard to quantify, so I can't provide any related evidence to prove my robot's ability. I can only provide my record when I play the game with the AI.If the ply is odd, it will only consider the attack, so I only set the depth to an even number.Since I'm an only normal player, some moves are only depends on my intuition and many lose is due to my carelessness, so this table cannot prove the ability of AI.It can only be an reference.

C. *Difference in the number of expanded nodes with different methods*

To compare the performance between difference methods, the situation of Figure.7 has been given.

TABLE III
Record about the Number of wins between me and AI

Search Depth	Me	Robot	Number of matches
1	5	0	5
2	3	2	5
4	2	3	5

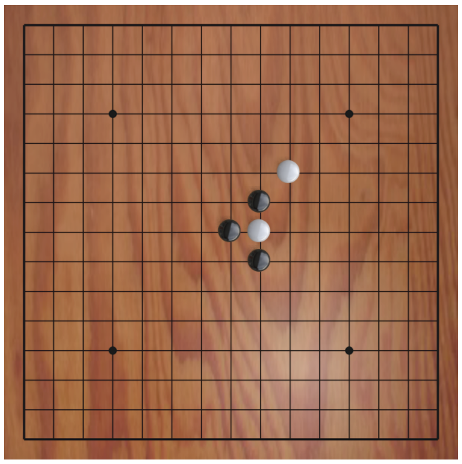

Fig. 7. Given situation

For unoptimized Minimax with more than 4 layers, the number of nodes is too large to complete the search within one hour, so here I can't provide the data about the number of nodes of the unoptimized minimax search more than 4ply.It is obviously that for all the methods, the number of nodes are increasing exponentially with the growing of the number of depth.The alpha-beta pruning works very well compared with the one without it. The one who applied the heuristic search can improve its performance significantly, since it sorts the successors in an order and increases the pruning times.

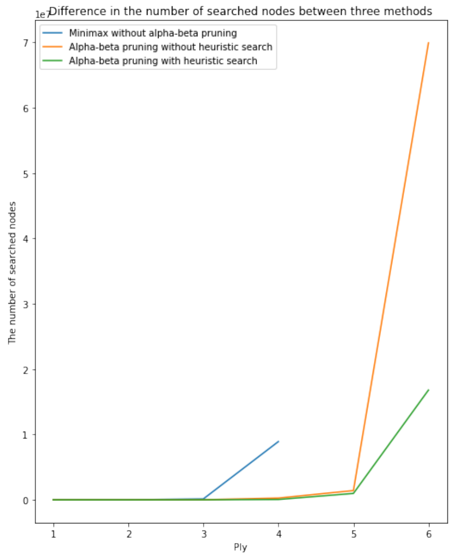

Fig. 8. Compare the number of searched nodes between three methods

IV. Self Evaluation

A. Strength

1) C extension: For this project, I combined Python with C with a very elegant way to make an balance between developing speed and running speed, and it increases the speed of evaluation time by more than 200 times indeed.

2) Heuristic Search: For alpha beta pruning, I improved it by sorting the successors order in advance, and it proves to work very well eventually.

B. Limitation

In this project, I think the time is limited for me, since there are many way to optimize the process of searching.Here are some limitations:

1) The evaluation function can still be faster: Since I am not skilled in using C language pointers. So I always copy the memory of the contents in array rather than copy the address of it, which will cost more time compared with copying the address.

2) Threat-space search should be tried: Allis proposed a threat-space search in his paper in 1993 [1].The idea of the threat-space search is only consider the direct three and direct four in the evaluation function,which means it will reduce the search space to a space of attacking moves only. Theoretically, this methods can reach more deepen depth and create an advantage for the robot immediately.

3) Iterative Depth Search: The biggest problem for the depth first search is that it hardly to find the optimal solution. In other word, the solution it finds is not the shortest way to win. For instance, if the robot find a Double-Four in the 4ply and the 6ply, and they will return the same value. There is a chance for the AI to choose the later one, which will postpone the victory.Therefore, we need to do the minimax search from the maximum search of 1, and increased it by 2 in each time, When it searched for the optimal solution, it will stop and return the value, which will speed the search up significantly since the best solution often appears in the surface layer.

4) Zobrist Hashing: The most regrettable thing of this project is that I did not implement Zobrist hashing algorithm due to the time limitation. Zobrist Hashing algorithm is used to implement a kind of hash table, transposition table.It can be used to avoid computing the same position more than one [5].The reason why I said this is the most regrettable part is that the Browser/Server architecture is extremely suitable for implementing the Zobrist Hash since it can make full use of the battle record with other players.

Zobrist hashing starts with initializing a matrix by generating bitstrings randomly. When the board situation has been changed, the final Zobrist hash value can be calculated by combining those bitstrings using bitwise XOR.

