Fall 2019

New heuristic algorithm to improve the Minimax for Gomoku artificial intelligence

Han Liao

New Heuristic Algorithm to improve the Minimax for Gomoku Artificial Intelligence

by

Han Liao

A report submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Joseph Zambreno, Major Professor

Iowa State University

Ames, Iowa

2019

TABLE OF CONTENTS

Page

LIST OF TABLES iii

LIST OF FIGURES iv

ACKNOWLEDGMENTS v

ABSTRACT vi

CHAPTER 1. INTRODUCTION 1
- 1.1 Background 1
- 1.2 Related Work 2
  - 1.2.1 Minimax Algorithm 2
  - 1.2.2 Alpha-Beta Pruning 4
  - 1.2.3 Other Algorithm 5
  - 1.2.4 New Heuristic Algorithm 7
  - 1.2.5 Greedy Algorithm 8

CHAPTER 2. GUI and System construction 9
- 2.1 GUI and functions 9
- 2.2 Back-end support 11
  - 2.2.1 Board Evaluation 11
  - 2.2.2 Opening Book 13

CHAPTER 3. New Heuristic Algorithm 14
- 3.1 Candidates Selection 14
- 3.2 Best Path 16

CHAPTER 4. EXPERIMENT AND RESULTS 17
- 4.1 Algorithm Competition 17
- 4.2 Parameter Impact 20
- 4.3 Example 21

CHAPTER 5. CONCLUSIONS AND FUTURE CHALLENGE 23

BIBLIOGRAPHY 24

LIST OF TABLES

Page

4.1 Black First Advantage 17
4.2 Comparison with *Greedy* 18
4.3 *New Heuristic* vs *Minimax* 18
4.4 Depth Comparison 20
4.5 Candidate selection range 21

LIST OF FIGURES

		Page
1.1	Alpha-Beta Pruning	5
2.1	GUI	10
2.2	Simulator	10
2.3	Stone shapes	11
2.4	Opening book	13
3.1	Evaluation Range	15
3.2	New Evaluation Range	15
3.3	Flowchart of the back-end	16
4.1	Computation numbers	19
4.2	Computation number	20
4.3	Minimax Example	22
4.4	New Heuristic Example	22

ACKNOWLEDGMENTS

I would like to take this opportunity to thank those who have contributed to my research and thesis. Firstly, I would like to express my sincere gratitude to my major professor Dr. Zambreno for the continuous support of my master's study and research. He led me to identify a research objective and gave a lot of guidance on many decision-making aspects. I feel very honored to have such a patient and experienced advisor to guide me.

Besides my advisor, I would like to thank my best friend, MengYao, who put forward some constructive suggestions on my research project. My sincere thanks also goes to David, who provided his equipment in the experiment, which helped me to get perfect test results.

Last but not least, I would also like to thank all of my friends and family for their support and financial assistance.

ABSTRACT

Back in the 1990s, after IBM developed Deep Blue to defeat human chess players, people tried to solve all kinds of board game problems using computers. Gomoku is one of the popular board games in Asia and Europe, people also try to simulate and solve it through computer algorithms. The traditional and effective strategy for Gomoku AI is the tree search algorithm. The minimax algorithm is one of the most common game trees used in AI strategy. But obviously, the biggest problem with this is that as the number of stones on the board and the number of search depth increases, even computers have to spend a lot of time calculating each step. The number of nodes with exponential increment is really difficult to achieve in practical terms. In the paper, we will discuss in detail how to improve the most basic minimax algorithm. The direction of the research is on how to improve the efficiency of the algorithm and maximize the search depth. The most common means used now is to cut out the clutter in the search tree through Alpha-Beta pruning. Moreover, we offer a new heuristic algorithm which can boost the depth search processing a lot. The core idea is how to sort and reduce the potential candidates for each depth and nodes, and how to return the best path in a recursive way. We finally will compare and compete with the traditional minimax algorithm and New Heuristic minimax algorithm in the experimental testing session. Based on the API developed individually, this paper will explain back-end algorithms and the program user interfaces itself in detail.

CHAPTER 1. INTRODUCTION

1.1 Background

The Gomoku originated in the Heian period in Japan. It is a board game with a strong logic strategy and focuses on reasoning and calculation. Its rules are simple, with both sides of the competitor taking turns playing with black and white stones at 225 empty intersections. It follows the rule that the black side always moves the first stone. People usually play Gomoku on a 15 x 15 board, and usually, the first move falls in the center of the board, which causes the whole game to unfold around the center of the board. The winner is the player who forms an unbroken chain of five pieces horizontally, vertically, or diagonally. Allis (1994) came up with an algorithm that mathematically proved Gomoku is a solved game. But in computer science, people are still figuring out how to create more offensive and strategic artificial intelligence. Gomoku is a board game based on precise mathematical calculations. Correspondingly, Gomoku and Go are very similar from their appearance, but because Go's uncertainty rules are completely beyond the current level of human cognition, Go as a whole does not fall into the category of accurate calculations. Gomoku's rules are therefore easier to implement at the code level through the fundamental algorithms rather than using neural networks.

There are some famous tournaments for Gomoku AI program since 1989. Although the deep learning algorithm is very popular in recent years, people still believe that given the precise calculation of the Gomoku problem, it doesn't need to use deep learning, which is an accumulated big data, to solve. Junru Wang and Lan Huang (2014) proposed a genetic algorithm that can search in deeper depth than a traditional game-tree. Therefore, programs with big data learning ability like Alpha Go are not allowed to be used in most competitions. At present, the most commonly used algorithm is the game search tree. Minimax is one of the most commonly used algorithms, but because of the differences in each developer's board evaluation system and search depth, this leads to differences in the gaming level at the end.

As we mentioned, there exists a perfect solution for the black side to get a sure-win in the free-style rule. To ensure the fairness of the game, standard rules have been created in this perception. Standard rules include the specificity of the opening rule and the prohibition of some special stone shapes. So this research program we developed is based on the standard rules to ensure fairness.

1.2 Related Work

1.2.1 Minimax Algorithm

Minimax is a computer algorithm that is widely used in board games. As long as it is a competitive game with two players, minimax's basic guidelines are appropriate. In the two-player competition game, everyone wants to maximize their own benefits in every decision. So the maximum value for the current player is the highest value he can get when his opponent always makes the best actions.

In Minimax, we actually named the two players maximizer and minimizer. Maximizer always wants to get the highest score while the minimizier tries to get the opposite lowest score. This is also completely consistent with our practical experience. So in each search, the algorithm stand by the side of the current player's maximization of benefits. Each "best action" output from the algorithm is not the maximum under the current situation, but the best choice to make after the trade-off, considering that the opponent will minimize the benefits to current player. According to Wikipedia (2019), its formal definition is $v_i = \max_{a_i} \min_{a_{-i}} v_i(a_i, a_{-i})$ where i is the index of the current player, $-i$ denotes all other players except player i, a_i is the action taken by player i , a_{-i} and v_i is the value function of player i.

