The Gomoku AI

Peijun Ruan 11610214
School of Computer Science and Engineering
Southern University of Science and Technology
Email: 11610214@mail.sustc.edu.cn

1. Preliminaries

This is a project of Artificial Intelligence,which aims at implementing a simple ai of Gomoku. Gomoku also called five in a row, is a board game for two players, in which the aim is to place five pieces with same color in a row without block.As a game with complete information, people obviously want to calculate all the step to guarantee to win. Considering the huge calculation that need to deal with, even predicting 5 steps is very hard. In this project, I choose to implement a one-step algorithm.

1.1. Software

This project is implemented in Python using IDE PyCharm. The libraries being used includes Numpy,Re and Random.

1.2. Algorithm

The algorithm used in this project is mainly heuristic search.The main method to implements this search algorithm and some other functions is iteration.There totally have five functions in this project's Python code file.

2. Methodology

The rule of Gomoku is very simple,you just need to try your best to place five stone in the same row and avoid your opponent win ahead of you.This part will show how I implement the main structure of the Gomoku AI.

2.1. Representation

First of all,it is unavoidable to consider how to represent the data you use in playing gomoku, such as black stone,white stone the chessboard, the choice your program makes at last and so on.The following list show some data structure I use to represent data and the chess patterns I use.

- **Chessboard**: a numpy two-dimensional array.
- **List of candidate position**: list of tuple.
- **Output position**: the last tuple in the candidate list.
- **Chess color**: the color of stone:
  - color_white: use 1 to represent
  - color_black: use -1 to represent
- **Chess patterns**[1]:
  - five:300000 pts
  - live_four:100000 pts
  - cut_four:30000 pts
  - live_three:15000 pts
  - cut_three:5000 pts
  - live_two:3000 pts
  - cut_two:1000 pts

2.2. Architecture

Here are all functions in Python file gobang with pesudocode format.

- **Given**:
  - __initial__:initial the AI.
  - go:read a chessboard and output the decision.
- **Self define**:
  - get_candidate_simple: find all the valid position.
  - check_neighbors: check whether in a 4*4 field exists a stone.
  - get_score: count the score if AI places stone here.
  - get_stone_around: count stones of mine and opponent's.

2.3. Detail of Algorithm

Here show the psudocode of all the functions.

- **check_neighbors:**Use transversal to search the chessboard, if there is no stone around this position, it is useless to place a stone here

Algorithm 1 check whether in a 4*4 field exists a stone.

Input: $tuple$ index,$np.array$ chessboard
Output: $true\ or false$

function CHECK_NEIGHBORS($chessboard, index$)
$x, y \leftarrow index[0], index[1]$
  for $i = -3 \rightarrow 3$ **do**
    for $j = -3 \rightarrow 3$ **do**
      if $i == 0 \& j == 0$ **then**
        continue
      else if $0 <= x + i <= 14\ \&\ 0 <= y + j <= 14\ \&\ (x + i, y + j)\ is\ empty$ **then**
        return $true$
      end if
    end for
  end for
  return $false$
end function

- **get_candidate_simple:**Search the whole chessboard to find all valid position

Algorithm 2 find all the valid position.

Input: $np.array$ chessboard
Output: $list$ candidatelist

function GET_CANDIDATE_SIMPLE($chessboard$)
$list \leftarrow [\ ]$
  for $i = 0 \rightarrow 14$ **do**
    for $j = 0 \rightarrow 14$ **do**
      if $chessboard[i][j]\ is\ empty\ \&\ check\_neighbors(chessboard, (i, j))$ **then**
        $list.add[i, j]$
      end if
    end for
  end for
  return $candidatelist$
end function

- **get_score:** Check all the chess pattern if place a stone here to measure the importance of this position

Algorithm 3 calculate the score of chess pattern.

