ISSN: 2664-9640

Gobang game system based on improved α-β shear branch algorithm

Zhuang Yan

School of Computer, Harbin Engineering University, Harbin,150001, China

Abstract

Gobang game is an important application in the field of artificial intelligence. Various advanced machine game systems have promoted the rapid development of machine game theory and related technologies [1]. Because the computer game research in chess is relatively mature, there are moves generation, chess position evaluation, game tree search algorithm and various parameter optimization strategies in chess machine game. This paper mainly aims at the algorithm design of gobang game to realize man-machine war, which is divided into three stages: The game tree search without pruning, the game tree search with alpha-beta pruning, the game tree search with heuristic evaluation and limit the search range to optimize the pruning algorithm, the effects of the three-stage implementation are compared, and the final heuristic evaluation of the optimal algorithm under the gobang performance is the best.

Keywords

Game tree; max- min algorithm; alpha-beta pruning; heuristic evaluation.

1. Introduction

Machine game is an important branch and research direction in the field of artificial intelligence, and it is also an important research basis for other research fields of artificial intelligence. The realization of many research works on this basis can greatly improve the intelligent effect of the machine, so it is widely accepted by many scholars. Gobang is a very classic two-player strategy game [2], and the design and implementation of this strategy is what we need to study, and how to efficiently obtain results based on this strategy is worth our in-depth discussion and research, which also triggers another task module-optimization. The focus of this paper is to realize the advancement of the machine implementation strategy of the man-machine game of gobang game, and compares the optimization effect from the implementation of the initial general strategy or method to the implementation of the optimized strategy method. Specifically, max-min calculation [3], alpha-beta pruning search algorithm, and the pruning algorithm with heuristic evaluation [4] and the restriction of chess search range are used to optimize, so that the gobang game is realized step by step, and the superiority of the algorithm is judged by comparing the running time of the three parts. In addition, in order to obtain the comparison effect of this part more conveniently, I designed the interface separately to show the difference of the results of each different search strategy.

2. Algorithm principle and system design

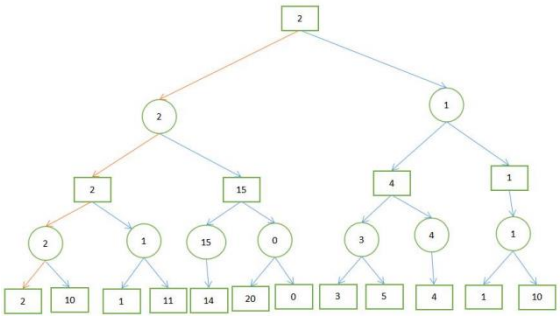
2.1. Search algorithm

2.1.1. The max-min search

For each blank, the AI gets a score based on the type evaluation function. The search process for maximum is to get the position with the highest score, which is called the MAX layer. If it is the player's turn to play chess, the player should select the position where he can get the highest