Vardi (1992) mentioned minimax's decision-making characteristics are not probabilistic, because the output results are macro-derived to evaluate the expected value and utility. It does not analyze the probability of each outcome, but only predicts and calculates every possible scenario. The pseudo code is shown below:

Algorithm 1 Minimax

```
function MINIMAX(position, depth, maximizingPlayer)
    if $depth == 0$ or game over in position then
        return static evaluation of position
    if maximizingPlayer then
        $maxEval = -\infty$
        each child in position
        $eval =$ Minimax(child, depth-1, false)
        $maxEval = \max(maxEval, eval)$
        return maxEval
    else
        $minEval = +\infty$
        each child in position
        $eval =$ Minimax(child, depth-1, true)
        $minEval = \min(minEval, eval)$
        return minEval
```

This is a recursive function, consisting of three parts. The first part is the bottom output, the condition is that the number of search depth reached the target, or the game has ended. The second and third parts solve the maximum value of the current player and the optimal solution of the opponent respectively. In the pseudo, the *maximizingPlayer* is a boolean variable. When the current depth is odd (*maximizingPlayer* == **true**), meaning that is the move for the first player, it will be graded based on the established board evaluation equation. Conversely, if it is even depth (*maximizingPlayer* == **false**), that is the move for the second player, it will find the node in the current situation that is most unfavorable to the first player. In fact, it is easy to find that this algorithm is still an exhaustive solution process, because if the algorithm does not make any changes, it will return the final answer after searching all the possibilities. So we need to explore how to improve and accelerate this process more efficiently.

1.2.2 Alpha-Beta Pruning

Alpha-beta pruning is an advanced algorithm that can reduce the number of nodes in Minimax. In the minimax algorithm, we talked about how the game tree searches all the possibilities to reach a final result, but in this process, many nodes do not really need to do a deep search. alpha-beta pruning will stop evaluating a move when it reaches a node that is worse than previously examined. The process then routinely computes the remaining nodes over the nodes that are currently abandoned. When we apply the standard minimax algorithm, alpha-beta returns the same result as minimax would, but it cut-off some branches that cannot influence the final decision. The pseudo-code is shown below:

Algorithm 2 Alpha—Beta Pruning

```
function MINIMAX(position, depth, alpha, deta, maximizingPlayer)
    if $depth == 0$ or game over in position then
        return static evaluation of position
    if maximizingPlayer then
        each child in position
        $eval$ = Minimax(child, depth-1, false)
        $maxEval = \max(maxEval, eval)$  $alpha = \max(alpha, eval)$
        if $beta \leq alpha$ break
        return maxEval
    else
        each child in position
        $eval$ = Minimax(child, depth-1, true)
        $minEval = \min(minEval, eval)$  $beta = \max(beta, eval)$
        if $beta \leq alpha$ break
        return minEval
```

Two values, alpha and beta, are introduced in this algorithm. They represent the maximum score of the player currently needed to be moved and the minimum score of the opponent. The initial value of alpha is negative infinity, whereas beta is positive infinity. When an opponent's minimum score is lower than the current player's maximum score, then we can stop the process because they will never be able to appear in the actual game. In Nasa (2018)'s paper, which also implemented minimax and alpha beta, and then compared their experimental data. It was clear that in their results, alpha-beta had a faster response speed.

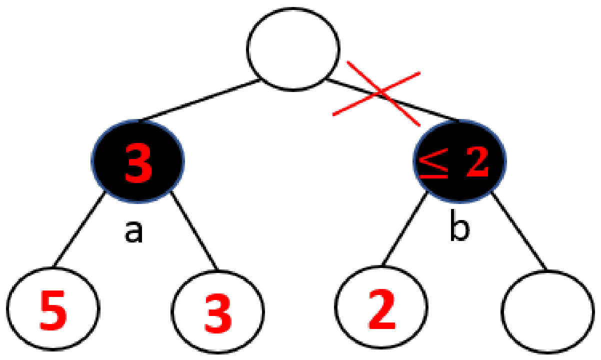


Figure 1.1: Alpha-Beta Pruning

In figure 1.1, we give a simple example. Under the first depth search rule, we already examined the value of node *a* is 3, and node *b* is less than or equal to 2, then this satisfies the rule we described that alpha, the maximum value of the opposing player is less than the value beta. Then we can completely abandon the branch of node *b*.

1.2.3 Other Algorithm

Articial intelligence is the hottest topic in the eld of computer, and human has made great progress after 50 years of research and exploration. In solving the board game that human beings have been engaged in for thousands of years. In the book written by Ian Milling (2009) has elaborates and lists some games which can be solved by simple algorithms. Articial intelligence has been vigorously developed and studied. From "deep blue" in the last century which completely surpass human in the eld of chess, to Alpha Go developed by Google company in 2016 thoroughly surpass human's Go level, and the rise of self-driving cars, humans seem to be getting better at articial intelligence. But research on basic algorithms has never stopped.

In the early last century, Vardi (1992) introduced the most basic algorithm such as game tree and minimax to solve some relatively easy board games, such as the simplest tic - tac - toe and checkers. This kind of board game has an obvious characteristic that is able to use computer fast computing, data storage and processing ability to estimate the next moves. The alpha-beta algorithm can enumerate all the possibilities and select the next step through the inherent checkerboard estimator. Based on the proposal of pruning by Donald E.Knuth (1975), the computing speed of alpha-beta is accelerated, and the unnecessary branch of search is simplified, which greatly accelerates the processing. Based on this, minimax even has its unique advantages and can be more efficient than alpha-beta in the sense that minimax never evaluates a node that alpha-beta can ignore G.C.Stockman (1979). Vardi (1992) proposed a new minimax algorithm. The difference lies in the use of slack variables to handle inequality constraints the conditions and has a trust-region strategy taking advantage of the structure of the problem.

For some board games, such as chess, the alpha-beta algorithm is no longer available. Building an evaluation function based on heuristic knowledge for a non-terminal position is a difficult and time-consuming issue in those problems. The Monte Carlo Search Tree (MCT) made it possible for computers to surpass humans. The landmark development was that in 1997, the artificial intelligence Deep Blue developed by IBM thoroughly beat the acknowledged excellent chess players in the world. The algorithm used MCT by Philipp Rohlfshagen and Colton (2012). Coulom (2007) concluded three main steps in MCT: constructing the probability, implementing the sampling of probability distribution and establishing various estimates. Guillaume Chaslot and Istvan Szita (2008) proposed using the MCT framework to prevent explorations of the search space from being used to predict the most promising game actions. They believe that MCT can be effectively used in most board games and computer games.

But for some complicated board games, even MCT can't be simulated and evaluated. That is why Go has been considered the last bastion of human intelligence in front of computers. Even so, the emergence of neural networks broke the deadlock by Cooper et al. (2008). Convolutional neural network and multidimensional recurrent network are steeped separately using high - level human games by Rongxiao (2016). Similar to Gomuku, Zhentao Tang et al. (2016) combined with the Monte Carlo search tree and adaptive dynamic programming which can strengthen neural network training. Prior to Alpha Go's final test, Silver David (2016) published a formal notice in Nature announcing that Go would be resolved in a new way, claiming that the value networks would help to evaluate board position and policy networks to select moves. Alpha Go beat Lee Sedol a few months later.

