Gomoku Project

Wang Zhiyuan 11610634
CSE
Computer Science and Technology
11610634@mail.sustc.edu.cn

1. Preliminaries

1.1. Software

For this project, I write it by Python. The package I have used is use $numpy$ and $copy$

1.2. Algorithm

For this Project, I use the method of heurlstlc search. The primary part of this Algorithm is the design of the evaluation function.

And to optimize the Algorithm, I use the Min-Max Analysis to design a game tree, and use Alpha-Beta purning to simplify the process of search. But limited by the time. The depth of the tree should less then 8.

2. Methodology

2.1. Representation

In my code, according to the example given by the teacher, I design six method:

- $count()$
- $calcute\_value()$
- $get\_pos\_value()$
- $get\_pos\_list()$
- $tree()$
- $go()$

For these methods:

- The $go()$ is the method that test program will call.
- The $count()$, $calcute\_value()$ and $get\_pos\_value()$ can calucute the value of each coordinate in the chessboard which is null now.
- The $get\_pos\_list()$ and $tree()$ will build a game tree.

2.2. Architecture

- $go()$
  - $count()$
  - $calucute\_value()$
  - $get\_pos\_value()$
  - $calcute\_pos\_list()$
  - $tree()$
    - $count()$
    - $calucute\_pos\_value()$
    - $get\_pos\_value()$
    - $calcute\_pos\_list()$

2.3. Detail of Algorithm

Firstly, I need to design a evaluation function to get the value of all the coordinate with null color. I calculate the value of one location by combinate the conditions of 8 directions of this coordinate:

- Count how many chess with he same color as yours in one direction
- Count how many chess with the same color as yours if there is one null chess in one direction
- In the end of this direction is null chess or the versus color chess.

After get the conditions of all the 8 directions of the null chess coordinates. I can combinate two direction in one line and get the result in this line.

- In the **algorithm 1**. The list returned have two integer:
  - The first number show the status in this direction that without any null chess.
  - The second number show the status in this direction that with one and only one null chess.
  - For each number: If the number of the chess with self.color is n, then the value will be $2 \times n$ while there is null chess at the end, or the value will be $2 \times n - 1$
- According to the sum of the value of the two directions in one line, I can get the chess shape in this line. You can get the detail in the **algorithm 2** and the value append corresponding to the chess shape is shown on **TABLE 1.**.
- Then I will statistics the shapes on the 4 lines and give the weight value.

algorithm 1 Count

```
function COUNT(self, chessboard, a, b, j, k, COLOR)
    i ← 0
    m ← 0
    flag ← 0
    while (−1 < a + j < 15 and −1 < b + k < 15 and chessboard[a + j, b + k]! = −COLOR) : do
        if chessboard[a + j, b + k] == COLOR : then
            if flag == 0 : then
                i ← i + 1
            else:
                m ← m + 1
                a ← a + j
                b ← b + k
            end if
        else:
            if −1 < a + 2 ∗ j < 15 and −1 < b + 2 ∗ k < 15 : then
                if chessboard[a + (2 ∗ j), b + (2 ∗ k)] == COLOR and flag == 0 : then
                    m ← i
                    flag ← 1
                    a ← a + j
                    b ← b + k
                else: break
                end if
            else: break
            end if
        end if
    end while
    if (−1 < a + j < 15 and −1 < b + k < 15) : then
        if chessboard[a + j, b + k] == COLOR_NONE : then
            i ← i ∗ 2
            m ← m ∗ 2
        else:
            m ← m ∗ 2 − 1
            i ← i ∗ 2 − 1
        end if
    else:
        i ← i ∗ 2 − 1
        m ← m ∗ 2 − 1
    end ifreturn [i, m]
end function
```