ISSN: 2664-9640

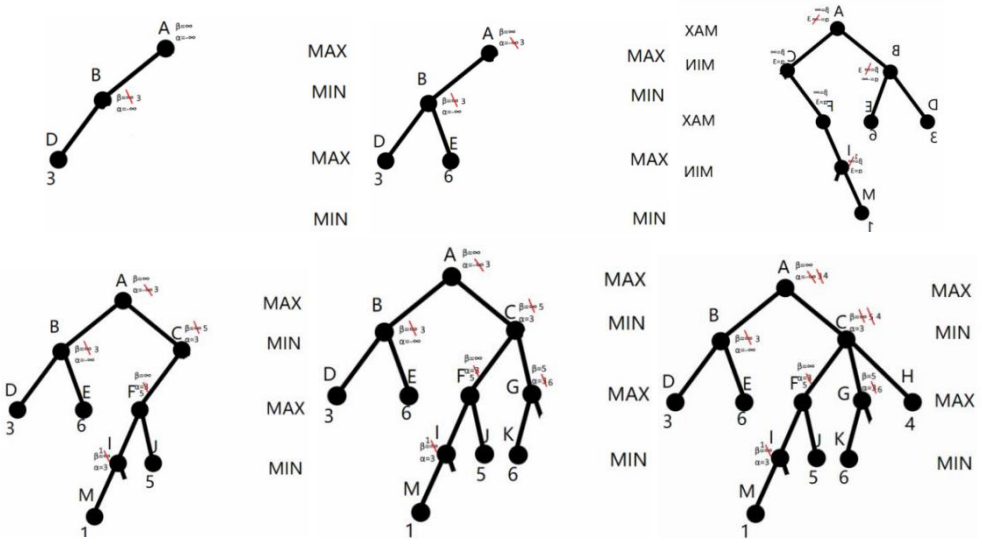
score, namely the position with the lowest AI score, which is called the MIN layer. When it is a multi-layer search, the first layer is the AI playing chess, the second layer is the player playing chess, and so on. Assuming there are n search nodes in each layer, then there are n^n search nodes in the n layer, which will form a huge game tree. If both hands make the best decision, the game will go as follows (orange color direction).



2.1.2. The alpha-beta pruning

The disadvantage of the min-max search algorithm is that when the number of layers of the game tree increases, the number of nodes to be searched increases exponentially. However, some nodes in the actual search that do not need to be searched. The alpha-beta pruning is the pruning of branches in the search tree that do not need to be searched according to the size relationship between α and β values, which can improve the search speed. At present, alpha represents the largest lower bound among all possible solutions, and beta represents the smallest upper bound among all possible solutions. In the process of solving, α and β will approach gradually. For a certain node, the situation of α>β appears, then it means that this point will not produce the optimal solution, we're not going to extend it anymore. The basic principle of the game tree pruning is completed: when the α value of a node in the min layer is less than or equal to the β value, all the unsearched child nodes of the node are pruned. When the α value of a max layer node is greater than or equal to the β value, all the unsearched child nodes of this node are pruned.

Where α value is the current most favorable score of the node in this layer, the β value is the current α value of the parent node, the root node is in the max layer, and the β value is initialized to +∞. If it is the max layer, the α value of the initialization node is -∞, and the score of the child node is greater than this value. If it is the min layer, the α value of the initialization node is +∞, and the score of the child node must be smaller than this value. The following figure shows the pruning process:



ISSN: 2664-9640

2.1.3. Heuristic evaluation

The key to affecting the efficiency of alpha-beta pruning is to prune earlier, so the positions with high scores are searched first. The general pruning is shown in Fig. 2-1. In the second layer part, only the second child node with node value 9 is pruned.

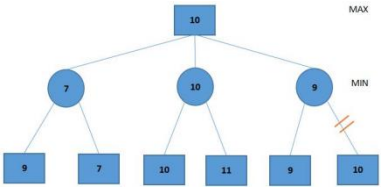


Figure 2-1

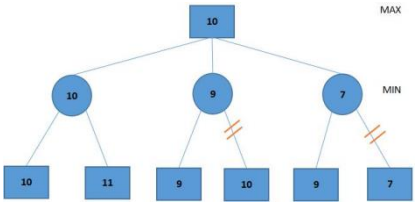


Figure 2-2

After passing the heuristic evaluation, we can first estimate the score of each layer of nodes, and get the game tree in the order of the node values at each layer from large to small. By calling the order of the child nodes, we can prune faster. For example, according to the estimated scores of the child nodes, Fig. 2-2 is the result of re-sorting the game tree in Fig. 2-1. It can be seen that the second child nodes with a node value of 9 and 10 have been pruned, which speeds up the search efficiency. To achieve this, it is necessary to estimate the score for each position that can be played, so that the position with the highest estimated score is ranked first. The estimated scoring method I use is to play black or white for an empty position, obtain the chess patterns that can be formed in the four directions of this point, and then score.