Although humans seem to have lost out on board games. But the algorithm research has never stopped. Combination with MCT and DNN Reinforcement learning, Silver David (2017) propose a system that trains machine without the prior human skills. This has been achieved on Alpha Zero in 2017, developed by Anton (2018). Seems like a neural network with MCT has been able to solve most of the board game, we hope to have another perspective to achieve Gomoku. The different between Gomuku and Go is, the changes in Go is more complex and unreliable, but Gomuku is still a game need to precise calculation, the choice of each move must be necessary and powerful. Indeed, scholars Guillaume Chaslot and Istvan Szita (2008) used neural networks to solve Gomoku. But we believe this approach combines the characteristics of Gomuku is too complex. Therefore, we will improve Gomuku AI by improving minimax or matching the new checkerboard numerical analyzer.

1.2.4 New Heuristic Algorithm

The main content of this paper is the new heuristic algorithm. On the basis of minimax and alpha-beta, we will select and sort potential candidates through the specific nature of Gomoku. This will greatly reduce the number of nodes in each step of exploration. Second, we want to build a database similar to the hash table to store the values of all possible best paths. It will greatly speed up the process of having to recalculate each move as the game progresses.

In theory, minimax can be used without setting the number of search depth, so that it searches the entire board extensively until it finds the optimal solution. But obviously it is impossible for Gomoku. Gomoku and chess are not the same, there will be fewer and fewer pieces in chess, and the number of pieces on Gomoku will only be more and more. So we often don't end up with the game when judging the number of search depths. That's why the development of artificial intelligence in Gomoku is limited. On a typical PC using the most basic minimax, the maximum search depth that can be reached is 3, based on the premise that it can still play normally. We need to study and discuss how to ensure smoothness while also maximizing the scope of the search.

1.2.5 Greedy Algorithm

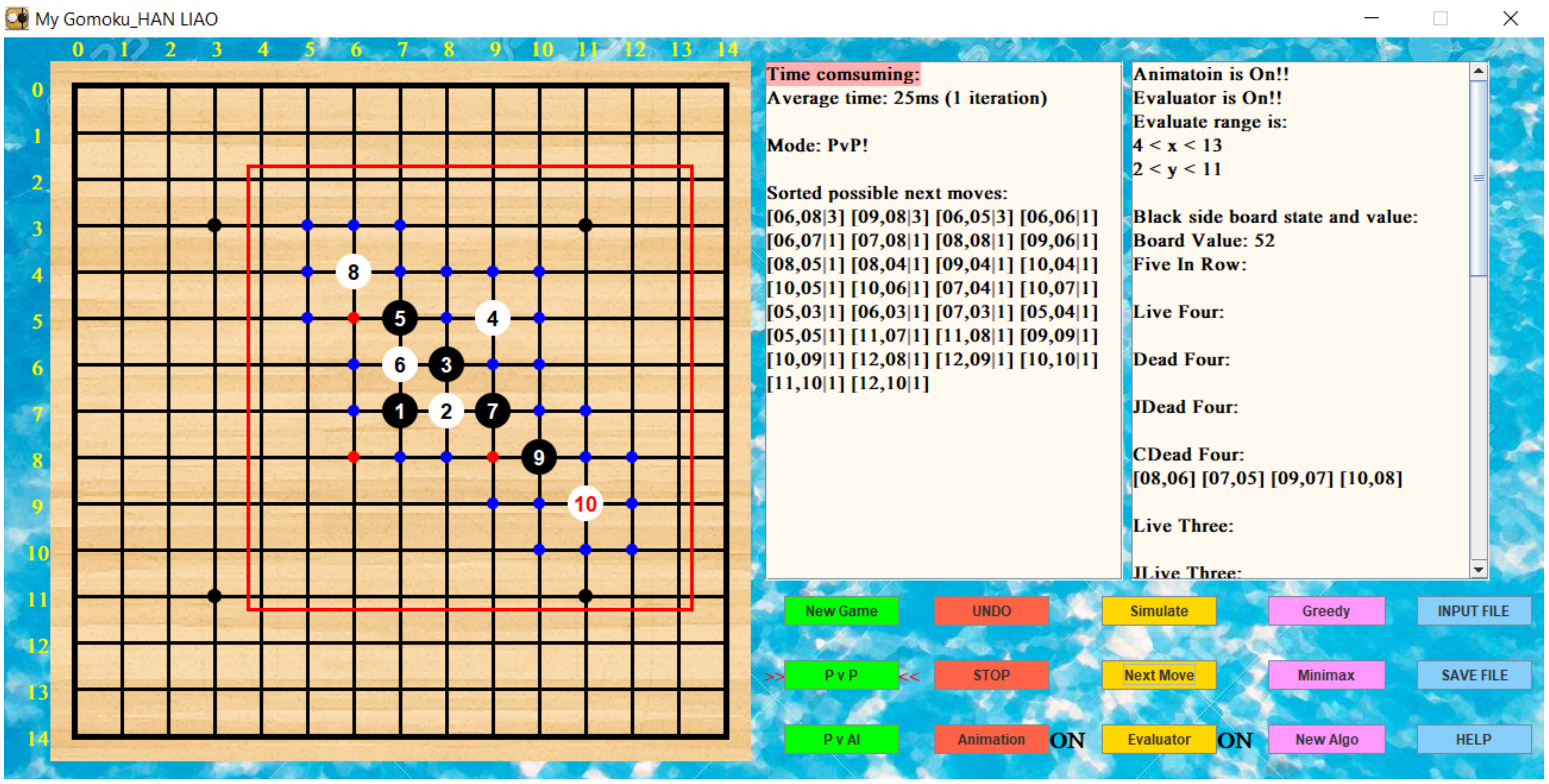
We will just briefly introduce the greedy algorithm. When we play a video game, it always allows us to choose different difficulty level. Based on this idea, we added a simple greedy algorithm to the test session. It will score black and white both sides separately while reading the current board score. For example, if the current moving turn is known to be black, the checkerboard score for both black and white is evaluated. If the black score is greater than the white side, the position of the most beneficial black side will be found in the board. If the white score is greater than the black party, the current player is at a disadvantage. The maximum score position for the corresponding white party is found, and then the black stone is dropped here. Simply put, the algorithm chooses between offense and defense, and then find the position that is most favorable to current player. As one of the simplest algorithms, we will use it as one of the most basic experimental subjects to test more advanced algorithms during the test phase.

CHAPTER 2. GUI and System construction

2.1 GUI and functions

We used Java.Swing library to develop the windows of the user interface. In figure 2.1, there are three parts in this user interface: board, data display area, and function buttons. The left side is a standard 15*15 board, and the right side contains two data readouts that display some data based on the current board situation. For example, the numerical evaluation and details of the current checkerboard, the position of candidates selected according to certain rules, the time required by the current AI calculation and the number of times the algorithm functions were calculated, etc. The current situation can be clearly seen on the board, and those small blue or red dots generated at some empty intersections, which are intuitive to show the probability of the next move of the potential candidate mentioned earlier. We have a total of 12 functional buttons, including board reset, mode conversion, undo, animation and stop, data testing, three different algorithms, plus some auxiliary functions such as reading and writing files. These features play a vital role in the test phase. Shown in Figure 2.2 is the interface of a test system that requires the user to select the corresponding algorithm for both black and white. There are three parameters that can be modified in the New Heuristic algorithm, which include search depth, enabling potential candidates, and range. After entering the name of the experiment report and the number of experiments, you can click the OK button to start the simulation and get the final report.

