Project Gomoku Report

Zhaoxu Zhang 11611308

***Abstract*—This is the first project of class SUSTech CS303 , Artificial Intelligence . This project requires us to design a algorithm for chess game – "Gomoku" , which is a suffient-researched filed. Each student's program will be upload to an online platform to fight aginst others'.**

I. Preliminaries

A. Software

This project is written by python 3.6 using IDE pycharm . The library include numpy , random .

B. Algorithm

This project mainly use one-level basic search algorithm . Precisely , it just evaluate any possible next drop p and choose "the most prospective point" to go.

II. Methodology

Despite the Gomoku rule is so simple that can be describe in a sentence, the implementation is full for details handing and requires appropriate data structure to represent. This part describes the representation, the more detail of algorithm and the architecture I used in the codes.

A. Representation

To represent the chess in Gomoku , we define some rules:

- 1 : represent white chess
- -1 : represent white chess
- 0 : represent empty chess
- actor : the color of you
- unactor : the color of your anemy , we also use unactor to represent a position's color if it is out of the chessboard's size
- chess form : the situation of the chess . Many chess forms are defined below like win5 , alive4 , die4 , lowdie4 , die3 , tiao3 , alive2 , lowalive2 , die3 , die2 . Each of them will be provided with a vivid representation like "110011" in detailed algorithm
- detrimental chess form : once one side achieve this form , nothing can stop his victory

B. Architecture

Here is some functions in the program :

- Given functions
  - ***go***: get the next point to go from the candidate_list
- Self-defined functions
  - ***generate***: generate a list of the neighbors of current occupied positions that is not occupied as the list to evaluate values
  - ***calculate_person***: evaluate the value of a position if the next step is to be here (for both computer or people)
  - ***calculate_person2***: to evalueate a none-occupied position's value to determine weather the chess put at other empty position will lead a detrimental chess format here
  - ***Slice***: get chess situation of the eight direction for a position
  - ***Valid***: to judge weather a position can be put a chess

C. Detail of Algorithm

To better understand the algorithm of this program , I first answer few questions to get you understand structure of this program.

- How to know where to go next ? Using "Go function" I will evaluate each position for two situation : if my chess put here next step and if his chess put here next step. Then I will extract the higest position for me and him and compare them . If my max points larger than or equal to his, then I will go my max position. Otherwise , I will go him.
- What if there are multiple positions have same max points ? Then I will choose the position where opposite side have higher points.
- How to evaluate a position ? I will judge the chess form of a position from 4 directions and give points to different chess form .

Here describe the detail of algorithm in some vital functions.

- Go: in this function we get possible positions , calculate each position's value , decide which one to go

```
1:  function GO(Array, left, right)
2:      neighbors ← GENERATE(Chessboard)
3:      my_max, his_max, my_values, his values ← ∅
4:      for a in neighbors do
5:          valuea ← CALCULATE_PERSON(chessboard, a, mycolor)
6:          valueb ← CALCULATE_PERSON(chessboard, a, hiscolor)
7:          update my_max and my_values
8:          update his_max and his_values
9:      end for
10:     if my_max_value ≥ his_max_value then
11:         if I have many max points then
12:             add the point to candidate_list where his
13:             value is the lowest
14:         end if
15:     else
16:         if he have many max points then
17:             add the point to candidate_list where my
18:             value is the lowest
```

```
        end if
    end if
end function
```

- Generate : we use it to generate a list of possible positions

```
function GENERATE(Chessboard)
    neighbors ← ∅
    for all points a in Chessboard that is occupied do
        for points t around a do
            //around means 4 directions and length
            // <= 2
            if CHECK(t,Chessboard) then
                neighbors.add(t)
            end if
        end for
    end for
    return neighbors
end function
```

- Valid : we use it to check weather a position is possible to put chess

```
function VALID(p, chessboard)
    if t.x,t.y all in Chessboard's range then
        if t is not occupied then
            return True
        end if
    end if
    return False
end function
```

- calculate_person : we use this to calculate the value for a position . Here , the credits to different chess forms mainly comes from traditional chess skills and actural practice and test .

```
function CALCULATE_PERSON(chessboard, p, actor)
    //save chess form of 4 directions
    list all_directions ← SLICE(chessboard, p)
    for x = 1 → 4 do
        //save the number of each chess form
        dict result ← ∅

        //judge the chess form
        //Here we suppose the actor now is 1
          when representing the chess form
        //only half of the form are provided
          the second half can be get by symmetry
        if num == 5 then
            result['win5'] + +
        end if
        if num == 4 then
            if chess form is like 011110 then
                result['alive4'] + +
            end if
            if chess form is like -11111-1 then
                result['not threat'] + +
            end if
            if chess form is like -111110 then
                result['die4'] + +
            end if
        end if
        if num == 3 then
            if chess form is like -101110-1 then
                result['die3'] + +
            end if
            if chess form is like 101110_ then
                result['lowdie4'] + +
            end if
            if chess form is like 0011100 then
                result['alive3'] + +
            end if
            if chess form is like _-1111-1_ then
                result['not_threat'] + +
            end if
            if chess form is like -11110-1 then
                result['not_threat'] + +
            end if
            if chess form is like -111100 then
                result['die3'] + +
            end if
            if chess form is like -111101 then
                result['lowdie4'] + +
            end if
        end if
        if num == 2 then
            if chess form is like _011001_ then
                result['die3'] + +
            end if
            if chess form is like 001100 then
                result['alive2'] + +
            end if
            if chess form is like 01101-1 then
                result['die3'] + +
            end if
            if chess form is like 011011 then
                result['lowdie4'] + +
            end if
            if chess form is like 011010 then
                result['tiao3'] + +
            end if
        end if
        if num == 1 then
            if chess form is like 011101 then
                result['lowdie4'] + +
            end if
            if chess form is like 010111 then
                result['lowdie4'] + +
            end if
            if chess form is like 011010 then
                result['tiao3'] + +
            end if
            if chess form is like -111010 then
                result['die3'] + +
            end if
            if chess form is like 11001 then
                result['die3'] + +
            end if
            if chess form is like 10101 then
                result['die3'] + +
            end if
            if chess form is like 001010 then
```

```
                result['lowalive2'] + +
            end if
            if chess form is like 010010 then
                result['lowalive2'] + +
            end if
        end if

        //Give credits
        total ← 0
        if result['win5'] ≥ 1 then
            return 100000
        end if
        if result['alive4'] ≥ 1 and result['alive3'] ≥ 1) or (result['alive4'] ≥ 1 and result['tiao3'] ≥ 1 then
            return 12000
        end if
        if result['alive4'] ≥ 1 then
            return 11000
        end if
        if result['die4'] ≥ 2 or (result['die4'] ≥ 1 and result['alive3'] ≥ 1) or (result['die4'] ≥ 1 and result['tiao3'] ≥ 1 then
            return 10000
        end if
        if (result['lowdie4'] ≥ 1 and result['alive3'] ≥ 1) or (result['lowdie4'] ≥ 1 and result['tiao3'] ≥ 1) then
            return 9000
        end if
        if result['alive3'] ≥ 2 or result['tiao3'] ≥ 2 or (result['alive3'] ≥ 1 and result['tiao3'] ≥ 1) then
            return 5000
        else
            //test weather put chess here will lead detrimental chess form elsewhere
            for all chess point t in chessboard do
                if t's color equal to actor's color then
                    q = CALCULATE_PERSON2(t)
                    if q ≥ 5000 then
                        return q
                    end if
                end if
            end for
        end if
        if (result['alive3'] ≥ 1) then
            if actor == self.color then
                total+= 1500
            else
                total+ = 900
            end if
            if result['die4'] >= 1 then
                if actor == self.color then
                    total+ = 1000
                else
                    total+ = 1400
                end if
            end if
            if result['tiao3'] >= 1 then
                if actor == self.color then
                    total+ = 800
                else
                    total+ = 30
                end if
            end if
            if result['lowdie4'] >= 1 then
                if actor == self.color then
                    total+ = 900
                else
                    total+ = 900
                end if
            end if
            if result['alive2'] >= 2 then
                if actor == self.color then
                    total+ = 500
                else
                    total+ = 20
                end if
            end if
            if result['alive2'] >= 1 then
                if actor == self.color then
                    total+ = 200
                else
                    total+ = 20
                end if
            end if
            if result['lowalive2'] >= 2 then
                total+ = 380
            end if
            if result['lowalive2'] >= 1 then
                total+ = 18
            end if
            if result['die3'] >= 1 then
                total+ = 3
            end if
            if result['die2'] >= 1 then
                total+ = 2
            end if
        end if
    end for
end function
```

- calculate_person2 : it is same with calculate_person except with credits given only to detrimental chess forms

III. Empirical Verification

A. Design

To solve this problem , I didn't implement a highly complicated algorithm. However , I just use a one-level basic search algorithm with an excellent evaluation function to beat most of people.

My idea first comes from a blog which is not complete and sophisticated. By playing with other algorithm , I make great modification and supplement to it like more chess form and more efficient pattern matching.

I use pattern matching from four directions which ensures that there will not be duplicate. Also , I added a double check to check whether it will create a detrimental

chess form at other point which guarantees that there will not be loss.

The idea of point I give to each chess form comes from traditional chess skills and continuous practice with others . Especially , I give different credits to the same chess form for different actors . For example , for alive3 , if actor is me , then I will give 1500 , else if actor is enemy , I will give 900. In this way , there is different priority of chess forms for different actors, which means I can go "alive3" instead of stoping him to go "alive3" when these two happens at same time.

B. *Data and data structure*

Here I mainly use dict and list in python.

C. *Performance*

As is mentioned before , the performance of this program is unbelievable outstanding. It squashed into top 10 several times and finally ranked between 10 and 20. However , the time complexity of this algorithm is almost $O(n)$, which means it can give result in a flash.

IV. Acknowledgements

I want to thanks for all classmates whose algorithm give me chance to train my algorithm. I also want to thanks for TA Yao Zhao who clarify my confusion about this project. At last I would like to thank forward to all the student assistances who will assess my codes and reports and maintain the Gomoku online system.

References

[1] I'm professor Qu – Gomoku AI algorithm https://www.cnblogs.com/songdechiu/p/5768999.html.