2.2. Scoring function

2.2.1. Drop valuation

```
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
{0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0},
{0, 1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 1, 0},
{0, 1, 2, 3, 4, 4, 4, 4, 4, 4, 4, 3, 2, 1, 0},
{0, 1, 2, 3, 4, 5, 5, 5, 5, 5, 4, 3, 2, 1, 0},
{0, 1, 2, 3, 4, 5, 6, 6, 6, 5, 4, 3, 2, 1, 0},
{0, 1, 2, 3, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1, 0},
{0, 1, 2, 3, 4, 5, 6, 6, 6, 5, 4, 3, 2, 1, 0},
{0, 1, 2, 3, 4, 5, 5, 5, 5, 5, 4, 3, 2, 1, 0},
{0, 1, 2, 3, 4, 4, 4, 4, 4, 4, 4, 3, 2, 1, 0},
{0, 1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 1, 0},
{0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0},
{0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
```

2D array of position scores (the higher the score towards the center)

2.2.2. Setting the chess type scores

Chess	1: Black 2: White 3: Empty chess	Score
Long even	**11111**	**10000**
live four	**011110**	**1000**
rush four	**011112、0101110、0110110**	**1000**
live three	**01110、010110**	**200**
sleep three	**001112、010112、011012、10011、10101、2011102**	**20**
live two	**00110、01010、010010**	**10**
sleep two	**000112、001012、010012、10001、2010102、2011002**	**5**
die four	**211112**	**-10**
death three	**21112**	**-10**

Although the scoring table is not fixed, after many attempts, I choose to use such as scoring value, and to confirm that it is the computer scoring speed and the winning rate effect is a better scoring method under the existing conditions.

2.2.3. Evaluation function

Getting position score: modify the genmove function. In the empty position, the evaluatepointscore function will be called to obtain the score of the chess game formed when you place your own or opponent's pieces, and then add the positions with higher scores to a separate list, which only consider the positions with high scores.