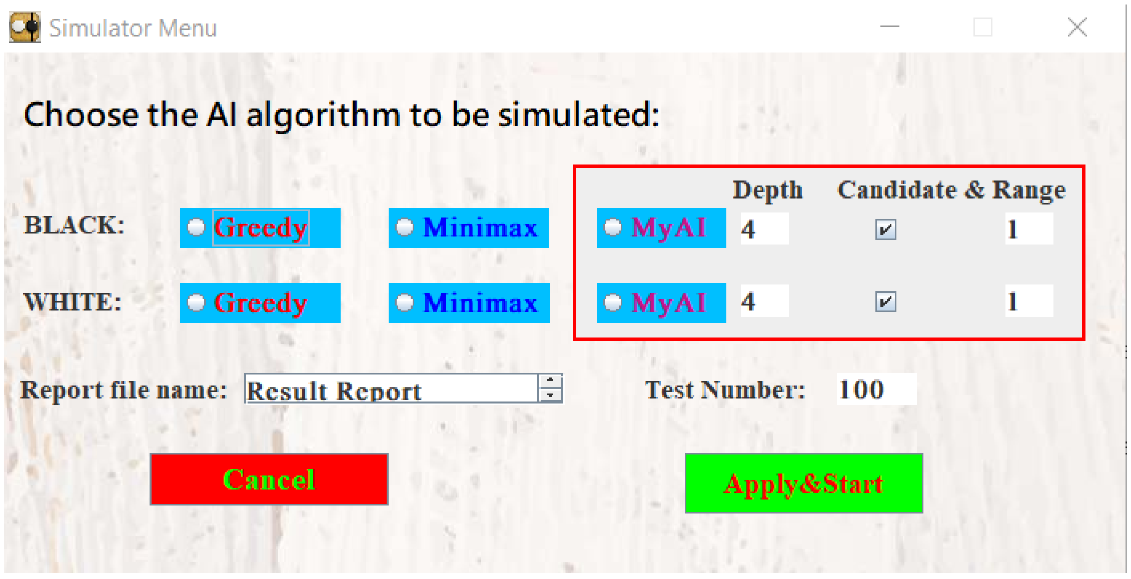
Gomoku is a game that requires precise calculations, and we simply to study the algorithm ourselves, it is difficult to intuitively infer the quality of the algorithm from the huge sets of test data. Without a clear and versatile interface, even later numerical tuning and debug can become very difficult. Although half of the code in the entire program is about user interface development, a complete, clear interface ensures that I am helpful in the later algorithmic adjustment and testing phases.



```
Time comsuming:
Average time: 25ms (1 iteration)

Mode: PvP!

Sorted possible next moves:
[06,08|3] [09,08|3] [06,05|3] [06,06|1]
[06,07|1] [07,08|1] [08,08|1] [09,06|1]
[08,05|1] [08,04|1] [09,04|1] [10,04|1]
[10,05|1] [10,06|1] [07,04|1] [10,07|1]
[05,03|1] [06,03|1] [07,03|1] [05,04|1]
[05,05|1] [11,07|1] [11,08|1] [09,09|1]
[10,09|1] [12,08|1] [12,09|1] [10,10|1]
[11,10|1] [12,10|1]
```

```
Animatoin is On!!
Evaluator is On!!
Evaluate range is:
4 < x < 13
2 < y < 11

Black side board state and value:
Board Value: 52
Five In Row:

Live Four:

Dead Four:

JDead Four:

CDead Four:
[08,06] [07,05] [09,07] [10,08]

Live Three:

JLive Three:
```

Figure 2.1: GUI



Choose the AI algorithm to be simulated:

				Depth	Candidate & Range	
BLACK:	Greedy	Minimax	MyAI	4	✓	1
WHITE:	Greedy	Minimax	MyAI	4	✓	1

Report file name: Result Report   Test Number: 100

Figure 2.2: Simulator

2.2 Back-end support

2.2.1 Board Evaluation

How to help AI better understand and read the current board situation is very important. Therefore, it is necessary to establish a complete board data analysis system. In this regard, every Gomoku AI has a different board numerical evaluation system, we are not prepared to refer to other people's systems at this point, because it is actually related to my own understanding of the Gomoku game itself. A total of 20 basic chess shapes have been summarized, taking into account Zheng Peiming (2016) test results in board evaluation and combining my own personal experience. They are shown in figure 2.3.

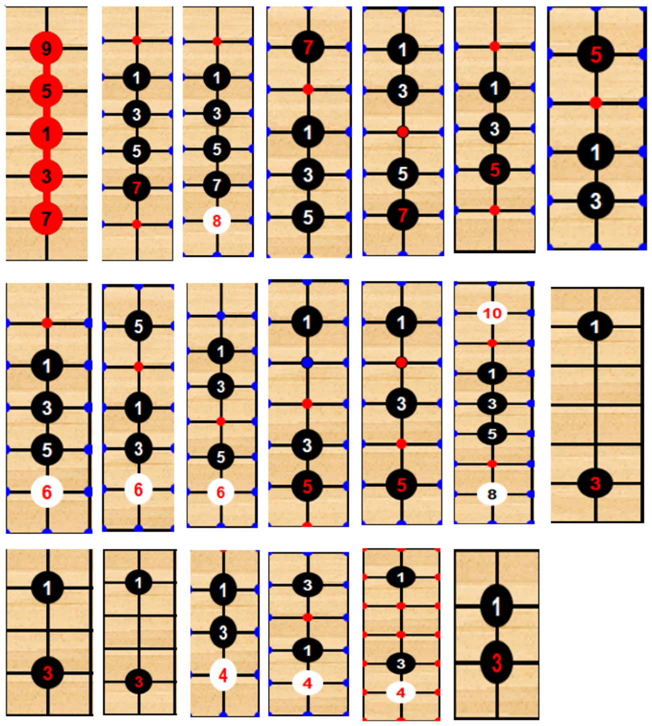


Figure 2.3: Stone shapes

There are 7 categories of stone shapes:

$$FiveInRow, LiveFour, DeadFour, LiveThree, DeadThree, LiveTwo, DeadTwo$$

The priority of these seven categories is also decreasing. The highest priority is $FiveInRow$ that five stones connected into a line. The number word before each type refers to the number of stones on the current line, including vertical, horizontal, and diagonal lines. The *live* shape indicates that the next step must be raised to a higher priority. For example as $LiveThree$, no matter what the opponent's action, we can let it rise to at least $DeadFour$ level or even higher level $Livefour$.

