Final Project: Gomoku

DuMengfei   Zhang Xianyin
18307130148   18307130146

University of Fudan — January 19, 2021

1 Introduction

Gomoku is also known as the five-in-row,which is a board game played with black or white pieces in an 20×20 board. The player who first get the five continuous pieces by row or column or oblique line is the winner of the game.Human players strategies usually involve blocking opponents lines, keeping his line unblocked, and perform multiple attacks.There are also many algorithms to deal with this type of game problem. And the improving ways come from them.

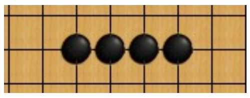
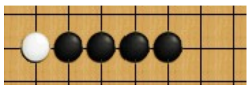
2 Improvements

(I) For human strategies,

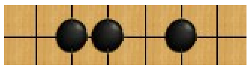
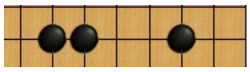
(i) VCT(Victory of Continuous Three) and VCF(Victory of Continuous Four) are the prevalent strategies for people to get victory.The player combines four-pieces and three-pieces chess shape to continuously get four continuous pieces in that force the oppoent can not block the two threat in one step.

In our implement, we combine VCT and VCF to find the kill chess ()

(ii) There also exists many special chess shapes which has different threat level.For example, Live-Four is the absolutely most threatening chess shape in the game, while the Sleep-Four is less threatening chess shape.(The concreate chess shapes are as follows)

And the blanks in the chess type is also important for the chess's threat. For example , Live-Three with one blank is much more threatening than Live-Three with two blanks.

For different chess types ,we can set more accuracy weights according to the threatening level in that make sure our evaluation function more effective.

As human can remember the potential chess he has already thought in his mind, AI should also **store** the chess table to avoid repeated exploration and finally accelerate the simulating kill chess or evaluataion process.

Directly storing the chess table requires lots of space. So we use **Zobrist**. We firstly initialize a random number table for the chess table for both black and white chess. which requires 20 * 20 * 2 random numbers in (0 , 1e30). Then using **exclusive or** to get a **approxiemately unique** chess table value.

(II) For different algorithms, we implement the MCTS algorithm. And the details of this algorithm will be discussed later.

(III) For time limitation, we try to use package of time in python to deal with it. And we also try to use the idea of Greedy to deal this problem.

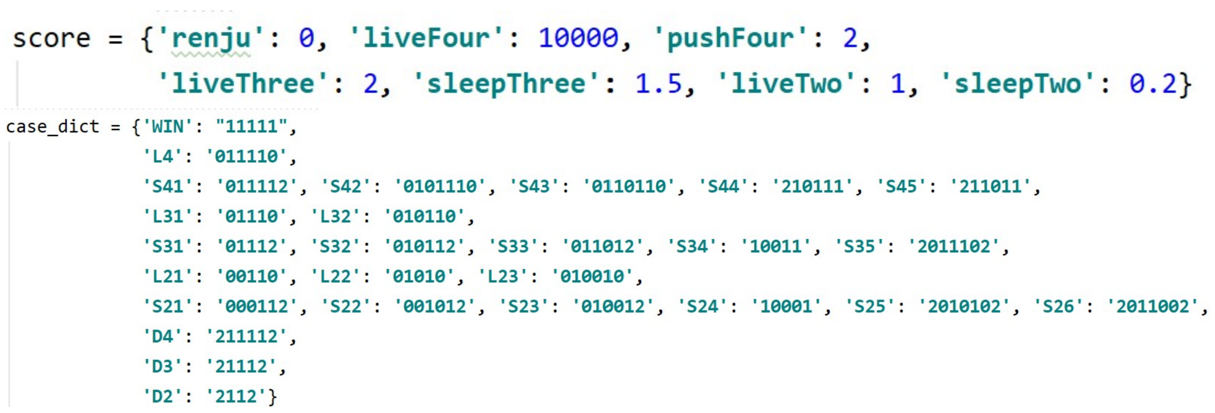
3 Method

In this report , we will introduce three methods to deal with the Gomoku.

3.1 Alpha-beta pruning

Based on the version in the midterm, we use some human strategies to strengthen our agent.
The details are as follows:

(a) The chess type is more than the last version. In the last version, we just have seven chess types. And in the new version, we consider the blanks in the chess type due to the number of blanks can influence the threatening level of chess type.The details are as follows.



```
score = {'renju': 0, 'liveFour': 10000, 'pushFour': 2,
          'liveThree': 2, 'sleepThree': 1.5, 'liveTwo': 1, 'sleepTwo': 0.2}
case_dict = {'WIN': "11111",
             'L4': '011110',
             'S41': '011112', 'S42': '0101110', 'S43': '0110110', 'S44': '210111', 'S45': '211011',
             'L31': '01110', 'L32': '010110',
             'S31': '01112', 'S32': '010112', 'S33': '011012', 'S34': '10011', 'S35': '2011102',
             'L21': '00110', 'L22': '01010', 'L23': '010010',
             'S21': '000112', 'S22': '001012', 'S23': '010012', 'S24': '10001', 'S25': '2010102', 'S26': '2011002',
             'D4': '211112',
             'D3': '21112',
             'D2': '2112'}
```

The more chess type can help us judge more situations during the game.