Input: $np.array$ chessboard,$tuple$ index,int color
Output: int score

function GET_SCORE($chessboard, index, color$)
$x, y \leftarrow index[0], index[1]$
$score \leftarrow 0$
$pattern\_list \leftarrow \{\ \}$
$position \leftarrow \{left\_to\_right, up\_to\_down, leftdown\_to\_rightup, leftup\_to\_rightdown\}$
  for i in $position$ **do**
    find chess pattern p by searching i at (x, y)
    if $p\ in\ Chess\ patterns$ **then**
      $score+ = p's\ score$
      $pattern\_list[p]+ = 1$
// record the number and type of pattern
    end if
  end for
  return $score, pattern\_list$
end function

- **get_stone_around:** Search how many stone around here. more concentrated the stones are located , more likely to get advantage

Algorithm 4 count stones of mine and opponent's.

Input: $np.array$ chessboard,$tuple$ index, int color
Output: $enemy$,$mine$

function GET_STONE_AROUND($chessboard, index$, $color$)
$x, y \leftarrow index[0], index[1]$
$enemy, mine \leftarrow 0$
  for $i = -1 \rightarrow 1$ **do**
    for $j = -1 \rightarrow 1$ **do**
      if $i == 0\ \&\ j == 0$ **then**
        continue
      else if $14 >= x + i >= 0\ \&\ 14 >= y + j >= 0$ **then**
        if $chessboard[x + i][y + j] == -color$ **then**
          enemy+=1
        else if
          then mine+=1
        end if
      end if
    end for
  end for
  return $enemy, mine$
end function

- **go:** Given a chessboard, search the whole board to determine which position will get the largest benefit.

Algorithm 5 output the final decision position.

Input: $np.array$ chessboard
Output: the last tuple in candidatelist

function GO($chessboard$)
$list \leftarrow get\_candidate\_simple(chessboard)$
$maxscore \leftarrow -1$ $candidatelist \leftarrow [\,]$
    for i $inlist$ **do**
        $attack, attack\_pattern \leftarrow get\_score(chessboard, i, color)$
        $defence, defence\_pattern \leftarrow get\_score(chessboard, i, -color)$
        if $attack\_pattern$ and $defence\_pattern$ $can\ consist\ of\ some\ special\ pattern$ **then**
            add extra score for those special chess pattern to attack and defence
        end if
        $value \leftarrow attack + defence$
        if $value > maxscore$ **then**
            $maxscore \leftarrow value$
            $candidatelist.append(i)$
        end if
    end for
end function

3. Empirical Verification

For empirical verification, using code_check_test.py file to check whether the algorithm works and can give the right answer. The result shows my algorithm works well.

3.1. Design

The main function in this project is go() and I implement other four functions to assist go() to meet the goal that give a decision where to place the next stone.And among the four functions(except go function), get_stone_around is an extra function aims at reducing useless computing and decrease the running time.

3.2. Data and data structure

Data used in this project is some chess_log files to improve the details and fix bugs. Data structures used include tuple,list,dictionary,array.

3.3. Performance

Instead of using game tree, this AI is simply designed using whole-chessboard-transversal.The time complexity will not be very high,which is about $\mathcal{O}(n^2)$.This AI can make decision very quickly compared to game tree algorithm.

3.4. Result

See the score and rank on website **10.20.96.148**. The score suggests my algorithm works not bad.

3.5. Analysis

This AI project is just a simple realization and not even use some advanced algorithms like game tree, Monte Carlo tree. Compared with those AI, this only consider a few situation. Thus, it may perform not so well as those using game tree and it can not handle some special opening like D4,I7. But it is also unreal to implement an AI that can deal with all possible situation by using the tools and data in our hands. In a word, this kind of simple AI's performance absolutely depends on how well you know about Gomoku, and it reminds me that it is a long way to program a true AI.

Acknowledgments

I would like to thank my classmates Kai Lin and Ziyuan Ye who give me inspiration on the AI's logic and inform me some knowledge on Gomoku game, which helps me improve the AI's performance a lot.Last I would like to thank SA who will check my codes and reports.

References

[1] Cnblogs.com.(2018).五子棋AI的思路-我是老邱-博客园.[online]Available at: https://www.cnblogs.com/songdechiu/p/5768999.html [Accessed 27 Oct. 2018].