No matter how complex the current chess game is, after analyzing all the stones, the program will generate a table containing the number of 20 stone shapes. But the most important step is that we need to digitize these chess forms, an accurate and effective representation of the score in the current board form plays a vital role in the operation of the algorithm. In this regard, after a lot of testing and research, the program will follow the following rules to score these 20 chess forms:

Algorithm 3 GetBoradEval()

```
1:  if $f(\textbf{FiveInrow.size()}! = 0)$ then
2:      $\textbf{BoardEval}+ = 100000$
3:  if $f(\textbf{Livefour.size()} == 1)$ then
4:      $\textbf{BoardEval}+ = 15000$
5:  if $f((\textbf{Livethree.size()} \geq 2)||(\textbf{Deadfour.size()} == 2)||(\textbf{Deadfour.size()} == 1\textbf{Livethree.size()} == 1))$ then
6:      $\textbf{BoardEval}+ = 10000$
7:  if $f(\textbf{Livethree.size()} + \textbf{jLivethree.size()} == 2)$ then
8:      $\textbf{BoardEval}+ = 5000$
9:  if $f(\textbf{Deadfour.size()}! = 0)$ then
10:     $\textbf{BoardEval}+ = 1000$
11: if $f(\textbf{JDeadfour.size()}! = 0)$ then
12:     $\textbf{BoardEval}+ = 300$
13: if $f(\textbf{CDeadfour.size()}! = 0)$ then
14:     $\textbf{BoardEval}+ = (\textbf{CDeadfour.size()} * 50)$
```

2.2.2 Opening Book

The most powerful Gomoku artificial intelligence Yi Xin is developed by Kai (2017), who beat the world's best human player in 2018 and has won four consecutive Gomoku competitions. It has to be admitted that it is a very efficient and high-precision calculation program, but in the first four steps of the calculation spending a lot of time. This is because the game tree is searched and calculated based on the current state of the board. But if it's a blank board, minimax is actually making a fuzzy attempt, which is why Yi Xin's performance in the first four steps was poor, but as the game progressed, the situation on the board became clearer and Yi Xin became more and more effective. We analyzed a large number of game histories from pro Gomoku players and considering the limitations under the standard rules, found a total of 57 styles in the opening book. In figure 2.4, because of the limitation of the paper space, there are only 12 shown here.

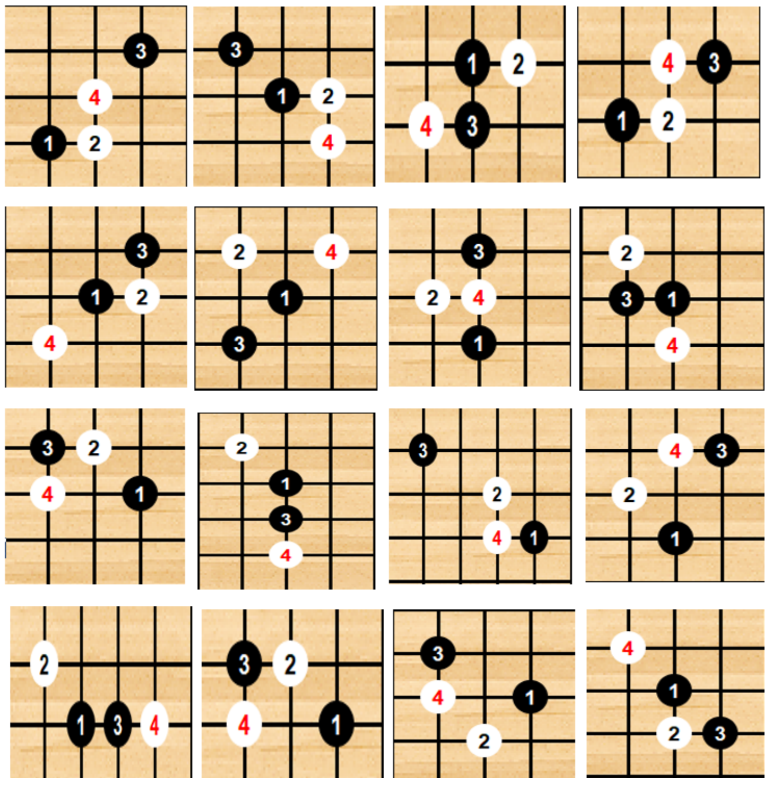


Figure 2.4: Opening book

CHAPTER 3. New Heuristic Algorithm

3.1 Candidates Selection

Minimax is a recursive equation whose essence is a tree node diagram. So at the top of the treemap is the result we need to return. Although we explained earlier how to simplify this process with alpha-beta pruning, experiments have shown that in most cases it does not accelerate much. The results of this experiment will be shown in a later test chapter. So can we consider optimizing this algorithm by reducing the number of nodes itself? That's how to filter out the advantages of a candidate. In the previous program, I was filtering within a certain range. As shown in the figure 3.1.A certain range ($3 \leq x \leq 12, 2 \leq y \leq 11$) is marked with a red straight line on the board. The original evaluation range is on the outermost side of the current stone position as the coordinates, extending 2 units outwards as the search range. Then it's easy to do with the code, and two for-loops can search for all the unoccupied position in this square area. But it's clear that this is a waste most of the time, for example, in the figure 3.1, the first point to search will be the upper left corner $(3, 2)$, and this is a very bad position because it has too little influence on the entire board. It is conceivable that at the very beginning we wasted a lot of time searching in a position that could not have been possible. In fact, if only with a square to define the possible search range, then according to different circumstances, the stone shape will not be static, so as in the example, area A and area B marked out most of the positions are not practical. But algorithms can't tell the difference, and computers don't define the current board shape as clearly as humans do. So we need to take a different approach to define the search range.

So based on the analysis, we need a search range that fully fits the current board condition. This allows us to narrow down the treemap in Minimax, and since the size of the tree plot depends on the depth of the search and the subset of each node, we can minimize the subset of each node, which will greatly reduce the time of the entire algorithm. As shown in the figure 3.2, The figure is represented by a small blue and red dots that represent the searchable range in the current board condition, although the searchable range will change as the number of search depths deepens. But it is clear that the new evaluation range is significantly less than before and is fully fit for the stones on the board. Specific test comparison data will be described in the next section. One of the minimax's biggest problems is that it is random in selecting candidates, and if we can sort potential candidates before expanding the game tree, then we can speed up the process when combined with pruning. To this, I sort the filtered candidates through the new way. For chess, pieces on the chess board have been moving, and the number of stones is constantly decreasing; For Go, although stones can not move, the number of stones is also changing. But compared to Gomoku, it has its uniqueness. Each stone is fixed and cannot change once its position is selected, and there is no phenomenon of removal. So in terms of outcomes, Gomoku's final outcome is closely related to every previous pieces. So can we sort potential candidates based on the results? According to the rules, the winner is the side who has five stones connected into a straight line. Then for each set of pieces of the same color connected on the board, it can be understood that it may become the final five connected. Then those potential candidates can be based on adjacent known stones to calculate the probability. In fact, the algorithm is not complex, if there are 4 stones even a line, then the end of the unoccupied position has 100% chance. If there are three, then 80%, and so on. Some of these computational details need to be used in the board evaluation system described in chapter 2.

