Gomoku Project

Kai Wang 11612909

School of Computer Science and Engineering

Southern University of Science and Technology

Email: 11612909@mail.sustc.edu.cn

1. Preliminaries

This project is implementing a simple AI for Gomoku. As we all know, Gomoku is an abstract strategy board game for two players. The winner is the first player to form an unbroken chain of five stones horizontally, vertically, or diagonally [1]. In theory, computers can compute all steps and choose the most correct one. But in fact, the oversize searching space will waste too much computing time and power. Therefore, the challenge is how to make the decision better and in 5s at most at the same time.

1.1. Software

This project is written in Python using IDE PyCharm.

1.2. Algorithm

The algorithm being used in this project includes Min-Max Search, Alpha Beta Pruning and Heuristic Search.

2. Methodology

This part introduces the architecture of my code and details of algorithms. By the way, I want to describe the procedures and logic for running the program when receive chess data.

2.1. Representation

This part is going to declare some variables have been used in my program.

There are some variables given by project template:

- color: the type of stone I take
  - color = 1: represent white color
  - color = -1: represent black color
- chessboard: numpy type data which contains status data of chess
- chessboard_size: the size of chessboard
- candidate_list: the last turple in this list is the next location I set my stone

Also, some attached data needed during operation on some functions will be specified inside functions:

- COLOR_BLACK: -1
- COLOR_WHITE: 1
- COLOR_NONE: 0
- INFINITY: 10000099999
- LEVEL: the number of levels in min-max search

algorithm

- Five-in-a-raw: >=5 stones, win
- One-step-left: must win, just left one step to victory
- Two-step-left: must win, just left two steps to victory

2.2. Architecture

go: the main function

minn: part of min-max

maxm: part of min-max

get: get the list of accessible points

OnePointValue: evaluate the score of a point

JUMIAN: evaluate the score of the whole chessboard

At the beginning, the *main* function will call the *go* function, I process the message includes stone's color and chessboard data in go function. The go function also initials min-max search, get the available space and then call the *minn* function. In the *minn* function, I justify whether it is a kill step, if so it will return INFINITY. Then get the available set and call *maxm* function. Repeat this process 2 times and end in *minn* function.

Finally, the min-max will return a evaluate value which I defined in *JUMIAN* function. The *JUMIAN* function will return our total points minus those of our opponents.

2.3. Details of Algorithm

Here describe the details of algorithm used in my code.

minn/maxm are two functions to implement min-max search and alpha-beta pruning. I write them in one templet because of their similarity.

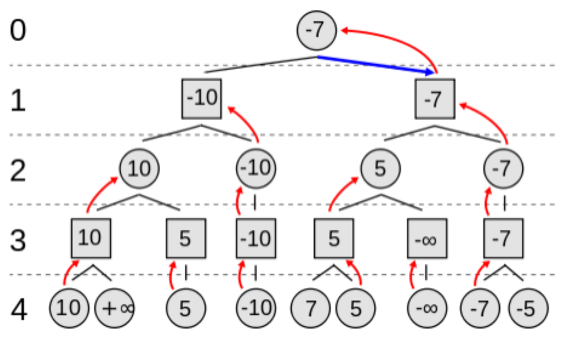

Fig1. MinMax (from wikipedia)

Algorithm 1 minn/maxm

```
Input: node, color, level, chessboard, max
Output: score
1: set the stone on the node
2: if five-in-a-raw then return INFINITY
3: else if one-step-left and opponent not five-in-a-raw
4:     then return value+100000
5: else if two-step-left and opponent not one-step-left
6:     then return value+100000
7: else if level<=0
8:     then value <- JUMIAN(chessboard)
9: end if
10: nodes[] <- get(color, chessboard)
11: for node in nodes[]
12:   do value <- maxm/minn(node, -color, level-1,chessboard, max/min)
13:   if value < min / value > max
14:       then min/max = value
15:   end if
16: end for
```

Get is a function used to find all available places. For reducing compute space, I need to choose a part of them.

Algorithm 2 get (Heuristic search)

```
Input: color, chessboard
Output: nodes list
1: firstlist <- all empty positions which distance to the nearest none empty point less than 3
2: finalist <- firstlist sorted by node's OnePointValue, take the first three points
3: return finalist
```

OnePointValue is a function which to calculate score of a place.

Algorithm 3 OnePointValue

```
Input: node, color, chessboard
Output: score of one point
1: dir = []
2: for every direction
3:     do dir.append(DIR_Direction(point, chessboard))
4:     // return how many identical pieces are there and how many other pieces or empty places after that
5: end for
6: for i in range 0 to 3
7:     dir[i], dir[i+4] describe the situation of one direction.
8:         Five-in-a-raw + 1 when >=5 stones, win
           One-step-left + 1 when must win, just left one step to victory
           Two-step-left + 1 when must win, just left two steps to victory
9: end for
10: set score for different type of situations
11: return score
```

Through experiments and references search, the scores of every chessboard type as follows.

type	score
Changlian >= 1	100000
huosi > 1 or chongsi >= 2 or (huosi >= 1 and huosan >= 1)	60000
huosi == 1	30000
chongsi >= 1 and huosan >= 1	20000
huosan >= 2	10000
chongsi == 1 and miansan == 1	300
chongsi == 1	1500
huosan == 1 and miansan > 1	1000
miansan == 1 or huoer == 1	50
huosan == 1	900
huoer >= 1 and mianer >= 1	100
mianer == 1	30
sisi >= 1	-3
sisan >= 1	-5
sier >= 1	-10
else	0

3. Empirical Verification

Empirical Verification is to check the Code_check downloaded from Sakai.

3.1. Design process

All algorithms are learned from the lab, I used them in writing my program. I also search how to make a AI for Gomoku in google, several blogs give me lots of inspirations. After I thought I was familiar with algorithms I would use, I took nearly 5 hours on writing the first version which only supported 1 level search and passed the test. Several days after that, I modified my evaluate function to get a better performance in match. But I got to know, 1-level can not be enough. Therefore, I rewrote my AI using min-max search algorithm. For reducing compute complexity, I used the heuristic search method to reduce the compute space.

3.2. Result

The result is not bad but still far from my expectation. Although it takes me really much time. The first reason I think is I am still not familiar with algorithms and waste too much time to review code to find errors. This has provided me with a lot of experience in future application development. The second reason is the architecture of my code is not clear enough. Every time I reviewed it waste a lot of time. Next time I will establish a clear architecture first and then write the inside codes. All in all, I have learned a lot in this project.

Acknowledgments

I would like to thank my classmates who discussed algorithms with me which help me finish this project. And I also want to thanks for TA Yao Zhao, TA Zeng who teaches algorithms and developed the battle platform to urge us to improve the program. Last I would like to thank forward to all the student assistances who will assess my codes and reports.

Reference

[1] Wikipedia contributors, [Online]. Available: https://en.wikipedia.org/wiki/Gomoku, [Accessed: 27- Oct-2018]