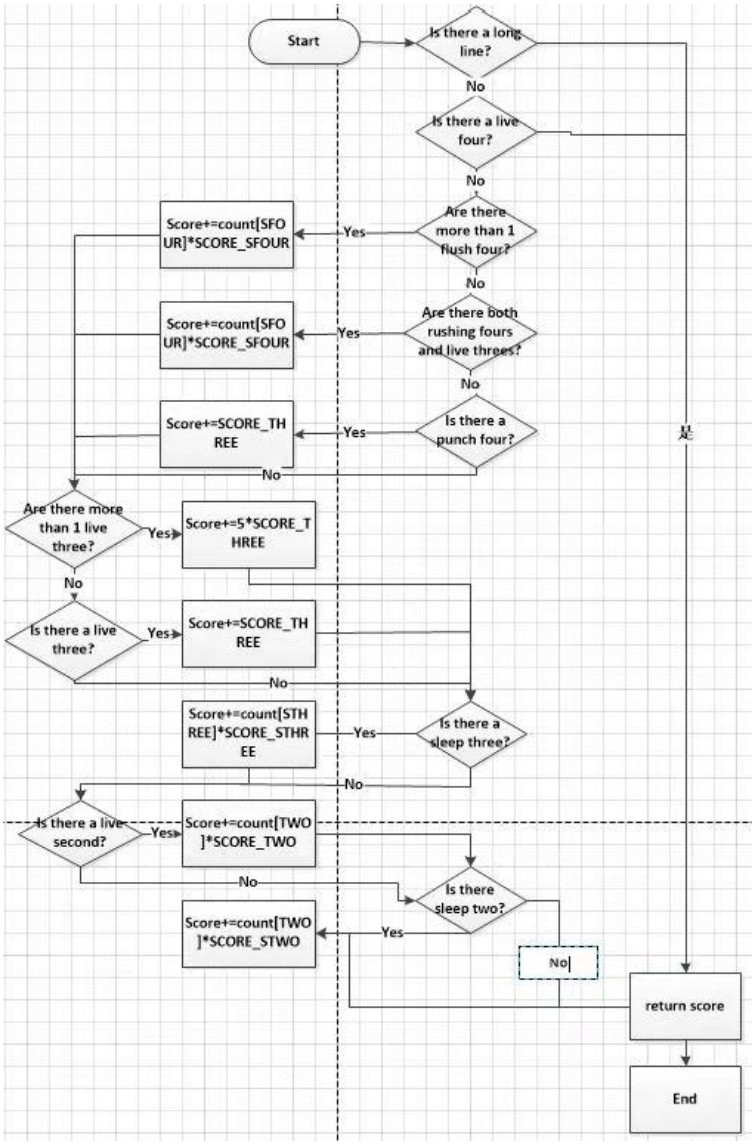
ISSN: 2664-9640

At the end of the function, you can see that there is a limit on the maximum number of positions. We set the value to 10 and only look at the first 10, because we have already evaluated all the available empty positions. After sorting the scores, we remove the last position in advance to reduce the number of nodes searched in the game tree.

The evaluation function at this position: place your or your opponent's pieces separately, search the four directions of this position and get the statistics of the positions by calling evaluatepoint function, and then call getpointscore function to get the score of your or the opponent's pieces at this position.

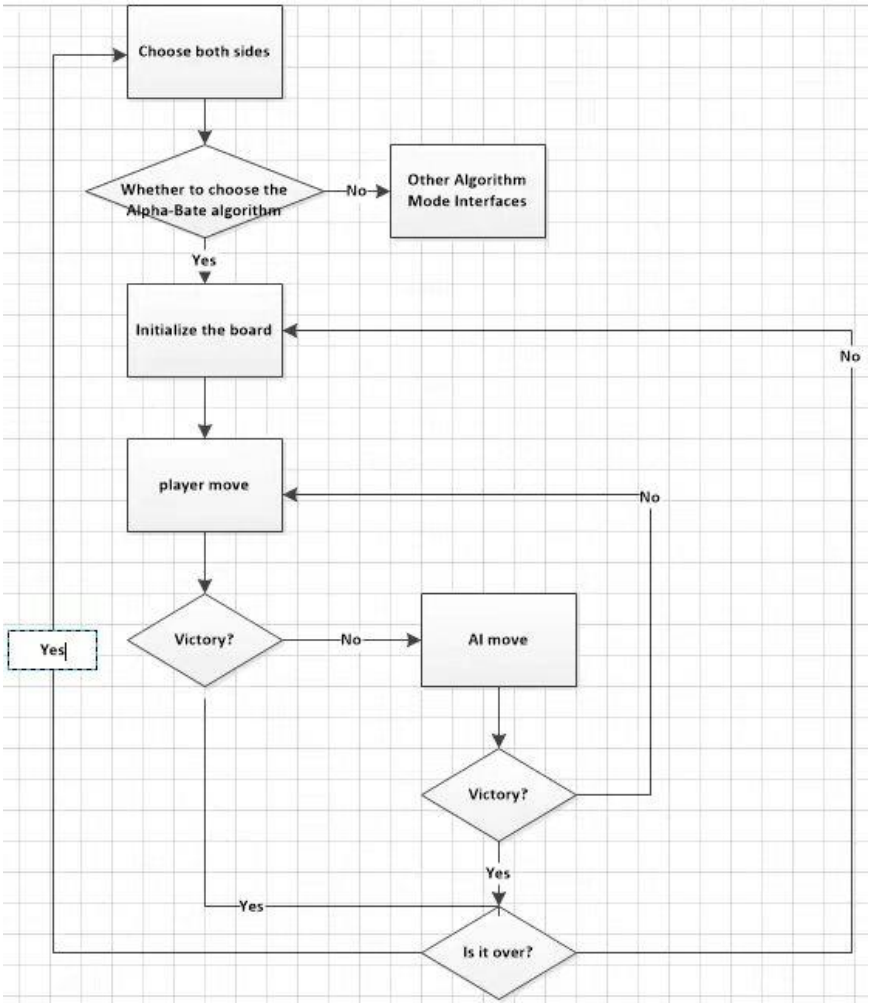
The chess type scoring function in this position: count the scores of all the chess types, pay attention to the score setting of the positions, so the sum of the scores of the chess types with low priority is less than that of the chess types with high priority. In the case of a single strike four, only the live three is given the same score. If there are fives or live fours in the chess pattern, return directly. The chess scores less than or equal to the live three are directly added.