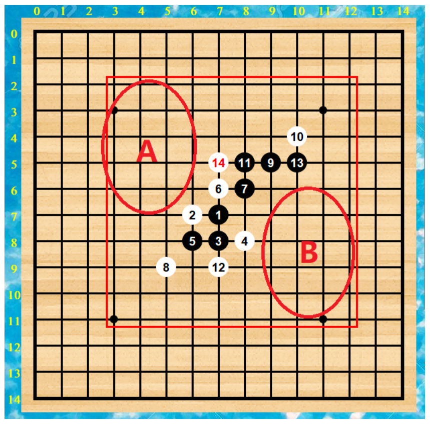


Figure 3.1: Evaluation Range

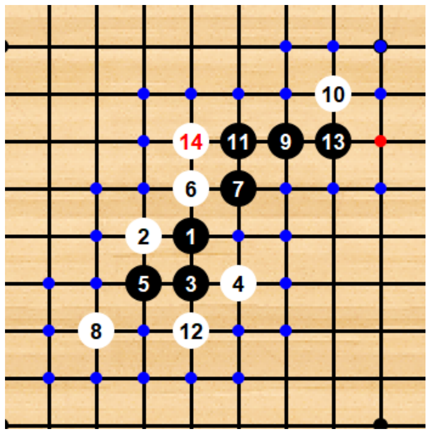


Figure 3.2: New Evaluation Range

3.2 Best Path

By narrowing the search range and sorting potential candidates, and then combining the technology of pruning, it can greatly narrow down the search results for a single calling of the minimax algorithm. Each time the program calls the algorithm to the output final result, it has made a complete calculation and prediction of the current situation of the board. But we only need one node output every time, so we spend a lot of time in the process but only get a simple result. If we can back up the useful information in these processes, will it be accelerated for the next time we recall the algorithm? Inspired by this, we try to explore what data can be retained by us and serve the next calculation in each minimax process.

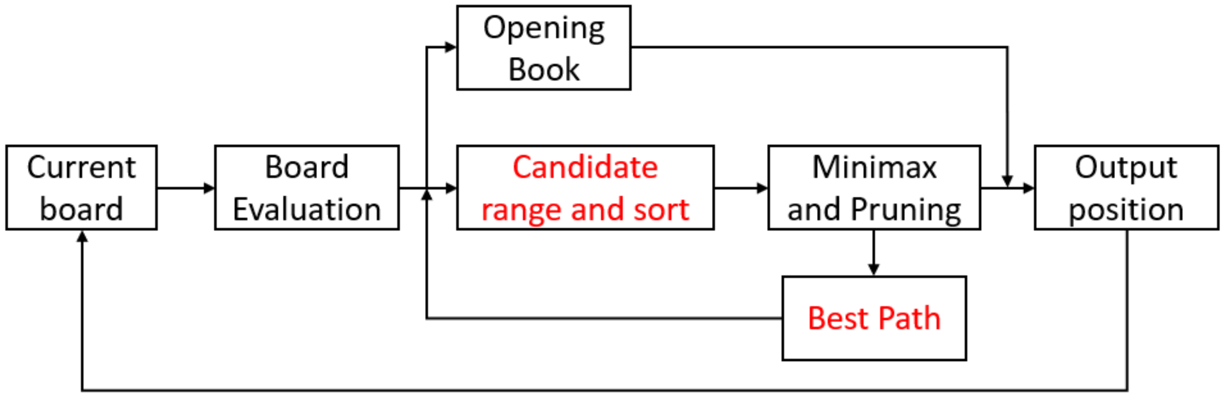


Figure 3.3: Flowchart of the back-end

As shown in the flowchart 3.3, it is the entire process of the back-end, in which we calculate the values of the board, and the recursive equations are constantly with new child and node generation and return nodes. If the best path in the current situation can be preserved by some means, that is the optimal path to the next of the node we are about to output. The direction of the chessboard with such a high probability will follow this route. Then next time when we recall the algorithm, it can refer to the best path to exclude potential candidates and make the whole process more efficient.

CHAPTER 4. EXPERIMENT AND RESULTS

Because this is program development and algorithm optimization research, so there are two things that need to be done when testing their specific performance. First of all, a reasonable reference object and a competitor are needed. We found the open-source code developed by user Canberkakcali on GitHub through the minimax algorithm. It can be said that this code is very simple and easy to understand, and also conforms to all the rules and algorithms of the most basic Minimax. We made 3 algorithms compete with each other to get the specific performance. Secondly, we will modify the parameters of the new heuristic algorithm to see how the variables affect the final result.

4.1 Algorithm Competition

Although Gomoku has been mathematically proven, the first moving black side is theoretically bound to win in the free-style rule. But the development of artificial intelligence has not yet supported this view. For curiosity, we tested three different algorithms. Let these three algorithms fight themselves in 1000 games. The results are shown in the Table 4.1.

Draw	**Greedy(B)**	**Greedy(W)**
112	482	406
Time Consuming	895 ns	1903 ns

Draw	**Minimax(B)**	**Minimax(W)**
324	457	219
Time Consuming	43 ns	83 ns

Draw	**Heurist(B)**	**Heurist(W)**
56	503	441
Time Consuming	29308 ns	25844 ns

Table 4.1: Black First Advantage

The *Minimax* algorithm in the table is exactly the code found on Github. We can see that in the *Greedy* algorithm comparison, the black and white winning rate is 50:50. But there was a relatively large change in the Minimax comparison, and black was significantly more advantageous. Finally, the conclusion given by the comparison of *New Heuristic* is also close to 50:50. So what we agree on here is that for artificial intelligence, the winning rate for both black and white under the free-style rule is unpredictable. Next, let the 3 algorithms battle with each other, and the result is as shown in Table 4.2.

Draw	**Greedy(B)**	**Minimax(W)**
82	367	551
Time Consuming	439 ns	23 ns

Draw	**Minimax(B)**	**Greedy(W)**
32	734	234
Time Consuming	26 ns	541 ns

Draw	**Greedy(B)**	**Heuristic(W)**
8	347	645
Time Consuming	560 ns	10367 ns

Draw	**Heuristic(B)**	**Greedy(W)**
18	806	176
Time Consuming	15073 ns	267 ns

Table 4.2: Comparison with *Greedy*

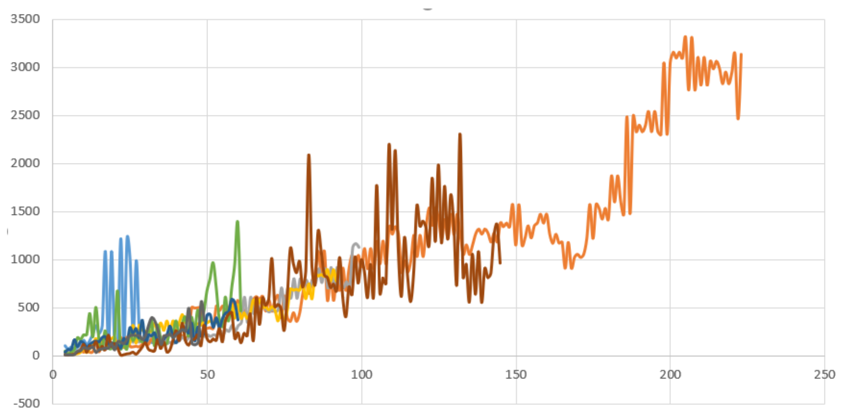
It is clear that the *Minimax* and *New Heuristic* algorithms are ahead of *Greedy*, whether they are black or white side. And when the *New Heuristic* algorithm takes the black side, it dominates *Greedy* and the score is close to 8:2. We can predict that the following *New Heuristic* will be slightly superior to *Minimax*. The outcome of their battle is shown in the Table 4.3.