(b) Consider the kill chess. Before the normal min-max search, the new version will first compute some kill chess based on the board situation. For example, if in our turn we have a pushFour and a liveThree both in one position in the board and the oppoent has other threatening chess, we will choose the position instead of using min-max search.

(c) VCF and VCT. In the process of computing kill chess, we mainly use the idea of VCF and VCT.

During the computing kill chess, we already win no matter opponent set chess. For opponent, if he wants to "**struggle**", he would try hard to "block" us. We store the chess table by Zobrist method until we succeed to save the repeated exploration. If he would not "struggle", that is to say, he does not block us. This way, special chess table is not stored, but the opponent will lose faster.

3.2 Greedy

However, the alpha-beta pruning may sometimes be overtime. So we consider whether there exists a algorithm that can compute position as faster as it can. So we implemnt the Greedy algorithm.
The basic idea is as follows:

Algorithm 1: `Greedy`

```
Input: $board, player$
Result: $Best\_Position$, such that $(x, y)$

successors ← get neighbors(board) Myboard ←
  evaluate(board,player1) Oppoentboard ←
  evaluate(board,player2)
for successor in successors do
    v_current ← position_evaluate(board,position)
    V ← v_current + Myboard - Oppoentboard
end
if V > Maxvalue then
    Maxvalue ← V
    Best_Position ← successor
end
return Best_Position
```

However, the result of Greedy may not very good, so we finally add a time constrain to help agent not be overtime.

3.3 MCTS

Monte Carlo Tree Search (MCTS) is a method to find an optimal action by taking random samples in the problem space and construct a search tree according to the results. It has already had a profound impact on some problems which can be represented as trees of sequential decisions,liek games and planning problems.

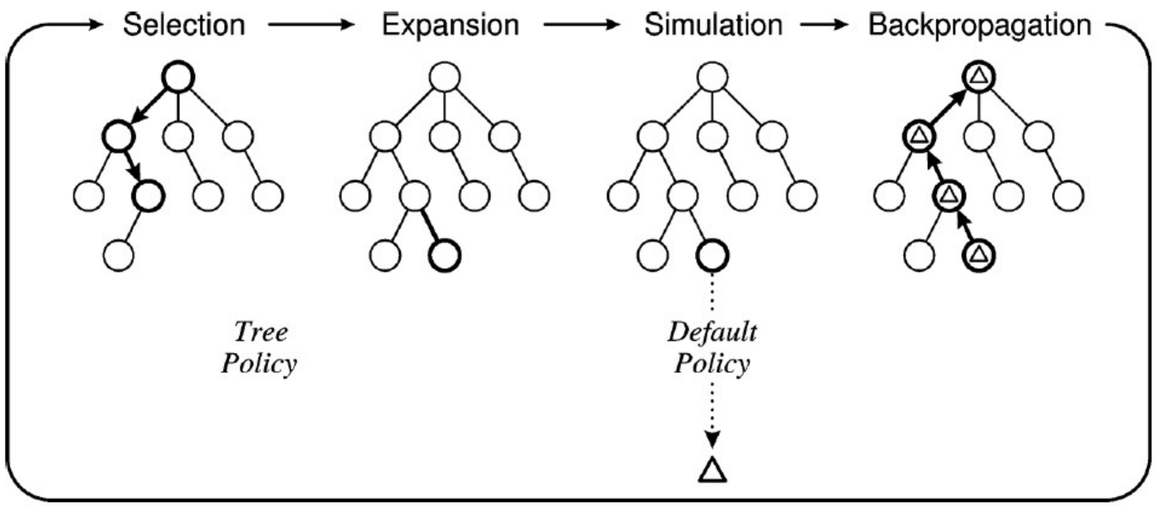
Algorithm process

(1) Selection: Starting at the root node, a child selection policy is recursively applied to descend through the tree until the most urgent expandable node is reached. A node is expandable if it represents a nonterminal state and has unvisited (i.e., unexpanded) children.

(2) Expansion: One (or more) child nodes are added to expand the tree, according to the available actions.

(3) Simulation: A simulation is run from the new node(s) according to the default policy to produce an outcome.