The flow chart is as follows:



ISSN: 2664-9640

2.3. System structure diagram

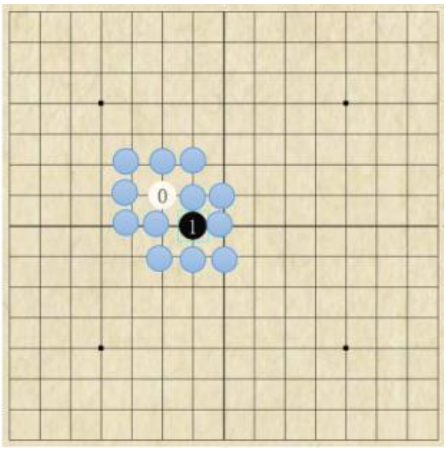


2.4. Optimized Design

2.4.1. Heuristic Evaluation (described in 2.2.3)

2.4.2. Limitting the range of locations to get a drop

The hasneighbor () function is used to limit the range of acquiring pieces, only the empty positions of the outer layers of the existing chess game can be obtained and judged. Each piece that has been played can be obtained in eight directions from this position, and only the empty positions within the limited range can be obtained by ignoring the positions of other pieces that have been dropped. This reduces search time. The explanation is shown in the figure below:



ISSN: 2664-9640

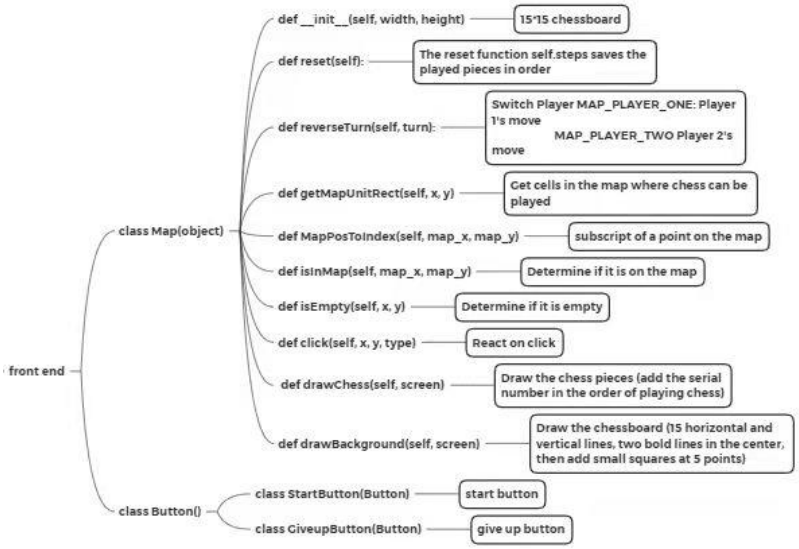
2.4.3. Limit the number of game tree search layers

Because each time AI considers one more layer, the number of nodes searched must be multiplied by the number of nodes in this layer, and limiting the number of search layers will have a certain optimization effect. Because of the lack of consideration of one layer, such AI is not intelligent enough, and the winning rate is also reduced. Therefore, I did not design a few layers, and chose a layer of four based on the balance of the win rate and speed.