Draw	**Heuristic(B)**	**Minimax(W)**
3	647	305
Time Consuming	15073 ns	89 ns

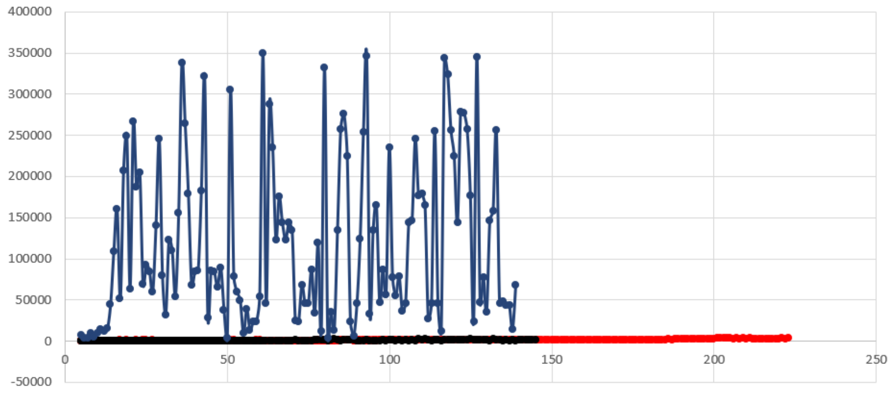
Draw	**Minimax(B)**	**Heuristic(W)**
46	428	526
Time Consuming	107 ns	20394 ns

Table 4.3: *New Heuristic* vs *Minimax*

As expected, the *New Heuristic* is stronger. In fact, for this result is no exception, comparing the two algorithms themselves are much the same. However, since the traditional *Minimax* does not improve in speed, the depth of the research is very limited. Taking into account time and actual use, the research depth in *Minimax* is 3 . And *New Heuristic* is 8. So more possibilities predictions will increase the result accuracy.



(a)



(b)

Figure 4.1: X-axis present the number of pieces on board, and Y-axis present the amount of calculation (a) Computation numbers for *Greedy* and *Minimax* algorithm (b) Computation numbers for *New Heuristic* Algorithm.

As shown in the Figure 4.1, Figure(a) shows the numbers of the calculation required. We will find that as the stones on the board increase, so does the amount of computation that needs to be calculated. However, a comparison with the calculation steps required by *New Heuristic* shows that the blue line on Figure(b) represents a trend chart of the number of *New Heuristic* calculations in the entire game, both are not in an order of magnitude at all. And the computational volume trend is unpredictable and there is no fixed trend. This is actually easier to explain at the code level. Because the range of candidates in the *greedy* and *minimax* algorithms is related to the number of known pieces on the board. As the game progresses, the more optional positions on the board, the more computations are required. In terms of *New Heuristic*, although we have the same number of candidates, because we rank the likelihood of the candidates, we may get the best answer early on, so the amount of calculation required is not fixed.

4.2 Parameter Impact

In testing, there are three parameters that can be modified: depth of the search, whether to use the optimized candidate, the candidate search range. In fact, the previous tests have included algorithmic comparisons for different search depths. What we just did was "Minimax" 3 depth compared to *New Heuristic* 8 depth. What if we all use the *New Heuristic* algorithm but just change the depth? The result is shown in Table 4.4.

Depth 4(B) vs Depth 8(W)	345:643:12
Depth 6(B) vs Depth 8(W)	500:477:23

Table 4.4: Depth Comparison

We made 2000 comparisons with 4, 6 and 8 depths, respectively. In the first set of data, we find that the deeper depth on searching, the greater the chance of winning. It was supposed to be the same for the second set of data, but we found that the winning margin was close to 50:50. So we conclude that the black first rule still affects the final result.

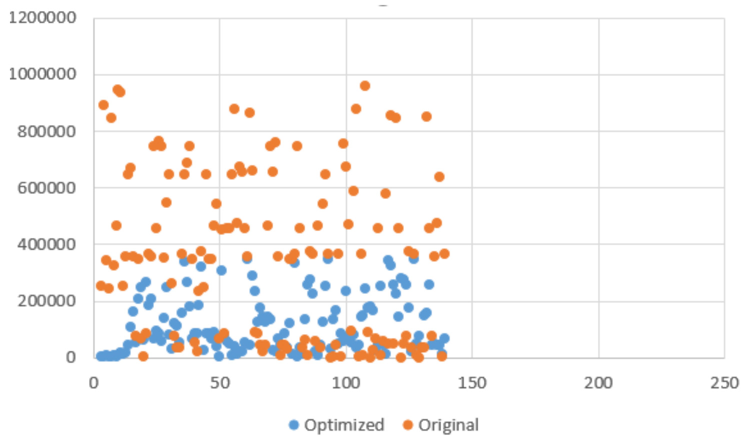


Figure 4.2: Computation number

Shown in the figure 4.4 is a comparison of the impact of using optimized candidates on the calculated quantity. It is still obvious that there are fewer calculations for candidates by optimization and sorting. This includes only a comparison of calculated quantities and has no effect on the accuracy and end result of each move.

one unite distance(B) vs two units distance(W)	517:447:36
two units distance(B) vs one unite distance(W)	508:431:61

Table 4.5: Candidate selection range

Last but not the least, we test the selection range of candidates from one unit distance from the current board stone to two units distance from known stones, the 1000 data shown in the Table 4.5 did not significantly affect the final result. Since the search range is larger and more time consumed, there is no particular need for current procedures.

4.3 Example

From the above test results, we can see that *New Heuristic* is still superior to other algorithms. Let's take a look at this with an example. In both of the figure 4.3 and 4.4, the previous 38 moves of both examples are identical. But *Minimax* and *New Heuristic* were at odds over the 39-move choice. *Minimax* believes (10,6) is a better choice because it forms a $LiveThree$ (move 33, move 9, and move 39). So far we can't say this choice is not good, but White has a response, the subsequent game situation is not certain. Instead, *New Heuristic* chose (10,7) as the best answer. Although the advantages of this step are not so obvious, a careful analysis of the 40th move must choose (11, 8), because if the black choice (11,8) will result in a $DeadFour$and $LIveThree$, at this time the white side has been the irretrievable situation. But even if White makes the right choice as (11,8), the final move of the *New Heuristic* made (11,7) will completely end the game, because this time produced a double $LiveThree$. So *New Heuristic* already took control of the game early in move 39. This example also reflects the gap between the two algorithms from the side.

