

Gomoku

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 heuristics 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 pruning to simplify the process of search. But limited by the time. The depth of the tree should less than 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 calculate 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()*
 - *calcute_value()*

- *get_pos_value()*
- *calcute_pos_list()*
- *tree()*
 - * *count()*
 - * *calcute_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 combine the conditions of 8 directions of this coordinate:

- Count how many chess with the 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 combine 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

```
1: function COUNT(self, chessboard, a, b, j, k, COLOR)
2:   i ← 0
3:   m ← 0
4:   flag ← 0
5:   while ( $-1 < a + j < 15$  and  $-1 < b + k < 15$  and
   chessboard[a + j, b + k] ≠ −COLOR) : do
6:     if chessboard[a + j, b + k] == COLOR : then
7:       if flag == 0 : then
8:         i ← i + 1
9:       else:
10:        m ← m + 1
11:        a ← a + j
12:        b ← b + k
13:      end if
14:    else:
15:      if  $-1 < a + 2*j < 15$  and  $-1 < b + 2*k < 15$  : then
16:        if chessboard[a + (2 * j), b + (2 * k)] ==
        COLOR and flag == 0 : then
17:          m ← i
18:          flag ← 1
19:          a ← a + j
20:          b ← b + k
21:        else: break
22:      end if
23:    else: break
24:  end if
25: end while
26: if ( $-1 < a + j < 15$  and  $-1 < b + k < 15$ ) : then
27:   if chessboard[a + j, b + k] ==
28:   COLOR_NONE : then
29:     i ← i * 2
30:     m ← m * 2
31:   else:
32:     m ← m * 2 − 1
33:     i ← i * 2 − 1
34:   end if
35: else:
36:   i ← i * 2 − 1
37:   m ← m * 2 − 1
38: end if return [i, m]
39: end function
```

TABLE 1. THE TABLE OF THE 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

```
1: function VALUE_OF_LINE(i1, i2, m1, m2)
2:   flag = i1 + i2
3:   if flag > 6 then
4:     y.append(1)
5:   else if flag == 6 then
6:     if (i1 * i2 < 0 or i1 == i2 or (i1 == 5 or i2 ==
7:     5)) then
8:       y.append(1)
9:     else:
10:      y.append(2)
11:    end if
12:  else if flag == 5 then
13:    y.append(3)
14:  else if flag == 4 then
15:    if (i1 * i2 < 0) then
16:      y.append(10)
17:    else:
18:      y.append(4)
19:    end if
20:  else if flag == 3 then
21:    y.append(5)
22:  else if flag == 2 then
23:    if (i1 == i2 or i1 * i2 < 0) then
24:      y.append(10)
25:    else:
26:      y.append(6)
27:    end if
28:  else if flag == 1 then
29:    y.append(7)
30:  else if flag == 0 then
31:    if (i1 * i2) == 0 then
32:      y.append(8)
33:    else:
34:      y.append(10)
35:    end if
36:  else if flag == −1 then
37:    y.append(9)
38:  end if
39:  if (m1 > 5 or m2 > 5) then
40:    y.append(3)
41:  else if (m1 > 3 and x[i + 4][0] > 1) or (m2 >
42:  3 and i1 > 1) then
43:    y.append(3)
44:  else if (i1 > 1 and m2 > 1) or (m1 > 1 and x[i +
45:  4][0] > 1) then
46:    y.append(4)
47:  else if (i1 >= 0 and m2 > 3) or (m1 > 3 and x[i +
48:  4][0] >= 0) then
49:    y.append(4)
50:  end if
51: end function
```

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

- Use the evaluation function get the list include the points with the 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 pruning.

algorithm 3 tree

```

1: function TREE(chessboard, value, pos_list, time)
2:   for pos ∈ pos_list do
3:     if time < 8 then
4:       chessboard[pos[0], pos[1]] ← alpha_beta
5:       pos_list_temp ← GET_POS_LIST(chessboard, −alpha_beta)
6:       value_temp ← TREE(chessboard, −alpha_beta, value, pos_list_temp, time +
1)
7:       chessboard[pos[0], pos[1]] ← 0
8:       if alpha_beta == color then
9:         if value_temp[1] > value[0] then
10:          value[0] ← value_temp[1]
11:        end if
12:      else
13:        if value_temp[0] < value[1] then
14:          value[1] ← value_temp[0]
15:        end if
16:      end if
17:      if value[0] + 2 ≥ value[1] then
18:        break
19:      end if
20:    else
21:      chessboard[pos[0], pos[1]] ← alpha_beta
22:      value_temp ← CALCUTE_VALUE(chessboard, pos[0], pos[1], alpha_beta)
23:      interger ← interger + 1
24:      chessboard[pos[0], pos[1]] ← 0
25:      value[0] ← value_temp
26:    end if
27:  end for
28: end function

```

3. Empirical Verification

3.1. Design

3.2. Data and data structure

3.3. Performance

3.4. Result

3.5. Analysis

Acknowledgments

References

- [1] XXXXXXXX