References

[1] Allis, L.V., Herik, H.J. van den, and Huntjens, M.P.H. (1993). Go-Moku and Threat-Space Search. Report CS 93-02, Department of Computer Science, Faculty of General Sciences, University of Limburg. Maastricht, The Netherlands. ISSN 0922-8721.

[2] Sakata, G. and Ikawa, W. (1981). Five-In-A-Row, Renju. The Ishi Press, Inc. Tokyo, Japan.

[3] Michael Maschler, Eilon Solan, Shmuel Zamir (2013). Game Theory. Cambridge University Press. pp. 176180. ISBN 9781107005488.

[4] Knuth, D.E. and Moore, R.W. (1975). An Analysis of Alpha-Beta Pruning. Artificial Intelligence, Vol. 6, No. 4, pp. 293-326.

[5] A. L. Zobrist, "Ordered Minimal Perfect Hash of the Human Genome and Implications for Duplicate Finding," WRI World Congress on Computer Science and Information Engineering, CSIE(CSIE), Los Angeles, CA, 2009

APPENDIX A
INTERFACE BETWEEN C AND PYTHON

```c
//
//  main.c
//  gomoku
//
//  Created by            on 30/12/2018.
//  Copyright    2018 Ziyang Wei. All rights reserved.
//
#include "Python/Python.h"
#include <stdio.h>
#include <stdlib.h>
#include "evaluate.h"
#include "check_win.h"

static PyObject* GMK_check_win(PyObject *self, PyObject *args){
    PyObject *py_board;
    int board [15][15];
    int index = 0;

    if(!(PyArg_ParseTuple(args,"O", &py_board))){
        return NULL;
    }
    if(PyList_Check(py_board)){
        int SizeOfList = (int)PyList_Size(py_board);
        for(int Index_i = 0; Index_i < SizeOfList; Index_i++){
            PyObject *ListItem = PyList_GetItem(py_board, Index_i);
            int NumOfItems = (int)PyList_Size(ListItem);
            for(int Index_k = 0; Index_k < NumOfItems; Index_k++){
                PyObject *Item = PyList_GetItem(ListItem, Index_k);
                int item_num = (int)PyLong_AS_LONG(Item);
                board[index/15][index%15] = item_num;
                index ++;
            }
        }
    }
    int winner = check_win(board);
    return (PyObject*)Py_BuildValue("i", winner);

}


static PyObject *GMK_evaluate(PyObject *self, PyObject *args){
    int agent;
    PyObject *py_board;
    int board [15][15];
    int index = 0;

    if(!(PyArg_ParseTuple(args, "iO",&agent, &py_board))){
        return NULL;
    }
    if(PyList_Check(py_board)){
        int SizeOfList = (int)PyList_Size(py_board);
        for(int Index_i = 0; Index_i < SizeOfList; Index_i++){
            PyObject *ListItem = PyList_GetItem(py_board, Index_i);
            int NumOfItems = (int)PyList_Size(ListItem);
            for(int Index_k = 0; Index_k < NumOfItems; Index_k++){
                PyObject *Item = PyList_GetItem(ListItem, Index_k);
                int item_num = (int)PyLong_AS_LONG(Item);
                board[index/15][index%15] = item_num;
                index ++;
            }
        }
    }
    long result = evaluate(agent, board);
    return (PyObject*)Py_BuildValue("l", result);
}

static PyObject *GMK_heuristic_evaluate(PyObject *self, PyObject *args){
    int agent;
    PyObject *py_board;
    int board [15][15];
    int index = 0;
    int y; int x;

    if(!(PyArg_ParseTuple(args, "iiiO",&agent,&y,&x, &py_board))){
        return NULL;
    }
    if(PyList_Check(py_board)){
        int SizeOfList = (int)PyList_Size(py_board);
        for(int Index_i = 0; Index_i < SizeOfList; Index_i++){
            PyObject *ListItem = PyList_GetItem(py_board, Index_i);
            int NumOfItems = (int)PyList_Size(ListItem);
            for(int Index_k = 0; Index_k < NumOfItems; Index_k++){
                PyObject *Item = PyList_GetItem(ListItem, Index_k);
                int item_num = (int)PyLong_AS_LONG(Item);
                board[index/15][index%15] = item_num;
                index ++;
            }
        }
    }
    long result = heuristic_evaluate(agent, y, x, board);
    return  (PyObject*)Py_BuildValue("l", result);
}


static PyMethodDef GMKMethods[] = {
    {"evaluate", GMK_evaluate, METH_VARARGS},
    {"check_win", GMK_check_win, METH_VARARGS},
    {"heuristic_evaluate", GMK_heuristic_evaluate, METH_VARARGS},
    {NULL,NULL},
};

static struct PyModuleDef MainModule = {
    PyModuleDef_HEAD_INIT,
    "GMK",
    NULL,
    -1,
   GMKMethods,
};

void PyInit_GMK(){
    PyModule_Create(&MainModule);
}
```

