Report for Project1: Gomoku

1. Preliminaries

1.1 Software and Hardware

This project use *Python ()* version-3.7.3 with editor *PyCharm-2019.1.3()* and the platform is *Windows 10 Professional Edition* (version 1903) with AMD® RYZEN™ R5-2600X @3.90~4.20GHz 6 Cores and 12 Threads.

1.2 Algorithm

Algorithm: Basic search, traverse, Max-Min, α-β pruning.

2.Methodology

Describe the details of your representation/algorithm/architecture.

In this process, NO Max-Min and Aleph-Beta cut was really used because the multi-level cost too much time

The question of the Gomoku is very simply: find the highest point of the chessboard that can put pieces. What we should do is use a circle to get the scores of every possible point and find the highest one, then we need a judgement function to get scores of every possible point.

The judgement function first should consider “the more center, the scores higher”, so it has a 98-(x-7) ^2-(y-7) ^2 basic scores for points. Then it should traverse every line to get it’s joseki (Like “Live four” ”Dead Three”, and use a HashMap to accelerate) and count them together and get a estimate score – in this process should consider double case: one is this point is out pieces and the other is other’s.

The Algorithm is just a one-level score-judgement system. It first gets all possible points into a list, then use a role to get the score of the chessboard if the Pieces is put in here or another sides’ piece is set in here. The Role is still not hard, it just a traverse to get 15-line in cross and vertical, 21 diagonal line in double direction. And use roles to search the joseki. After all, it gets the joseki’s of each other side. Then, use a judgement-system to judge the scores, then Use the highest point to be the next step.

Use codes to describe it is that, first use get_arrays(chessboard) to get the 15-line in cross and vertical, 21 diagonal line in double direction, and then use judgement_location(chessboard) to get all the points, then In a circulate, Use a basic_decides function to get it’s scores(By the joseki’s Number). The joseki’s number come from serval traverse function use basic operation: use equal one by one. And use the model to get it’s score. Then the best location will come out.

The most important one to accelerate the process is three things, first one is the Lines, it’s actually **quotes of the chessboard**, which is a matrix, so it doesn’t need to repeat get it – when the chessboard change, it automatically changes. The second one is the **available points list**, it adds and delete points while the process instead of every circle need to produce it. The most important is a **HashMap**, which key is a Line’s information and color of This pieces and value is all the joseki’s case in this Line, it can speed the all process over 10 times.

The pseudo-code is:

```
Lines = getLines(chessboard)
Points = getPoints(chessboard)
HashMap<(Line),[]*length(josekis)>
Max_point = null
Scores = []
```

```
For I in points:
    If getScore(I,Lines, HashMap<(color,Line),[]*length(josekis)>)>max(scores):
        Max_point = I;
        Scores.append(getScores(I,Lines, HashMap<(color,Line),[]*length(josekis)>))
Return Max_point
getScores(I,Lines, HashMap<Line,[]*length(josekis)>):
    chessboard.point(I) = thisColor
    joseki1 = []
    for j in Lines:
        if HashMap[(thisColor,j)] != null:
            joseki1.append(HashMap[(thisColor,j)])
        else:
            use servel functions to traverse the joseki of j
            joseki1.append(josekis array)
            HashMap[(thisColor, j)] = josekis array
    Chessboard.point(I) = thisColor * -1
    Joseki2 = []
    for j in Lines:
        if HashMap[(thisColor*-1,j)] != null:
            joseki2.append(HashMap[(thisColor*-1,j)])
        else:
            use servel functions to traverse the joseki of j
            joseki2.append(josekis array)
            HashMap[(thisColor*-1, j)] = josekis array
    Then use servel if statement to calculate the score of the chessboard.
Return scores
```

PS: Although I write a Max-Min and aleph-Beta cut in codes, but only use level-0, which means don't exist any cut.

3. Empirical Verification

Describe the experiments that you conducted to test/verify the quality of your program. This may include (but not limit to) the following:

First of all, the experiment that I conducted to test the quality of my program is the default six cases.

3.1 How were the experiments designed?

The experiments are one blank chessboard to test the paras of the center-index, and five case for the joseki's.

3.2 What data did you use?

The data I use to test is the default six chessboard to test my program, And the Hyperparameters mainly come from three side, the first one is"名家手把手一起做运动:五子棋,洪峰,洪益娟著,江苏科学技术出版社,Page22-24",the second is "https://www.iteye.com/blog/zjh776-1979748",the 3rd come from the deduce of myself. The hyperparameters in this case is the scores of every small joseki like "double live four" or "death three", they come from the describe of situation of chessboard.

And I use time to count the performance of chessboard.

3.3 How did you measure the performance?

The first performance is must the default six case, it must can do the right action, and the most important performance target is time.one step just have five second, if the model need level-2, every step should only have 5/100/99 = 0.001s. I use the different

of begin and final time to measure the performance.

3.4 Experimental results

Although I do a lot of optimization, but now the system do once judgement of the all chessboard cost 0.016s. It cost too much time, so deep level can not be done at all because it cost too much time. What more, if don't consider time, the Max-Min and α-β pruning algorithm work all right(Although in one level α-β pruning don't do anything), but the deep level's final result is strange because the judgement-function seems can not give right evaluation of chessboard.

3.5Did the results meet your expectation about the program? Why or why not?

The time cost does not meet my expectation, it cost too much time; in another hand, the judgement function doesn't meet my expectation.

The time cost really too much time, every step just needs to judgement less than 100 equal and there even have a HashMap (from array to result) to it, but it cost more than 0.01s(what more, the server does not support *numba* speed up for *python*!).

The second is, it seems that my judgement-score function seems overfitting for the level-0 case, it's hard to give an accurate judge for the deep case.

But after all, because every step cost too much time, deep-level judgement is impossible for my program, so it doesn't matter.

Although this program doesn't use deep-level knowledge, it can evaluate the situation at a not bad precision.

4.References

List the references, please follow the IEEE format to prepare your references. The IEEE format can be found at:

[1]洪峰,洪益娟著,《名家手把手一起做运动:五子棋》,江苏科学技术出版社,Page.22-24

[2] "https://www.iteye.com/blog/zjh776-1979748"

姓名:　　学号:　　时间: