Project 1: Gomoku

Yuejian Mo 11510511
Department of Biology
Southern University of Science and Technology
Email: 11510511@mail.sustc.edu.cn

1. Preliminaries

This project is doing some simple implementations on Gomoku. Gomoku, also called Five in a Row, is an abstract strategy board game. It is traditionally played with Go pieces (black and white stones) on a Go board, using 15x15 of the 19x19 grid intersection. [1] Gomoku has less complicity than Go, which has encouraged Deep Mind to develop powerful AI engine AlphaGo in 2016. So Gomoku will be a better particle for beginner. Design of Gomoku AI in this project try to follow the minMax tree with alpha-beta punching, which powered AI Deep Blue to challenge top chess player successfully in the first time.

1.1. Software

This project is written by Python 3.7 with editor Atom and Vim. Numpy library is being used.

1.2. Algorithm

Depth-first search in the form of minMax tree with alpha-beta punching is used. The method being used is recursion. I defined three group of functions in order to find out next chess position from input chess board. Including given functions, here are nine functions in go.py, and one is for testing.

2. Methodology

Different chess layout present different chance to win. For example, live-four and double-three must lead to success. So I evaluate the score of possible layout in final level of DFS. For every layout, I scan each line in vertical, horizontal and diagonal to calculate score and sum three together valued as whole chess layout score. Then, all score were judged by minMax tree to found out best choose position. To speed up minMax tree, alpha-beta punching was used to reduce the number of node to expand.

2.1. Representation

Some main data are maintain during process: color, candidate list, chessboard. Others data would be specified inside functions.

- **color**: The chess color of current AI
- **candidate_list**: Candidate point coordination, nested list. Final tuple in list would be return as final choice.
- **chessboard**: The chessboard layout from player.
- **depth**: The search depth

2.2. Architecture

Here list all functions in given class AI:

- Given function:
  - **__init__**: initial function, including global variables.
  - **first_chess**: if AI is black chess, this function run.
  - **go**: receive current chessboard layout, and return AI's choice.
- Self-defined function:
  - **alphaBeta**: minMax tree and alpha-beta punching.
  - **genNext**: generate available position on chessboard.
  - **evaluateState**: evaluate current chessboard score of black or white
  - **evaluateLine**: evaluate current line score of black or white
  - **mapValue**: map different chess layout in single line to score

The class AI would be executed in test platform.

2.3. Detail of Algorithm

Here describes some vital functions.

- **alphaBeta**: use minMax tree with alpha-beta punching to choose best position

Algorithm 1 alphaBeta

Input: $source\_chessboard, depth, alpha, beta, chess\_color$
Output: max_value
if ($depth == 0$) **then**

  return evaluateState($source\_chessboard$)
end if
$child\_chessboard \leftarrow source\_chessboard$
$cadidate\_points \leftarrow genNext(child\_chessboard)$
$max\_value \leftarrow alpha$
$max\_point \leftarrow candidate\_points[0]$
for $point$ in $candidate\_points$ **do**
  $child\_chessboard[point] \leftarrow chess\_color$ {action}
  $value \leftarrow -alphaBeta(child\_chessboard, depth - 1, -beta, -alpha, -chess\_color)$ {recursion}
  $child\_chessboard[point] \leftarrow 0$ {undo action}
  if ($value > max\_value$) **then**
    $max\_value \leftarrow value$
    $max\_point \leftarrow point$
    if ($max_value >= beta$) **then**
      break
    end if
  end if
end for
if (depth == initial_depth) **then**
  $candidate\_list.append(max\_point)$ {add max value point to candidate list}
end if
return max_value

- **genNext**: generate most possible position

Algorithm 2 genNext

Input: $chess\_chessboard, chess\_color$
Output: $score$
  $pos \leftarrow emptyposition$
$neighbor\_points \leftarrow []$ {initial}
for $point$ in pos **do**
  if ($point$ has any neighbor in 4 direction) **then**
    add $point$ to $neighbor\_points$
    break
  end if
end for
return $neighbor\_points$

- **evaluateState**: return current chessboard layout score

Algorithm 3 evaluateState

Input: $chessboard, chess\_color$
Output: $score$
$score \leftarrow 0$ {Initial $score$}
for every $line$ in horizontal ,vertical, diagonal **do**
  $score \leftarrow score + evaluateLine(line, chess\_color)$
end for
return $score$

- **evaluateLine**: get single line score

Algorithm 4 evaluateLine

Input: $chess_line, chess\_color$
Output: $score$
$score \leftarrow 0$
$continue\_point \leftarrow 1$
$blcok\_point \leftarrow 0$
while $point$ in $line$ **do**
  {found own chess}**if** $point == chess\_color$ **then**
  $continue\_point \leftarrow 1$
  $block\_point \leftarrow 0$
  if ($point == -chess\_color$) **then**
    $block\_point \leftarrow block\_point + 1$
  end if
  $point \leftarrow$ next $point$ in $line$
  while $point$ still in $line$ **and** $point == chess\_color$ **do**
    $point \leftarrow$ next $point$ in $line$
    $continue\_point \leftarrow continue\_point + 1$
    if $point == -chess\_color$ **then**
      $block\_point \leftarrow block\_point + 1$
    end if
  end while
  $score \leftarrow score + mapValue(continue\_point, bllock\_point)$
 end if
 $point \leftarrow$ next $point$ in $line$
end while
return $score$

3. Empirical Verification

Empirical verification is compared with given test_code.py and public test platform.

3.1. Design

Successfully to evaluate chessboard score currently. When depth is two, program output current answer within five second. When I try to run in depth of four, right answer almost take half of hour. In a word, more powerful evalute function and punching(heuristic function) are required.

3.2. Data and data structure

I use test data provided by code_test.py to test. Numpy matrix are used widely.

3.3. Performance

Right output in depth of two and time of five second. It run so slow when depth is four.

3.4. Result

Pass all test but fail to attend completion.

3.5. Analysis

The evaluate chessboard layout score cost linear time. minMax tree cost most time. The deeper, the more time it cost.

Acknowledgment

Thanks TA Yao Zhao who explain question and provide general method to solve it. And I also thanks for Kebing Sun help me debug my algorithm mistake.

References

[1] Wikipedia contributors, [Online], Available: https://en.wikipedia.org/wiki/Gomoku, [Accessed: 31- Oct- 2018].