APPENDIX B
EVALUATION FUNCTION

```
//
//  evaluate.c
//  gomoku
//
//  Created by          on 02/01/2019.
//  Copyright    2019 Ziyang Wei. All rights reserved.
//

#include <stdio.h>
static int BLACK = 1;
static int WHITE = 2;
static int DIRECTIONS[4][2] = {{1, 0},{0,1}, {1,1}, {-1,1}};

static long FIVE = 100000;
static long DIRECT_FOUR = 10000;
static long FOUR = 1000;

static long DIRECT_THREE = 1000;
static long TWO = 100;
static long THREE = 100;

static int DEFENSIVE_RATIO = 1;

int board[15][15];

struct APoint
{
    int y;
    int x;
    int direction_y;
    int direction_x;
};

int get(struct APoint p, int offset){
    int new_y = p.y + p.direction_y * offset;
    int new_x = p.x + p.direction_x * offset;
    if(0<=new_y && new_y<15 && 0<=new_x && new_x < 15){
        return board[new_y][new_x];
    }
    return -1;
}

int is_equal(int *list1, int *list2, int len){
    for (int i=0;i<len;i++){
        if(list1[i] != list2[i]){
            return 0;
        }
    }
    return 1;
}

void print_array(int board[15][15]){
    for(int i=0;i<=14;i++){
        for (int j=0;j<=14;j++){
            printf("%d \t", board[i][j] );
        }
        printf("\n");
    }
}

long evaluate(int me, int out_board[15][15]){
    // initialize
    for(int i=0;i<15;i++){
        for(int j=0;j<15;j++){
            board[i][j] = out_board[i][j];
        }
    }
    int enemy = me == WHITE? BLACK:WHITE;
    int order[2] = {me, enemy};
    long attack_point = 0;
    long defense_point = 0;

    for(int y=0;y<15;y++){
        for(int x=0;x<15;x++){
            for(int i=0;i<2;i++){
                int agent = order[i];
                int rival = agent == WHITE? BLACK:WHITE;
                if(board[y][x] == agent){
                    for(int i=0;i<4;i++){
                        int direction_y = DIRECTIONS[i][0];
                        int direction_x = DIRECTIONS[i][1];
                        struct APoint p = {y, x, direction_y, direction_x};

                        int situation5_0[5] = {get(p, 0), get(p,1), get(p,2), get(p,3), get(p,4)};
                        int situation5_1[5] = {get(p, -1), get(p,0), get(p,1), get(p,2), get(p,3)};
                        int situation6[6] = {get(p,-1), get(p, 0), get(p,1), get(p,2), get(p,3), get(p,4)};
                        int situation7[7] = {get(p,-2), get(p,-1), get(p, 0), get(p,1), get(p,2), get(p,3),
     get(p,4)};
                        int situation6_2[6] =  {get(p,-2), get(p,-1), get(p, 0), get(p,1), get(p,2), get(p
    ,3)};
                        // Five
                        int five[5] = {agent, agent, agent, agent, agent};
                        if(is_equal(situation5_0, five, 5)){
                            if(agent == me){attack_point += FIVE;}else{defense_point += FIVE *
    DEFENSIVE_RATIO;}
                        }
                        else{// DIRECT FOUR
                            int direct_four_1[6] = {0, agent, agent, agent, agent, 0};
                            if (is_equal(situation6, direct_four_1, 6)){
                                if(agent == me){attack_point += DIRECT_FOUR;}
                                else{defense_point += DIRECT_FOUR * DEFENSIVE_RATIO;}
                            }
                            else{// FOUR
                                int four_1[6] = {0, agent, agent, agent, agent, rival};
                                int four_2[6] = {rival, agent, agent, agent, agent, 0};
                                int four_3[5] = {0, agent, agent, agent, agent};
                                int four_4[5] = {agent, agent, agent, agent, 0};
                                int four_5[5] = {agent, agent, agent, 0, agent};
                                int four_6[5] = {agent, agent, 0, agent, agent};
                                if(is_equal(situation6, four_1, 6) || is_equal(situation6, four_2, 6) ||
                                   is_equal(situation5_0, four_3, 5) || is_equal(situation5_0, four_4, 5)
    ||
                                   is_equal(situation5_0, four_5, 5) || is_equal(situation5_0, four_6, 5)){
                                    if(agent == me){attack_point += FOUR;}
                                    else{defense_point += FOUR * DEFENSIVE_RATIO;}
                                }
                                else{ // DIRECT THREE
                                    int direct_three_1[5] = {0, agent, agent, agent, 0};
                                    int direct_three_2[5] = {0, agent, 0, agent, agent};
                                    int direct_three_3[5] = {0, agent, 0, agent, agent};
                                    if(is_equal(situation5_1, direct_three_1, 5) ||
                                       is_equal(situation5_1, direct_three_2, 5) ||
                                       is_equal(situation5_1, direct_three_3, 5)){
                                        if(agent == me){attack_point += DIRECT_THREE;}
                                        else{defense_point += DIRECT_THREE * DEFENSIVE_RATIO;}}
                                else{// DEAD THREE
                                    int three_1[5] = {0, agent, agent, agent, rival};
                                    int three_2[5] = {rival, agent, agent, agent, 0};
                                    int three_3[6] = {0, agent, 0, agent, agent, rival};
                                    int three_4[6] = {rival, agent, agent, 0, agent, 0};
                                    int three_5[6] = {0, agent, agent, 0, agent, rival};
                                    int three_6[6] = {rival, agent, 0, agent, agent, 0};
                                    int three_7[5] = {agent, 0, 0, agent, agent};
                                    int three_8[5] = {agent, agent, 0, 0, agent};
                                    int three_9[5] = {agent, 0, agent, 0, agent};
                                    int three_10[7] = {rival, 0, agent, agent, agent, 0, rival};
                                    if(is_equal(situation5_1, three_1, 5) || is_equal(situation5_1, three_2
    , 5)||
                                       is_equal(situation6, three_3, 6) || is_equal(situation6, three_4, 6)
    ||
                                       is_equal(situation6, three_5, 6) || is_equal(situation6, three_6, 6)
    ||
                                       is_equal(situation5_0, three_7, 5) || is_equal(situation5_0, three_8
    , 5) ||
                                       is_equal(situation5_0, three_9, 5) || is_equal(situation7, three_10,
     7)
                                                   ){
                                                     if(agent == me){attack_point += THREE;}
                                                     else{defense_point += THREE * DEFENSIVE_RATIO;}

                                                }
                                                else{
                                                     int two_1[6] = {0, 0, agent, agent, 0, 0};
                                                     int two_2[5] = {0, agent, 0 , agent, 0};
                                                     if(is_equal(situation6_2, two_1, 6) || is_equal(situation5_1, two_2
     , 5) ||
                                                         is_equal(situation6, two_3, 6)){
                                                          if(agent == me){attack_point += TWO;}
                                                          else{defense_point += TWO * DEFENSIVE_RATIO;}}
                                                }

                                                }
                                           }

                                      }
                                 }

                            }
                       }
                  }
             }
     }
     return attack_point - defense_point;
}
```