3. System implementation

3.1. Front-end design

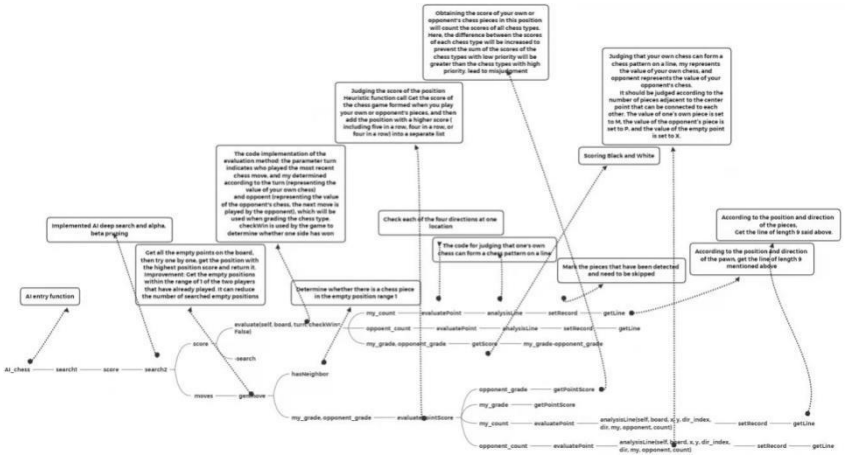
The following is an explanation of the front-end design diagram, using pygame to implement the interface, the implementation of the interface mainly has two functions: drawing the chessboard, drawing the chess pieces, drawBackground draws 15 horizontal and vertical lines with the two bold lines in the middle. The drawChess function draws the chess pieces and numbers them in the order in which they are played. The map class is used to save the data of gobang and provides the function to draw gobang self.map is initialized to a 15*15 chessboard, self.steps saves the played pieces in order, and the Button class is used to draw the button and the text on the button, self.button_color saves 2 colors, indicating the color when the button is enabled or disabled. Subclasses StartButton and GiveupButton have different actions when calling the click function. Only one of the start and Give up buttons can be clicked at a time. When a button is clicked, the unclick function of the other button is called.



3.2. Back-end design

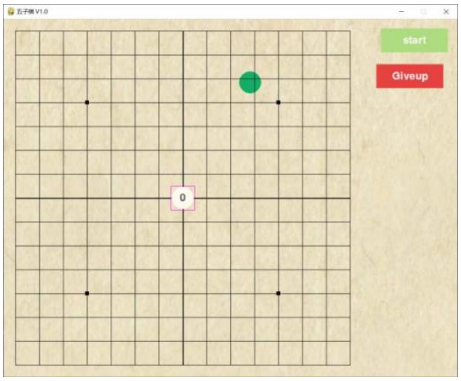
The implementation of AI is in the ChessAI class. The findBestChess function is the entry function of AI, which calls the search2() function to recursively generate the game tree and implement the alpha-beta pruning algorithm. The Search2 function is the code implementation of the AI logic. First, all the empty points on the chessboard are obtained by recursively calling the genmove() function, which calls the evaluatePointScore function to obtain the score of the chess game formed when you place your or opponent's pieces, and then add the position with the highest score to a separate list. The evaluate method is implemented by calling the evaluate function to get the value of the leaf node. Then continue to prune to get the best move position.

ISSN: 2664-9640



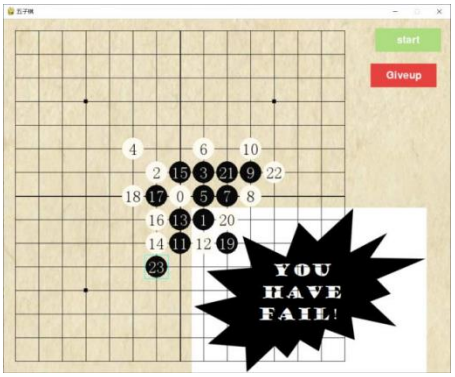
4. Experimental or test results

4.1. No α-β pruning



Without α-β pruning, it is difficult for AI to quickly judge the best move due to the huge game tree.