(4) Backpropagation: The simulation result is backed up (i.e., backpropagated) through the selected nodes to update



Concrete Description

(1) Selection: For tree policy , we use UCB1 algorithm to compute each node value and select the best children and expand it.

$$\mathrm{UCT} = \bar{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}}$$

(2) For expansion , the expansion terminating criterion is that all potential children have been visited in th Monte Carlo Tree. Andthe potential children are from get_successors().In the theory, it should consider all the position in the chess Board.

(3) For the function get_successors(): we can adjust the function to control the depth of the MC tree.To get more effective successor, we also implement some useful experience.

(4) Simulation: The black and white pieces are randomly put in the board until someone win the game or the process satisfy the condition.

(5) Backpropagation: The reward is defined as follows:
Player1 wins:return 1
Player2 wins:return 0
No one winsreturn 0.05(a relative small number) The value is not 0.5 in order to try best to find win point

(6) The number of the simulation is defined by the time of each action.

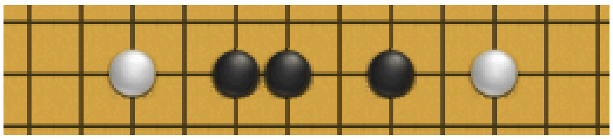
Imporvement

(1) Tuned method. In the process of expansion , we can use some experience to make it convergence faster. Replace $\frac{2ln\ n}{n_j}$ in UCB1 with $\frac{lnn}{n_j} * min\{0.25, V_j(n_j)\}$
where $V_j(s) = (0.5 * \sum_{r=1}^{s} X_{j,r}^2) - \overline{X_{j,s}} + \sqrt{\frac{2ln\ t}{s}}$ For example, for the chess shape: When you have a "LIVETHREE" and opponent do not have a "FOUR", it is definitely correct to set chess to achieve liveFour rather than pushFour. However, as MCTS does have randomness

As picture below, there are 3 candidate points for our Monte Carlo Tree. Do **20 trials**, with 200 **Simulations** for each trial.

The result is significant.
original UCB1 method rarely converges to "liveFour" : 0 or 1 / 20 trials (200 simulations per trial)
tuned methond is better: 3 or 4 or 6 / 20 trials (200 simulations per trial)

(Note that when simulation number to 2000 or more. Both "tune" or "not tuned" converges to correct answer.)

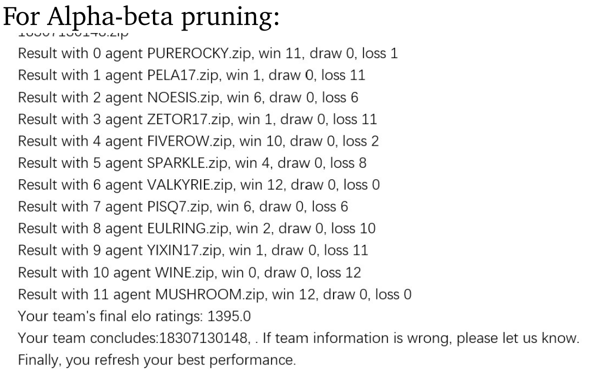


(2) Use the better successors to make the simulation effectively.

4 Results

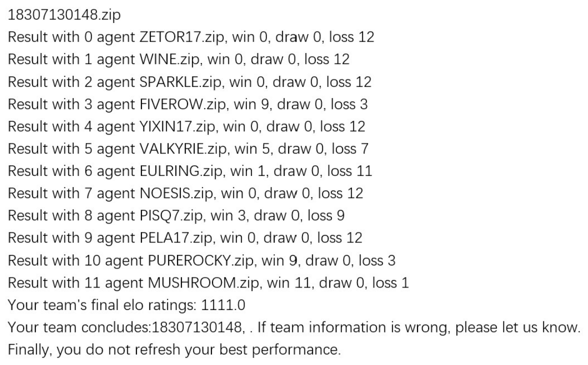
The final results of the three methods are as follows:

For Alpha-beta pruning:



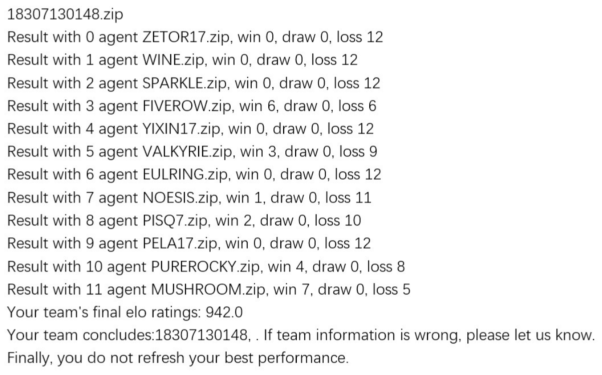
```
18307130148.zip
Result with 0 agent PUREROCKY.zip, win 11, draw 0, loss 1
Result with 1 agent PELA17.zip, win 1, draw 0, loss 11
Result with 2 agent NOESIS.zip, win 6, draw 0, loss 6
Result with 3 agent ZETOR17.zip, win 1, draw 0, loss 11
Result with 4 agent FIVEROW.zip, win 10, draw 0, loss 2
Result with 5 agent SPARKLE.zip, win 4, draw 0, loss 8
Result with 6 agent VALKYRIE.zip, win 12, draw 0, loss 0
Result with 7 agent PISQ7.zip, win 6, draw 0, loss 6
Result with 8 agent EULRING.zip, win 2, draw 0, loss 10
Result with 9 agent YIXIN17.zip, win 1, draw 0, loss 11
Result with 10 agent WINE.zip, win 0, draw 0, loss 12
Result with 11 agent MUSHROOM.zip, win 12, draw 0, loss 0
Your team's final elo ratings: 1395.0
Your team concludes:18307130148, . If team information is wrong, please let us know.
Finally, you refresh your best performance.
```

For Greedy:



```
18307130148.zip
Result with 0 agent ZETOR17.zip, win 0, draw 0, loss 12
Result with 1 agent WINE.zip, win 0, draw 0, loss 12
Result with 2 agent SPARKLE.zip, win 0, draw 0, loss 12
Result with 3 agent FIVEROW.zip, win 9, draw 0, loss 3
Result with 4 agent YIXIN17.zip, win 0, draw 0, loss 12
Result with 5 agent VALKYRIE.zip, win 5, draw 0, loss 7
Result with 6 agent EULRING.zip, win 1, draw 0, loss 11
Result with 7 agent NOESIS.zip, win 0, draw 0, loss 12
Result with 8 agent PISQ7.zip, win 3, draw 0, loss 9
Result with 9 agent PELA17.zip, win 0, draw 0, loss 12
Result with 10 agent PUREROCKY.zip, win 9, draw 0, loss 3
Result with 11 agent MUSHROOM.zip, win 11, draw 0, loss 1
Your team's final elo ratings: 1111.0
Your team concludes:18307130148, . If team information is wrong, please let us know.
Finally, you do not refresh your best performance.
```

For MCTS:



```
18307130148.zip
Result with 0 agent ZETOR17.zip, win 0, draw 0, loss 12
Result with 1 agent WINE.zip, win 0, draw 0, loss 12
Result with 2 agent SPARKLE.zip, win 0, draw 0, loss 12
Result with 3 agent FIVEROW.zip, win 6, draw 0, loss 6
Result with 4 agent YIXIN17.zip, win 0, draw 0, loss 12
Result with 5 agent VALKYRIE.zip, win 3, draw 0, loss 9
Result with 6 agent EULRING.zip, win 0, draw 0, loss 12
Result with 7 agent NOESIS.zip, win 1, draw 0, loss 11
Result with 8 agent PISQ7.zip, win 2, draw 0, loss 10
Result with 9 agent PELA17.zip, win 0, draw 0, loss 12
Result with 10 agent PUREROCKY.zip, win 4, draw 0, loss 8
Result with 11 agent MUSHROOM.zip, win 7, draw 0, loss 5
Your team's final elo ratings: 942.0
Your team concludes:18307130148, . If team information is wrong, please let us know.
Finally, you do not refresh your best performance.
```

We can find that due to the computing resource and the tim e limitation (the MCTS only simulation 2000 times which may not convergence),the alpha-beta pruning performs the best.

5 Conclusion

In this project, we have explored three approaches to develop an AI agent for Gomoku game. Due to the computing resource , the method based on simulation is not very useful. And the most effective approach is to use human knowledge about Gomoku to guide the agent to do the best.

To obtain an optimal agent, we consider the following directions:

(1) Add more expert knowledge.For example, there exists many must-win chess type in the no forbidden Gomoku. During the game , the chess situation may construct it .And by using this knowledge , our agent can win easily.

(2) Decrease the computing time. The computing time is the main problem during our programing. Although we set the time check but it just check the time limitation and return the position which may not be the best.So we can use more effective evaluation method or language like C++.

6 Refference

[1]Browne C B , Powley E , Whitehouse D , et al. A Survey of Monte Carlo Tree Search Methods[J]. Computational Intelligence & Ai in Games IEEE Transactions on, 2012, 4(1):p.1-43.
[2]Kocsis, Levente & Szepesvári, Csaba & Willemson, Jan. (2021). Improved monte-carlo search.