APPENDIX C
MINIMAX SEARCH

```python
# -*- coding: utf-8 -*-
# @Time  : 30/12/2018 22:23
# @Author : weiziyang
# @FileName: utils.py
# @Software: PyCharm

import GMK
from backend.gomoku.utils import time_counter

# agent
BLACK_AGENT = 1
WHITE_AGENT = 2


class MiniMax(object):
   def __init__(self, agent, board, pos, depth, heuristic=True):
      self.board = board
      self.agent = agent
      self.pos = pos
      self.enemy = WHITE_AGENT if agent == BLACK_AGENT else BLACK_AGENT
      self.depth = depth
      self.nodes_num = 0
      self.searched_nodes = 0
      self.cut_time = 0
      self.evaluate_time = 0
      self.heuristic = heuristic

   def search_possible_position(self, board):
      """
      Based on a fact that the new move you will take will only surround by the existing
          stone, which will save many times
      :param board: Game board state
      :return:
      """
      search_matrix = [(i, j) for i in range(-2, 3) for j in range(-2, 3) if (i, j) != (0, 0)]
      possible_position = set()
      for y in range(15):
         for x in range(15):
            if board[y][x] == WHITE_AGENT or board[y][x] == BLACK_AGENT:
               for each in search_matrix:
                  new_pos = (y+each[0], x+each[1])
                  if 0 <= new_pos[0] < 15 and 0 <= new_pos[1] < 15:
                     obj = board[new_pos[0]][new_pos[1]]
                     if obj == 0 and new_pos not in possible_position:
                        possible_position.add(new_pos)
      self.nodes_num += len(possible_position)
      return possible_position

   class Point(object):
      def __init__(self, p, direction, board):
         self.y = p[0]
         self.x = p[1]
         self.d = direction
         self.board = board

      def __getitem__(self, offset):
         new_y = self.y + self.d[0]*offset
         new_x = self.x + self.d[1]*offset

         if 0 <= new_x < 15 and 0 <= new_y < 15:
            return self.board[new_y][new_x]

   def evaluate(self, board, pos):
        """
        Core code, evaluate the board to decide what next move to take
        :param board:
        :param agent:
        :return:
        """
        mark = GMK.evaluate(self.agent, board)
        mark -= self.distance(pos, self.pos)
        return mark

    @staticmethod
    def distance(point1, point2):
        """
        Calculate the distance between two stone,
        :param point1:
        :param point2:
        :return:
        """
        return max(abs(point1[0]-point2[0]), abs(point1[1]-point2[1]))

    @time_counter
    def solve(self):
        mark, pos = self.search_max(self.board, self.depth, None, -float('inf'), float('inf'))
        if not pos:
            pos = (7, 7)
        self.board[pos[0]][pos[1]] = self.agent
        return pos, self.board, mark

    def search_max(self, board, depth, pos, alpha, beta):
        self.searched_nodes += 1
        if depth == 0 or GMK.check_win(board):
            return self.evaluate(board, pos), None
        possible_position = self.search_possible_position(board)
        if self.heuristic:
            possible_position_dic = {}
            for each in possible_position:
                possible_position_dic[each] = GMK.heuristic_evaluate(self.agent, each[0], each[1],
                    board)

        possible_position = sorted(possible_position, key=lambda a: possible_position_dic[a],
            reverse=False)
        max_mark = -float('inf')
        max_pos = None

        for position in possible_position:
            board[position[0]][position[1]] = self.agent
            mark = self.search_min(board, depth - 1, position, max_mark if max_mark > alpha else
                alpha, beta)[0]
            # mark = self.search_min(board, depth - 1, alpha, beta)[0]
            board[position[0]][position[1]] = 0
            if mark >= max_mark:
                max_mark = mark
                max_pos = position
            if mark >= beta:
                self.cut_time += 1
                break
        return max_mark, max_pos

    def search_min(self, board, depth,pos, alpha, beta):
        self.searched_nodes += 1
        if depth == 0 or GMK.check_win(board):
            return self.evaluate(board,pos), None
        possible_position = self.search_possible_position(board)
        if self.heuristic:
            possible_position_dic = {}
            for each in possible_position:
```

```
            possible_position_dic[each] = GMK.heuristic_evaluate(self.enemy, each[0], each[1],
                board)
        possible_position = sorted(possible_position, key=lambda a: possible_position_dic[a],
            reverse=True)
    min_mark = float('inf')
    min_pos = None

    for position in possible_position:
        board[position[0]][position[1]] = self.enemy
        mark = self.search_max(board, depth - 1, position, alpha, min_mark if min_mark < beta
            else beta)[0]
        # mark = self.search_max(board, depth - 1, alpha, beta)[0]
        board[position[0]][position[1]] = 0
        if mark <= min_mark:
            min_mark = mark
            min_pos = position
        if mark <= alpha:
            self.cut_time += 1
            break
    return min_mark, min_pos
```