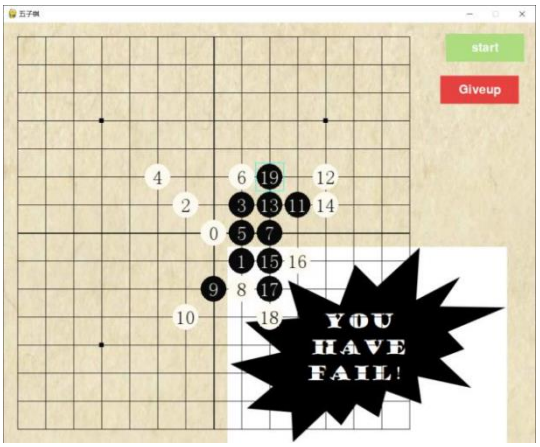
4.2. With α-β pruning




```
time[2.93] (9, 7), score[18] alpha[10571] belta[4319]
time[3.19] (10, 6), score[410] alpha[11053] belta[4221]
time[2.99] (7, 9), score[100000] alpha[8675] belta[3770]
time[1.48] (7, 8), score[100000] alpha[2986] belta[1698]
time[1.56] (7, 6), score[100000] alpha[3181] belta[1780]
time[1.36] (6, 7), score[100000] alpha[2817] belta[1556]
time[1.62] (9, 9), score[100000] alpha[2988] belta[1642]
time[1.51] (9, 6), score[100000] alpha[2817] belta[1494]
time[4.80] (6, 10), score[100000] alpha[9803] belta[4527
```

With the α-β pruning algorithm, the hasneighbor function judged that the empty position range is uneven in time within 1, which is basically 1-3s.

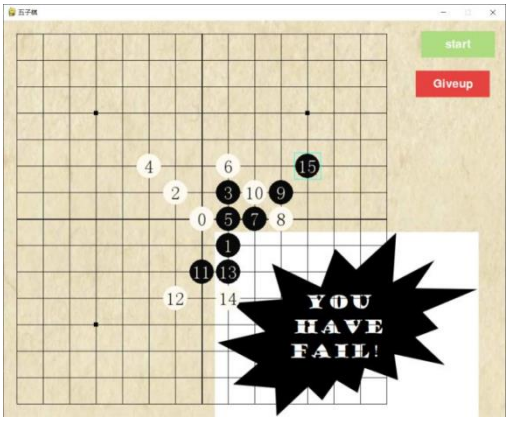
ISSN: 2664-9640




```
time[6.79] (9, 7), score[18] alpha[28027] belta[8955]
time[3.95] (7, 9), score[402] alpha[17989] belta[5190]
time[7.05] (10, 6), score[398] alpha[28363] belta[8635]
time[7.03] (9, 6), score[400] alpha[27662] belta[7626]
time[5.98] (9, 8), score[408] alpha[19402] belta[6111]
time[4.90] (9, 9), score[100000] alpha[17889] belta[477
time[4.21] (9, 5), score[100000] alpha[11662] belta[425

Process finished with exit code 0
```

With α-β pruning algorithm, in the hasneighbor function, the time was not uniform when the empty position was within 2, basically within 4-7s.




```
time[0.83] (8, 8), score[4] alpha[3205] belta[1156]
time[2.74] (8, 6), score[8] alpha[9350] belta[2985]
time[4.30] (8, 7), score[384] alpha[15868] belta[4515]
time[13.68] (9, 7), score[18] alpha[53012] belta[15241]
time[10.52] (10, 6), score[410] alpha[45040] belta[10202]
time[7.63] (7, 9), score[100000] alpha[28322] belta[8542]
time[14.82] (8, 9), score[100000] alpha[40105] belta[16698
time[52.80] (11, 5), score[100000] alpha[109676] belta[503

Process finished with exit code 0
```

With the α-β pruning algorithm, in the hasneighbor function, when it is judged that the empty position range is within 3, the time is not uniform, and the average time exceeds 10s.

