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 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
```

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
```

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

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.

algorithm 3 tree

function TREE($self, chessboard, alpha_beta, value, pos_list, time$)
  for $pos \in pos_list$ **do**
    if $time < 8$ **then**
      $chessboard[pos[0], pos[1]] = alpha_beta$
      $pos_list_temp = self.get_pos_list(chessboard, -alpha_beta)$
      $value_temp = self.tree(chessboard, -alpha_beta, value, pos_list_temp, time+1)$
      $chessboard[pos[0], pos[1]] = 0$
      if $alpha_beta == self.color$ **then**
        if $value_temp[1] > value[0]$ **then**
          $value[0] = copy.deepcopy(value_temp[1])$
        end if
      else:
        if $value_temp[0] < value[1]$ **then**
          $value[1] = copy.deepcopy(value_temp[0])$
        end if
      end if
      if $value[0] + 2 >= value[1]$ **then**
        $break$
      end if
    else:
      $chessboard[pos[0], pos[1]] = alpha_beta$
      $value_temp = self.calcute_value(chessboard, pos[0], pos[1], alpha_beta)$
      $self.interger = self.interger + 1$
      $chessboard[pos[0], pos[1]] = 0$
      $value[0] = value_temp$
    end if
  end for
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] XXXXXXX