APPENDIX D
CHECK WIN

```c
//
//  check_win.h
//  gomoku
//
//  Created by          on 30/12/2018.
//  Copyright    2018 Ziyang Wei. All rights reserved.
//

#ifndef check_win_h
#define check_win_h

#include <stdio.h>

#endif /* check_win_h */


int check_win(int board[15][15]){
    for(int stone=BLACK; stone<=WHITE;stone++){
        // check row by row
        for(int y=0;y<15;y++){
            for(int x=0;x<11;x++){
                int flag = 1;
                for(int i=0;i<5;i++){
                    if(board[y][x+i]!=stone){
                        flag = 0;break;
                    }
                }
                if(flag){
                    return stone;
                }
            }
        }
        // check col by col
        for(int y=0;y<11;y++){
            for( int x=0;x<15;x++){
                int flag = 1;
                for(int i=0;i<5;i++){
                    if(board[y+i][x]!=stone){
                        flag = 0;break;
                    }
                }
                if(flag){
                    return stone;
                }
            }
        }
        // check oblique direction(from top-left to bottom-right)
        for(int y=0;y<11;y++){
            for(int x=0;x<11;x++){
                int flag = 1;
                for(int i=0;i<5;i++){
                    if(board[y+i][x+i]!=stone){
                        flag = 0; break;
                    }
                }
                if(flag){
                    return stone;
                }
            }
        }
        // from bottom-left to top-right
        for(int y=4;y<15;y++){
            for(int x=0;x<11;x++){
                int flag = 1;
                for(int i=0;i<5;i++){
                    if(board[y-i][x+i]!=stone){
                        flag = 0; break;
                    }
                }
                if(flag){
                    return stone;
```

```
                                }
                            }
                        }
                    }
    return 0;
}
```

