Go Bang

Zeng Zhengran 11612527
School of Computer Science and Engineering
Southern University of Science and Technology
Email: 11612527@mail.sustc.edu.cn

1. Preliminaries

This project implements a Gobang AI using a classic game algorithm in the computer field. In the project, I mainly use the MinMax algorithm, with the Alpha-Beta pruning algorithm. With a reasonable evaluation function and inspired sorting, the search depth of MinMax can reach 4 layers in 2 to 3 seconds, which has certain chess power.

1.1. Software

The project is based on Python 3.6, and the development work is done on PyCharm IDE, using the numpy package, the random package, and the time package.

1.2. Algorithm

The project uses the MinMax algorithm, the Alpha-Beta pruning algorithm and the heuristic sorting algorithm. The MinMax algorithm is used to calculate all the possible situations in the next few steps of the AI; the Alpha-Beta pruning algorithm is used to discard the useless nodes that appear in the MinMax algorithm search process; the heuristic sorting algorithm is used to pre-sort the nodes to improve the effect of the pruning algorithm. Details will be given in the following sections.

2. Methodology

This section explains the data structure in the program, a detailed description of the algorithm, and the code architecture.

2.1. Representation

Some important data in the program, such as **chessboard**, **chess data**, **step offset** **output result**.

- **chessboard:** Current board
- **chess data:**
  - COLOR_BLACK: Black piece value
  - COLOR_WHITE: White piece value
  - COLOR_NONE: Empty point value
  - LIVE_ONE: Live one score
  - LIVE_TWO: Live two score
  - LIVE_THREE: Live three score
  - LIVE_TWO_THREE: Live double three score
  - LIVE_FOUR: Live four score
  - LIVE_FIVE: Live five score
  - DEAD_ONE: Block one score
  - DEAD_TWO: Block two score
  - DEAD_THREE: Block three score
  - DEAD_FOUR: Block four score
  - Pieces type:
    - for OXX__XXXX__
      $count$=6 $empty = 3$ $dead$=1
  - blackscoreboard: Score of each black piece on the board
  - whitescoreboard: Score of each white piece on the board
  - DirScoreCache: Scores in four directions of each piece on the board
- **step offset:**
  - idx: Candidate drop point
  - steps: Candidate point after calculation by MinMax algorithm
- **output result:**
  - candidate_list: Final drop point

2.2. Architecture

The following is the system structure of the project. In the process of Step1-Step3, if you encounter a chess type that must be processed, such as live four, double live three, and block four, the system will give priority to this chess.

Step1: Find all the empty points on the board.
Step2: For each point found in the previous step, calculate the scores at which the white and black spots fall at that point, and record the larger scores.
Step3: According to the score obtained by Step 2, all the empty points are sorted from large to small, and the first 10 points are selected, that is, the 10 points with the highest score.

Step4: The MinMax algorithm is used for the falling points of the previous step, and the scores of the points are re-evaluated. In the process of using the MinMax algorithm, the candidate points of each layer are obtained by the method of Step1-Step3, and the Alpha-Beta pruning algorithm is also used for pruning to optimize the algorithm speed.

2.3. Detail of Algorithm

The details of the main algorithm.

Algorithm 1 pointScore

Input:
Number of piece, $count$;
Blocked condition, $dead$;
Empty position, $empty$;
Output: The score of the enter piece type, $result$;

if count == COUNTS **then**
  if dead == DEADS **then**
    if empty == EMPTYS **then**
      return score
    end if
  end if
end if
return 0

Algorithm 2 getScore

Input:
Current chess board, $chessboard$;
The attribute of the input point, r, c, $color$;
Calculated direction, dir;
Output: The score obtained when placing the input point on the board, $result$;

if dir == 0 or 1 **then**
  find the stype of the point in horizontal line of r, c
  use $pointScore()$ get the score of this stype
  save the score on $DirScoreCache[role][r][c][1]$
end if
if dir == 0 or 1 **then**
  find the stype of the point in vertical line of r, c
  use $pointScore()$ get the score of this stype
  save the score on $DirScoreCache[role][r][c][2]$
end if
if dir == 0 or 1 **then**
  find the stype of the point in acclivity line of r, c
  use $pointScore()$ get the score of this stype
  save the score on $DirScoreCache[role][r][c][3]$
end if
if dir == 0 or 1 **then**
  find the stype of the point in declivity line of r, c
  use $pointScore()$ get the score of this stype
  save the score on $DirScoreCache[role][r][c][4]$
end if

Algorithm 3 sortPos

Input:
Current chess board, $chessboard$;
Board of black piece scores, $blackscoreboard$;
Board of white piece scores, $whitescoreboard$;
Current color, $color$;
Output: Candidate list, $result$;

for all empty $point$ in $chessboard$ **do**
  $point.blackScore$ = $blackscoreboard$[$point$]
  $point.whiteSocre$ = $whitescoreboard$[$point$]
  $point.maxScore$ = MAX($blackScore$,$whiteSocre$)
  if $point.maxScore$ is special type score **then**
    add $point$ to corresponding type list
  else
    add $point$ to normal list
  end if
end for
if special list is not empty **then**
  return special list
end if
sort normal list from largest to smallest with key $point.maxScore$
$result$ = The first 10 $point$ of normal list
return $result$