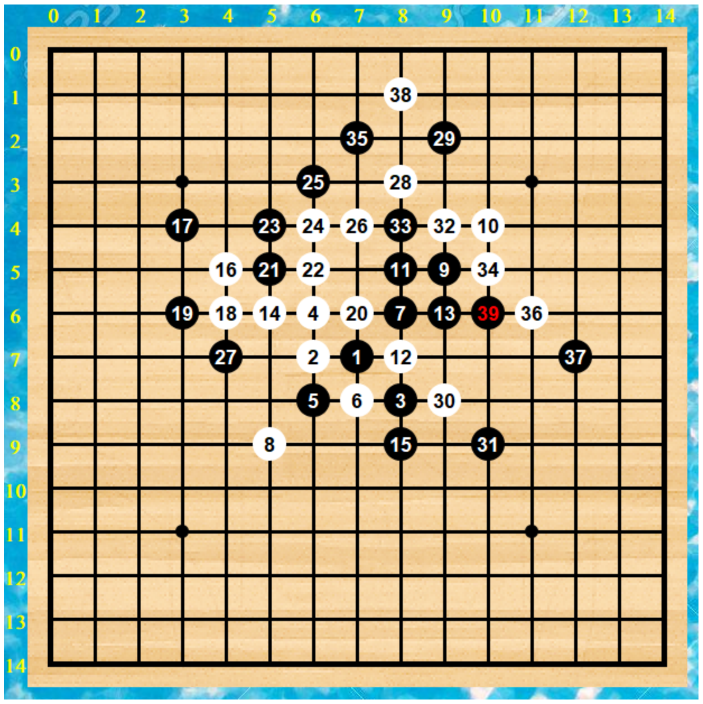


Figure 4.3: Minimax Example

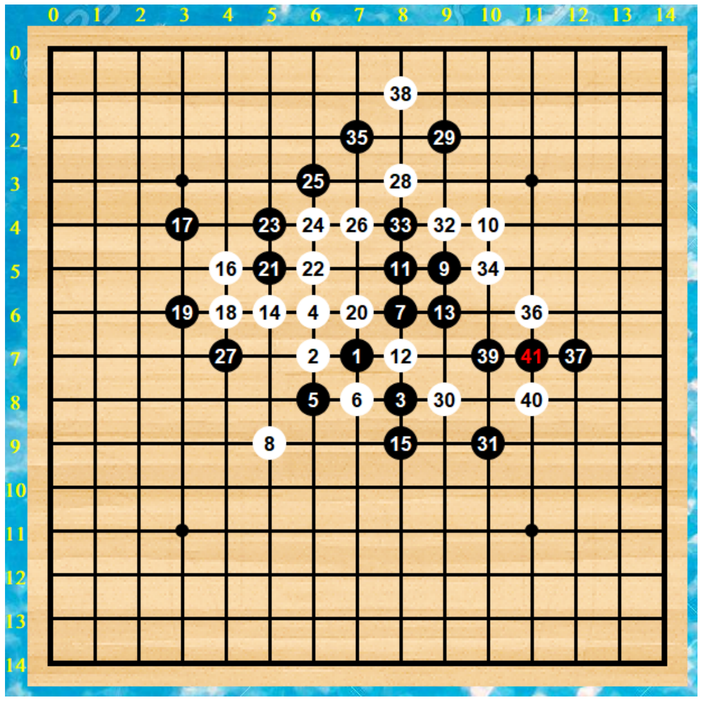


Figure 4.4: New Heuristic Example

CHAPTER 5. CONCLUSIONS AND FUTURE CHALLENGE

The Minimax algorithm is an ideal model. The algorithm itself is a brute force practice. Ideally, it would be infinitely close to the perfect answer. But it's clear that this exponential explosion of computing is unsustainable for both personal and supercomputers. From the experimental results, the optimization and sorting of candidates can greatly reduce the time it takes to run the program. In particular, this also directly affects whether it can get more deep searching. Therefore, simplifying the game tree as much as possible plays a crucial role in the output and final winning rate of each step. Especially with the help of the best path system, the calculations at each move are not becoming independent and irrelevant. In our experiments, we regret to find that the minimax algorithm does not prove that the black side has the advantage of winning for sure. We think that's what people have been pursuing all the time. We can't imagine that if we deepen the search indefinitely, we can come up with the answers we want. The maximum depth of our program can reach now is eight, which means that close to 2.5 billion possibilities can be evaluated at most. Although this figure may seem exaggerated, I think the program still needs to be improved, at least not at the level of Yi Xin, both in the results of the experiment and for my personal experience.

The best path system itself is just one of the easiest storage options. For example, in millions of recursive equations, we actually produce a lot of data overlap, such as the board evaluation will have to be recalculated. In future program improvements, we can consider whether to use the database, which can reduce a large number of duplicates brought about by the algorithm itself.

BIBLIOGRAPHY

Allis, V. (1994). *Searching for Solutions in Games and Artificial Intelligence*. Maastricht University, Netherlands.

Anton, R. (2018). Reevaluation of artificial intelligence engine alpha zero, a self-learning algorithm, reveals lack of proof of best engine, and an advancement of artificial intelligence via multiple roots. *Mathematical and Theoretical Physics*, 1(2):32–40.

Cooper, B. F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H.-A., Puz, N., Weaver, D., and Yerneni, R. (2008). Pnuts: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, 1(2):1277–1288.

Coulom, R. (2007). Efficient selectivity and backup operators in monte-carlo tree search. In van den Herik, H. J., Ciancarini, P., and Donkers, H. H. L. M. J., editors, *Computers and Games*, pages 72–83, Berlin, Heidelberg. Springer Berlin Heidelberg.

Donald E.Knuth, R. W. (1975). An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326.

G.C.Stockman (1979). A minimax algorithm better than alpha-beta. *Artificial Intelligence*, 12(2):179–196.

Guillaume Chaslot, S. r. and Istvan Szita, P. S. (2008). Monte-carlo tree search: A new framework for game artificial intelligence.

Ian Milling, J. F. (2009). *Articial Intelligence for Games*. Elsevier Inc, Miami.

Junru Wang and Lan Huang (2014). Evolving gomoku solver by genetic algorithm. In *2014 IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA)*, pages 1064–1067.

Kai, S. (2017). The strongest gomoku/renju engine in the world. http://www.aiexp.info/pages/yixin.html. [Online; accessed 28-November-2019].

Nasa, R. (2018). Alpha-beta pruning in mini-max algorithm –an optimized approach for a connect-4 game. *International Research Journal of Engineering and Technology*, 5(4):1637–1641.

Philipp Rohlfshagen, Stephen Tavener, D. P. S. S. and Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES*, 4(1).

Rongxiao, Z. (2016). Convolutional and recurrent neural network for gomoku. Master's thesis, Stanford University.

Silver David, H. A. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489.

Silver David, S. J. (2017). Mastering the game of go without human knowledge. *Nature*, 550:354–359.

Vardi, A. (1992). New minimax algorithm. *Journal of Optimization Theory and Applications*, 75(3):613–634.

Wikipedia (2019). Minimax — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Minimax&oldid=925373360. [Online; accessed 28-November-2019].

Zheng Peiming, L. H. (2016). Design of gomoku ai based on machine game. *Computer Knowledge and Technology*, 33(2).

Zhentao Tang, Zhao, D., Kun Shao, and Le Lv (2016). Adp with mcts algorithm for gomoku. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7.