

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.

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**Initialisation :*

1: first statement

*LOOP Process*2: **for** $i = l - 2$ to 0 **do**

3: statements..

4: **if** $(i \neq 0)$ **then**

5: statement..

6: **end if**7: **end for**8: **return** *max_value*

- genNext: generate most possible position

Algorithm 2 genNext

Input: *chess_line, chess_color***Output:** *score**Initialisation :*

first statement

*LOOP Process*2: **for** $i = l - 2$ to 0 **do**

statements..

4: **if** $(i \neq 0)$ **then**

statement..

6: **end if****end for**8: **return** *score*

- evaluateState: return current chessboard layout score

Algorithm 3 alphaBeta

Input: *chessboard, chess_color***Output:** *score**Initialisation :*

first statement

*LOOP Process***for** $i = l - 2$ to 0 **do**

3: statements..

if $(i \neq 0)$ **then**

statement..

6: **end if****end for****return** *score*

- evaluateLine: get single line score

Algorithm 4 evaluateLine

Input: *chess_line, chess_color***Output:** *score**Initialisation :*

first statement

*LOOP Process***for** $i = l - 2$ to 0 **do**

statements..

4: **if** $(i \neq 0)$ **then**

statement..

end if**end for**8: **return** *score*

3. Empirical Verification

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

3.1. Design

3.2. Data and data structure

3.3. Performance

3.4. Result

3.5. Analysis

Acknowledgments

References

[1] xxx