3. Empirical Verification

3.1. Design

The main modules in the project are evaluation functions, heuristic sorting, MinMax algorithm, and Alpha-Beta pruning algorithm. During the evaluation process, the program calculates and stores the scores in the four directions of the position, so that when updating the board score, you can only select the score in one direction to optimize the evaluation speed. Then in the heuristic process, pre-sort the candidate points that need to perform MinMax calculation, and find out the special cases that need to be prioritized to make full use of the pruning effect. After completing the above work, it is necessary to continue to optimize the evaluation function and solve the BUG in the evaluation function, in order to further improve the chess power. In addition, at the end of the project, I plan to add killing, but because there is still a bug, it is not used in the project.

3.2. Data and data structure

Some important data structure descriptions.

- chessboard: ł15X15numpy.ndarray
- $blackscoreboard$: A 15X15 numpy.ndarray
- $whitescoreboard$: A 15X15 numpy.ndarray.
- $DirScoreCache$: A 2X5X15X15 numpy.ndarray, used to store the score of each black and white piece in four directions
- $point$: A list of length 3, the first two are the position of the piece, and the third is the score

Algorithm 4 mm

Input:
Current chess board, $chessboard$;
Board of black piece scores, $blackscoreboard$;
Board of white piece scores, $whitescoreboard$;
Current color, $color$;
$deep$; $alpha$; $beta$;
Output: Optimal drop point score, $bestScore$;

_score = Total score for the color of the current board
$bestScore$ = _score
if $deep < 0$ or $|\_score| >= LIVE\_FIVE$ **then**
  return $bestScore$
end if
_steps = sortPos($chessboard$, $blackscoreboard$, $whitescoreboard$, $color$)
for all $stept$ in _steps **do**
  $point$ = $stept.point$
  put this $point$ in $chessboard$ and update $blackscoreboard$ and $whitescoreboard$
  $pScore$ = mm($chessboard$, $blackscoreboard$, $whitescoreboard$, $changeColour(color)$, $deep$-1, -$beta$, -$alpha$)
  $pScore$ = $-pScore$
  remove this point in $chessboard$ and update $blackscoreboard$ and $whitescoreboard$
  $bestScore$ = MAX($bestScore$, $pScore$)
  $alpha$ = MAX($alpha, bestScore$)
  if pScore >= beta **then**
    return pointScore
  end if
end for
return $bestScore$

- $steps$: List with a length of 6, the first two are the piece position, the third is the score when the blacks fall here, the fourth is the score when the whites fall here, and the fifth is the maximum value of third and fourth. The sixth is the current color

3.3. Performance

After using the heuristic and pruning algorithm, when the layers of the MinMax search algorithm is 4 and the number of children is limited to 10, the program needs 2 to 3 seconds to think of a move, and when the number of search layers is 6 The program takes about 25 seconds to think about a move. And for the killing type, the program can also directly identify and make decisions within 1 second.

3.4. Result

The program was ranked 34 in the first round of the championship and 25 in the second round of the championship, partly confirming the effectiveness of the program.

Algorithm 5 minMax

Input:
Current chess board, $chessboard$;
Board of black piece scores, $blackscoreboard$;
Board of white piece scores, $whitescoreboard$;
Candidate drop point $steps$
Current color, $color$;
$deep$; $alpha$; $beta$;
Output: Optimal drop point score, $alpha$;

for all $stept$ in $steps$ **do**
  $point$ = $stept.point$
  put this $point$ in $chessboard$ and update $blackscoreboard$ and $whitescoreboard$
  $pScore$ = mm($chessboard$, $blackscoreboard$, $whitescoreboard$, $changeColour(color)$, $deep$-1, -$beta$, -$alpha$)
  $pScore$ = $-pScore$
  $alpha$ = MAX($alpha, pScore$)
  remove this point in $chessboard$ and update $blackscoreboard$ and $whitescoreboard$
  set $point.socre$ = $pScore$
end for
return $alpha$

3.5. Analysis

Evaluation algorithm, MinMax algorithm and alpha-beta pruning algorithm are the most important parts of this project. After the above work, we can optimize the pruning process and evaluation process.Then we need to continue to optimize the evaluation function and solve bugs in order to further improve our chess skills.If further improvement is needed, we can consider adding the killing algorithm.

Acknowledgments

I would like to thank professor Ke Tang for his excellent teaching, which has pointed out a bright way for me in the process of learning artificial intelligence. I would also like to thank TA Yunwen Lei for his careful interpretation of the LAB. Last, I would like to thank all SA's efforts to provide a platform for fighting, and thank them for assess my codes and reports.

References

[1] lihongxun945. [Online]. Available: https://github.com/lihongxun945/myblog/issues/11 [Accessed: 29- Oct- 2018]