APPENDIX E
HEURISTIC FUNCTION

```c
long heuristic_evaluate(int agent_me, int y, int x, int out_board[15][15]){
    long point = 0;
    // initialize

    for(int i=0;i<15;i++){
        for(int j=0;j<15;j++){
            board[i][j] = out_board[i][j];
        }
    };
    int agent_enemy = agent_me == WHITE? BLACK:WHITE;
    int agents[2] = {agent_me, agent_enemy};
    for(int i=0;i<2;i++){
        int me = agents[i];
        int enemy = me == WHITE? BLACK:WHITE;
        board[y][x] = me;

        for(int d_index=0;d_index<4;d_index++){
            int direction_y = DIRECTIONS[d_index][0];
            int direction_x = DIRECTIONS[d_index][1];
            struct APoint p = {y, x, direction_y, direction_x};

                for(int i=0;i<5;i++){
                    int situation5[5] = {get(p, 0-i), get(p,1-i), get(p,2-i), get(p,3-i), get(p,4-i)};
                    int five[5] = {me, me, me, me,me};
                    int direct_four1[5] = {me, 0, me, me, me};
                    int direct_four2[5] = {me, me, me, 0, me};
                    if(is_equal(situation5, five, 5)){
                        point += FIVE;
                    }
                    if(is_equal(situation5, direct_four1,5)||is_equal(situation5, direct_four2,5)){
                        point += DIRECT_FOUR;
                    }
                };

               for(int i=0;i<6;i++){
                   int situation6[6] = {get(p,-1-i), get(p, 0-i), get(p,1-i), get(p,2-i), get(p,3-i), get(p
    ,4-i)};
                   int direct_four[6] = {0,me,me,me,me,0};
                   if(is_equal(situation6, direct_four, 6)){
                       point += DIRECT_FOUR;
                   }
                   int dead_four1[6]= {0, me,me,me, me, enemy};
                   int dead_four2[6]= {enemy, me,me,me, me, 0};
                   if(is_equal(situation6, dead_four1, 6) || is_equal(situation6, dead_four2, 6)){
                       point += FOUR;
                   }
                   int three1[6] = {0,me,me,0,me,0};
                   int three2[6] = {0,me,0,me,me,0};
                   if(is_equal(situation6, three1, 6) || is_equal(situation6, three2, 6)){
                       point += DIRECT_THREE;
                   };
               }
               for(int i=0;i<5;i++){
                   int situation[5] = {get(p, -1-i), get(p,0-i), get(p,1-i), get(p,2-i), get(p,3-i)};
                   int five[5] = {0, me, me, me,0};
                   if(is_equal(situation, five, 5)){
                        point += DIRECT_THREE;
                   }
               };

              for(int i=0;i<4;i++){
                  int situation[4] = {get(p,-1-i),get(p,0-i), get(p,1-i), get(p,2-i)};
                  int two[4] = {0, me, me, 0};
                  if(is_equal(situation, two, 4)){
                        point += TWO;
                  }
              }
        }
        board[y][x] = 0;
    }
    return point;
}
```