TABLE 1. THE TABLE OF THE CHESS SHAPE

The SUM	Chess Shape
1	Five chesses
2	Four chesses without different color chess
3	Four chesses with a different color chess on the end
4	Three chesses without different color chess
5	Three chesses with a different color chess on the end
6	Two chesses without different color chess
7	Two chesses with a different color chess on the end
8	One chess without different color chess
9	One chess with a different color chess on the end
10	Some chesses with two different color chesses

algorithm 2 Calculate_value

```
function VALUE_OF_LINE(i1, i2, m1, m2)
    flag = i1 + i2
    if flag > 6 then
        y.append(1)
    else if flag == 6 then
        if (i1 ∗ i2 < 0ori1 == i2or(i1 == 5ori2 == 5)) then
            y.append(1)
        else:
            y.append(2)
        end if
    else if flag == 5 then
        y.append(3)
    else if flag == 4 then
        if (i1 ∗ i2 < 0) then
            y.append(10)
        else:
            y.append(4)
        end if
    else if flag == 3 then
        y.append(5)
    else if flag == 2 then
        if (i1 == i2ori1 ∗ i2 < 0) then
            y.append(10)
        else:
            y.append(6)
        end if
    else if flag == 1 then
        y.append(7)
    else if flag == 0 then
        if (i1 ∗ i2) == 0 then
            y.append(8)
        else:
            y.append(10)
        end if
    else if flag == −1 then
        y.append(9)
    end if
    if (m1 > 5orm2 > 5) then
        y.append(3)
    else if (m1 > 3andx[i + 4][0] > 1)or(m2 > 3andi1 > 1) then
        y.append(3)
    else if (i1 > 1andm2 > 1)or(m1 > 1andx[i + 4][0] > 1) then
        y.append(4)
    else if (i1 >= 0andm2 > 3)or(m1 > 3andx[i + 4][0] >= 0) then
        y.append(4)
    end if
end function
```

Then, I will build the game tree with the evaluation fucntion I design above:

- Use the evaluation function get the list include the points with thee biggest weight value.
- Use each point in the list as the max layer and build the game tree.
- Add the Alpha-Beta value attribute and set the threshold value of Beta-Alpha to purning.
- The algorithm 3. is the algorithm to build the tree. In this algorithm, I do purning when the points have Beta value is not bigger than its Alpha value.

algorithm 3 game_tree

```
function TREE(chessboard, value, pos_list, time)
    for pos ∈ pos_list do
        if time < 8 then
            chessboard[pos[0], pos[1]] ← alpha_beta
            pos_list_temp ←GET_POS_LIST(
                chessboard, −alpha_beta
)
            value_temp ←TREE(
chessboard, −alpha_beta, value, pos_list_temp, time+1
)
            chessboard[pos[0], pos[1]] ← 0
            if alpha_beta == color then
                if value_temp[1] > value[0] then
                    value[0] ← value_temp[1]
                end if
            else
                if value_temp[0] < value[1] then
                    value[1] ← value_temp[0]
                end if
            end if
            if value[0] + 2 >= value[1] then
                break
            end if
        else
            chessboard[pos[0], pos[1]] ← alpha_beta
            value_temp ←CALCUTE_VALUE(
             chessboard, pos[0], pos[1], alpha_beta
)
            interger ← interger + 1
            chessboard[pos[0], pos[1]] ← 0
            value[0] ← value_temp
        end if
    end for
end function
```

3. Empirical Verification

3.1. Design

For the verification part. I use a flight platform and design three ways to test it:

- Flight with it by myself and observe if have any problem
- Set some initial sitiuation that check if the algorithm work as I design
- Use the AI VS Platform.In on 10.20.96.148 to test the code.

3.2. Performance

For the Time Performance:

- If I just use the evalution function to get the list of the points with the highest weight value and random to get one point as the result I submit. The speed of the AI will be very fast even ignore it.
- If I build a game tree and use Alpha-Beta value method to purning. If I limit the depth of the tree less then 8 and the size of the list less then 20, each step can be finished in 5 second.

3.3. Result

For the capacity of the program, I have use this program play with a Gomoku game of Tecent, this program can win in most of time.

3.4. Analysis

For this project, I think the most important part is the design of the evaluation, for that the structure of the game tree is same in most of time, and limited by the time we can't search deeply enough. So if the evaluation consider many enough shapes of the chess, and assign suitabe weight value, it can be seem that the depth of the tree has been boost a lot with a little time cost.

References

[1] R. L. Rivest, "Game tree searching by min/max approximation," *Artificial Intelligence*, vol. 34, no. 1, pp. 77–96, 1987.

[2] D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," *Artificial intelligence*, vol. 6, no. 4, pp. 293–326, 1975.

[1] [2]