4.3. Heuristic evaluation optimization


```
time[0.16] (8, 8), score[4] alpha[308] belta[163]
time[0.31] (8, 6), score[8] alpha[510] belta[253]
time[0.26] (8, 7), score[384] alpha[336] belta[192]
time[0.28] (7, 5), score[16] alpha[312] belta[147]
time[0.45] (7, 3), score[0] alpha[448] belta[197]
time[0.39] (8, 9), score[382] alpha[433] belta[183]
time[0.37] (9, 8), score[6] alpha[324] belta[174]
time[0.37] (10, 9), score[394] alpha[291] belta[167]
time[0.05] (7, 6), score[100000] alpha[9] belta[8]
time[0.01] (6, 5), score[100000] alpha[1] belta[1]

Process finished with exit code 0
```

After adding heuristic evaluation optimization, when the hasneighbor function determines that the empty position range is within 1, the time is between 0.01-0.4, which significantly reduces the time.

5. Conclusion

The topic of our group is the implementation of the gobang game system, so our idea is to intervene in several different AI algorithms through sockets, or the same AI algorithm implemented in different languages, and then choose the first and the rear to play the game. My part is to access the system as an optimized alpha-beta to compete with the other two algorithms. Therefore, everyone's algorithm part is basically implemented separately. In

ISSN: 2664-9640

addition to provide a final optimized alpha-beta algorithm for our group, my part also compares and analyzes the algorithm performance of different levels of advancement, which is reflected through the interface effect.

The first is the game tree that does not use pruning. Because the number of search nodes is exponential, the AI is slow to settle down, and then after adding the pruning algorithm, the search speed and the drop speed of the game tree are accelerated when a large number of nodes are cut off. Finally, heuristic evaluation is used to optimize the pruning algorithm. By learning its principle, I know that the search order of each layer of nodes during pruning search will also affect the efficiency of pruning. For example, high rated sites can be searched earlier, so prune faster. It avoids searching for other nodes in the same layer, because the value of other nodes is definitely smaller than the value of the previous node, and the pruning efficiency is improved. However, there is a problem that the intelligence of AI is weakened, and its long-term thinking ability is reduced. When expanding the search tree, only consider these drop points as new layer nodes, and carry out the next search, and other optimal solutions may be lost. If I use other smarter AI to play with my AI, my AI may fail because I don't have a long-term consideration to determine the position of the empty pieces. In addition, I also try to consider other optimization methods, such as searching empty position in the formation of board type, reducing the range of the position judgment of the move, limitting the range by hasneighbor. Only judge the position of one or two layers adjacent to all the pieces in the current chess game, that is, the position of up and down, left and right, and four azimuth angles, and do not judge other positions, which also has obvious performance improvement.

In addition, I also saw other optimization methods by consulting the paper, such as using a transition table to store the search results of the nodes that have been searched, including the game value of the subtree, the best move and position. By using the transition table to narrow the search window. I have tried adding this method to my algorithm design, and I encountered difficulties in the parameter association, so I did not use the optimization of this part of the transfer table. If time permits, the optimization of this part will be considered in the future to compare the superiority of the game with other algorithms. By doing the big homework of the gobang game system, I have gained a lot, tried and learned a lot of related algorithms, this is a step-by-step review, a step-by-step understanding and analysis of the last little digestion process. Understanding and learning the intelligent algorithm behind gobang will be very helpful for me to learn other intelligent algorithms in the future.

References

[1]Li Hao. Research and implementation of gobang man-machine game algorithm optimization [D].Dalian maritime university, 2020.DOI:10.26989/d.cnki.gdlhu.2020.000523.

[2]Tian Xiwen. Research on human-machine game parallel robot system [D]. Tianjin vocational and technical normal university, 2019.

[3]Zheng Jianlei, Kuang Fangjun. Research and implementation of gobang intelligent game algorithm based on minimax search and alpha beta pruning algorithm [J]. Journal of wenzhou university (natural science), 2019,40(03):53 -62.

[4]Wang Xiukun, Liu Jiannan. Adaptive genetic algorithm for optimizing game problem evaluation function parameters [J]. Computer engineering and applications, 2009, 45(20): 42-44+51.

My division of labor: interface design, alpha-beta pruning search, algorithm heuristic optimization.