Appendix F
GUI

```
// Material path
const IMG_PATH = 'assets/image/gomoku.png';
const TOP_BOARD_PATH = 'assets/image/topboard.png';
const BLACK_STONE = 'assets/image/stone_black.png';
const WHITE_STONE = 'assets/image/stone_white.png';
const ROBOT = 'assets/image/robot.png';
const BACK = 'assets/image/back.png';
const RESTART = 'assets/image/restart.png';

// Some canvas constant
const GAP = 40;
const MARGIN = 25;
const SIZE = 590;
const OBFUSCATION_RANGE = 10;

// Game constant
const GAME_SIZE = 15;
const BLACK_AGENT = 1;
const WHITE_AGENT = 2;

// Request constant
const URL = '/react';
const LOGGING_URL = '/logging';

// map the every position to a board position
let cor_map = {};
for(let y=MARGIN;y<=SIZE;y+=GAP){
    for(let x=MARGIN;x<=SIZE;x+=GAP){
        let point_y = (y-MARGIN)/GAP;
        let point_x = (x-MARGIN)/GAP;
        for(let i=-OBFUSCATION_RANGE;i<=OBFUSCATION_RANGE;i++){
            for(let j=-OBFUSCATION_RANGE;j<=OBFUSCATION_RANGE;j++){
                cor_map[[y+i,x+j]] = [point_y, point_x]
            }
        }
    }
}
let demo_mode = false;

// board array
let board = [];
for(let y=0;y<GAME_SIZE;y++){
    board[y] = [];
    for(let x=0;x<GAME_SIZE;x++){
        board[y][x] = 0
    }
}


// The choosen agent
let player_agent = BLACK_AGENT;
```

```
// log the last move
let last_player_move_stack= [];
let last_ai_move_stack = [];

function sleep (time) {
  return new Promise((resolve) => setTimeout(resolve, time));
}

// loading the top board
let top_board = document.getElementById('topboard');
const top_board_image = new Image();
top_board_image.src = TOP_BOARD_PATH;
let tp_canvas = top_board.getContext('2d');
const back_image = new Image();
back_image.src = BACK;
const restart_image = new Image();
restart_image.src = RESTART;
const white_stone = new Image;
white_stone.src = WHITE_STONE;
const robot = new Image;
robot.src = ROBOT;

top_board_image.onload = function(){
    tp_canvas.drawImage(top_board_image, 0, 0, 615, 100);
    tp_canvas.drawImage(back_image, 50, 20, 40, 70);
    tp_canvas.drawImage(restart_image, 150, 20, 40, 70);
    tp_canvas.drawImage(white_stone, 250, 20,40, 70);
    tp_canvas.drawImage(robot, 350, 20,40,70);
    ctx.fillStyle = "#000000";
    tp_canvas.fillRect(450, 20, 40, 70);
    ctx.fillStyle = "#ffffff";
    tp_canvas.fillRect(550, 20, 40, 70);
    ctx.fillStyle = "#FF0000";
};

top_board.addEventListener("mousemove",function(e){
    if((50<=e.offsetX && e.offsetX <=90 &&e.offsetY>20 && e.offsetY<90)||
        (150<=e.offsetX && e.offsetX<190 && e.offsetY > 20 && e.offsetY < 90)||
        (250<=e.offsetX && e.offsetX <=290 && e.offsetY>20 && e.offsetY < 90)||
        (350<=e.offsetX && e.offsetX <=390 && e.offsetY>20 && e.offsetY < 90)){
            top_board.style.cursor = 'pointer';
    }
    else{
        top_board.style.cursor = 'auto'
    }
});


function restart(){
    // clean board
    for(let y=0;y<GAME_SIZE;y++){
        for(let x=0;x<GAME_SIZE;x++){
            board[y][x] = 0;
        }
    }
    // clean stack
    last_player_move_stack = [];
    last_ai_move_stack = [];
    // draw board
    draw_board(board);
}

top_board.addEventListener("mousedown", function(e){
    demo_mode = false;
    // back
    if (50<=e.offsetX && e.offsetX <=90 &&e.offsetY>20 && e.offsetY<90){
        let player_move = last_player_move_stack.pop();
        board[player_move[0]][player_move[1]] = 0;
        let ai_move = last_ai_move_stack.pop();
        board[ai_move[0]][ai_move[1]] = 0;
        draw_board(board);
    }
    // restart
    if (150<=e.offsetX && e.offsetX<190 && e.offsetY > 20 && e.offsetY <90){
        restart();
    }
    // ai start
    if (250<=e.offsetX && e.offsetX <=290 && e.offsetY>20 && e.offsetY < 90)  {
        for(let y=0;y<GAME_SIZE;y++){
            for(let x=0;x<GAME_SIZE;x++){
                board[y][x] = 0;
            }
        }
        board[7][7] = BLACK_AGENT;
        draw_board(board);
        last_player_move_stack = [];
        last_ai_move_stack = [[7,7]];
        player_agent = WHITE_AGENT;
    }
    // demonstrate mode
    if (450<=e.offsetX && e.offsetX <=500&& e.offsetY>20 && e.offsetY < 90)  {
        alert('demo black mode has been open');
        demo_mode = true;
        player_agent = BLACK_AGENT;
    }
    if (550<=e.offsetX && e.offsetX <=600&& e.offsetY>20 && e.offsetY < 90) {
        alert('demo white mode has been open');
        demo_mode = true;
        player_agent = WHITE_AGENT;
    }
    // ai vs ai
    if (350<=e.offsetX && e.offsetX <=390 && e.offsetY>20 && e.offsetY < 90) {
        restart();
        let ai_a = BLACK_AGENT;
        let ai_b = WHITE_AGENT;
        board[7][7] = ai_a;
        let pos = [7, 7];
        last_player_move_stack.push(pos);
        let index = 0;
        let move_stack;
        let agent;
        let depth;
        let quit = false;
        let flag = false;

        while (true) {
            console.log('loop'+index);
            if (!(index % 2)) {
                agent = ai_a;
                move_stack = last_player_move_stack;
                depth = 2;
            }
            else {
                agent = ai_b;
                move_stack = last_ai_move_stack;
                depth = 4;
            }
            let data = {
                'state': board,
                'agent': agent,
                'pos': [pos[0], pos[1]],
                'depth': depth
            };
            console.log('data', data);

            $.ajax({
                type: 'POST',
                url: URL,
                async:false,
                data: {'data': JSON.stringify(data)},
                success: function (raw_res) {
                    flag = true;
                    let res = JSON.parse(raw_res);
                    if (res['status']) {
                        let msg = res['msg'];
                        let winner = msg['winner'];
                        let new_state = msg['state'];
                        let new_pos = msg['pos'];
                        board[new_pos[0]][new_pos[1]] = msg['agent'];
                        draw_board(board);
                        console.log('draw_board');
                        mark_red_point(new_pos[0], new_pos[1]);
                        move_stack.push([new_pos[0], new_pos[1]]);
                        pos = new_pos;
                        if (winner) {
                            sleep(500).then(() => {
