Report for Project Gomoku

11610511 Shiyi Li

October 2018

1 Preliminaries

This project is to implement a gomoku AI. Gomoku is a strategy board game. The size of chessboard is usually 15×15. The winner is the first player to form an unbroken chain of five stones horizontally, vertically, or diagonally[1]. In the project, forbidden moves are not considered. I use the greedy algorithm to decide the point of next step. An evaluation function is given to evaluate each point on the chessboard and find the point with the highest point as result. The project is written in python and the IDE is PyCharm. The additional library used in this project is numpy.

2 Methodology

The basic definition of each variables will be introduced in the methodology.

2.1 Representation

Several variables should be defined in advance in the algorithm.

- COLOR_BLACK: the chess at this point is black
- COLOR_WHITE: the chess at this point is white
- COLOR_NONE: the point is empty
- chessboard_size: size of chessboard
- color: the color of AI
- candidate_list: list of tuple, store the result of AI

2.2 architecture

- Class AI:
  - go: the entrance of the algorithm
- self defined:
  - search: traverse the whole chessboard to find the point with the highest score
  - get_point: decide whether the point is out of chessboard
  - not_only_one: find the point which does not have any neighbor
  - evaluate: evaluate the score of each point
  - count: figure out the line of the point in each direction and count the number of each type
  - matching: find out the number of each type in each line
  - chess_type: all the types of Gomoku

2.3 Algorithm

Here shows the exact algorithm used in the Gomoku AI.

Algorithm 1 search

Input: chessboard, color, candidate_list
Output: candidate_list

```
1:  max = −1 << 28
2:  put the central point into candidate_list
3:  for each point on the chessboard do
4:    if he point is empty and has neighbor then
5:      the point is black
6:      score1 ← the score of the point
7:      the point is white
8:      score2 ← the score of the point
9:      initialize the point to empty
10:     score ← max(score1, score2)
11:     if score > max then
12:       put the point into candidate_list
13:     end if
14:   end if
15: end for
```

search: traverse the whole chessboard and choose the best point.

Algorithm 2 count

Input: all_type, num, chessboard \\all_type is a list of all the chess type in current color, num is a list to store the count for each type
Output: num

```
    for each type of the current color do
      for each horizontal line do
3:      traverse the line and count the number of
        each type
      end for
      for each vertical line do
6:      traverse the line and count the number of
        each type
      end for
      for each diagonal line do
9:      traverse the line and count the number of
        each type
      end for
    end for
```

count: count all the special chess type on the chessboard if we choose one point and provide the data to evaluate the score.

Algorithm 3 evaluate

Input: chessboard, color, my_color \\my_color is the color of AI, color is the color of current point
Output: score

```
    score = 0
2:  all_type ← chess_type(color) \\get the chess type
    of current color
    num_type ← the count for all types of current
    point
4:  for each type do
      if the count for the type ≥ 1 then
6:      score ← score + corresponding score
      end if
8:    if the count for the type of live ≥ 2 then
        score ← score + corresponding score
10:   end if
      if the count for the type of live ≥ 1 & the count
      for the neighboring type of sleep = 1 then
12:     score ← score + corresponding score of type
        sleep
      end if
14:   if color ! = mycolor then
        score ← score ×0.9
16:   end if
    end for
```

evaluate: evaluate the score of each point.

3 Empirical Verification

3.1 Experiment Design

The experiment is to conduct a chessboard and test whether the algorithm can give the best point of next step. Experiments can include defense and attack. AI need to analysis the chessboard and give the best result.

3.2 Performance

Performance is measured by the next point. The algorithm performances well if next point can handle the attack of opponent or attack the opponent effectively. The time complexity is $O(n^4)$ if the size of

chessboard is nn. It would assume lots of time to find out the final answer.

3.3 Analysis

The Gomoku AI uses basic greedy algorithm which can only provide limited performance. First, the time complexity of the algorithm need to be improved further cause it needs to traverse the whole chessboard every time. Second, the evaluate can not provide convinced evaluation of each point when the number of chess on chessboard reached a certain level. It would perform better if it is implemented by minmax algorithm.

References

[1] Wikipedia contributors. https://en.wikipedia.org/wiki/Gomoku. Accessed October 26, 2018.