                                    if (winner === ai_a) {
                                        alert('AI 1 win')
                                    }
                                    else {
                                        alert('AI 2 win')
                                    }
                                }
                            );
                            // convey the winner message
                            let data = {
                                'player': last_player_move_stack,
                                'ai': last_ai_move_stack,
                                    'winner': winner,
                                    'player_stone': agent
                                };
                                $.post(LOGGING_URL, {'data': JSON.stringify(data)}, function (raw_
                                    if (status !== 'success') {
                                        let msg = JSON.parse(raw_res)['msg'];
                                        alert('upload to server failed\n' + msg)
                                    }
                                });
                                quit = true;
                            }
                            if (board.toString() !== new_state.toString()) {
                                alert('The board is not same');
                            }
                        }
                        else {
                            let msg = res['msg'];
                            alert(msg);
                        }
                    }
                });
                if (quit){
                    return
                }
                if(flag){
                    flag = false;
                    index++;
                }
                else{
                    setTimeout("console.log('sleeping')", 1000);
                }
            }
        }
});




// loading the board
let canvas = document.getElementById("board");
const img = new Image();
img.src = IMG_PATH;
let ctx = canvas.getContext("2d");

img.onload = function(){
    if (!last_player_move_stack.length){
        ctx.drawImage(img, 0, 0,615,615);
        ctx.fillStyle = "#FF0000";
    }
};

function draw_board(board){
    ctx.clearRect(0,0,615,615);
    ctx.drawImage(img, 0, 0,615,615);
    for(let y=0;y<15;y++){
        for(let x=0;x<15;x++){
            let stone = board[y][x];
            if(stone){
                let image = new Image();
                if(stone === WHITE_AGENT){
                    image.src = WHITE_STONE;
                }
                else{
                    image.src = BLACK_STONE;
                }
                ctx.drawImage(image, x * GAP + MARGIN - 20, y * GAP + MARGIN - 20, 40, 40);
            }

        }
    }
}

function mark_red_point(y,x){
    ctx.fillRect(x * GAP + MARGIN, y * GAP + MARGIN,3,3);
}


canvas.addEventListener("mousemove",function(e){
    let cord= cor_map[[e.offsetY,e.offsetX]];
    if(cord){
        canvas.style.cursor = 'pointer';
    }
    else{
        canvas.style.cursor = 'auto'
    }
});



canvas.addEventListener('mousedown', function(e) {
    let cord = cor_map[[e.offsetY, e.offsetX]];
    if (cord) {
        if (board[cord[0]][cord[1]] === 0) {
            board[cord[0]][cord[1]] = player_agent;
            last_player_move_stack.push([cord[0], cord[1]]);
            draw_board(board);
            let data = {
                'state': board,
                'agent': player_agent,
                'pos': [cord[0], cord[1]],
                'depth':4
            };
            let string = JSON.stringify(data);
            if(demo_mode){
                return
            }
            else {
                try {
                    $.post(URL, {'data': string}, function (raw_res, status) {
                        if (status === 'success') {
                            let res = JSON.parse(raw_res);
                            if (res['status']) {
                                let msg = res['msg'];
                                let winner = msg['winner'];
                                let new_state = msg['state'];
                                let new_pos = msg['pos'];
                                board[new_pos[0]][new_pos[1]] = msg['agent'];
                                draw_board(board);
                                mark_red_point(new_pos[0], new_pos[1]);
                                last_ai_move_stack.push([new_pos[0], new_pos[1]]);
                                if (winner) {
                                    sleep(500).then(() => {
                                            if (winner === player_agent) {
                                                alert('You win')
                                            }
                                            else {
                                                alert('AI win')
                                            }
                                        }
                                    );
                                    // convey the winner message
                                    let data = {
                                        'player': last_player_move_stack,
                                        'ai': last_ai_move_stack,
                                        'winner': winner,
                                        'player_stone': player_agent
                                    };

                                    $.post(LOGGING_URL, {'data': JSON.stringify(data)}, functio
                                        if (status !== 'success') {
                                            let msg = JSON.parse(raw_res)['msg'];
                                            alert('upload to server failed\n' + msg)
                                        }
                                    })
                                }
                                if (board.toString() !== new_state.toString()) {
                                    alert('The board is not same');
                                }
                            }
                            else {
                                let msg = res['msg'];
                                alert(msg);
                            }
                        }
                    })
                }
                catch (e) {
                    alert('Fail in connect to server')
                }
            }
        }
    }
});
```