Research on Hexchess Game System based Artificial Intelligence

LINGLING WANG, YIYANG WEI, FENG LI*
School of management science and Engineering, Anhui University of Finance and Economics
Bengbu 233000, CHINA

Abstract: With the rapid development of computer technology, artificial intelligence is emerging. Hex chess became popular because of its simple rules, but it also brought complex algorithms. Although the simple Monte Carlo tree search can be applied to the Hex game system, the search process is slow due to a large number of calculations. This paper proposes an improved Monte Carlo tree search algorithm based on the Upper Confidence Bound(UCB) formula to optimize the Hex game system and reduce the randomness of the Monte Carlo algorithm. To improve the efficiency of the search algorithm in the Hex game system, an effective system is adopted. Compared with the improved algorithm, not only the searching time of the Monte Carlo algorithm tree is improved, but also the performance of the algorithm is improved. At the same time, the system uses QT Creator to realize graphic interaction and complete the design of each module.

Keywords: Hex chess; Monte Carlo algorithm; UCB formula; strategy system.

Received: September 10, 2019. Revised: August 21, 2022. Accepted: September 11, 2022. Published: September 21, 2022.

1. Introduction

With the rapid development of computer technology and the improvement of hardware computing speed, people have a strong interest in computer chess [1], so the board game is gradually known to everyone. Since the 1950s, many famous scholars in the world began to get involved in the field of the game. John von Neumann, the father of the computer, proposed the most famous minimax theorem for game tree search [2]; The founder of artificial intelligence, John McCrthy, proposed the concept of "artificial intelligence"; A.l. Samuei was defeated by his program. Later, "Deep Blue", "Super Deep Blue", "Hand in Hand", "Go King" and the defeat of the world Go champion "AlphaGo" program represents the success of today's board game technology [3].

The core technology of computer games is to obtain the best result of the game within a limited time [4]. When the evaluation method is determined, there is little room for the optimization of valuation. Therefore, search algorithm has become the main research content of board games in the development of games [5]. The fundamental reason why the game can develop so quickly is that scholars are concerned about how to find a more efficient search algorithm. For the research of the Monte Carlo algorithm in recent years, the research has changed from the Monte Carlo simulation decision tree to the Monte Carlo tree search. In this paper, the UCB formula and Monte Carlo tree search are applied to the Hex game system to improve the performance and search speed of the algorithm [6]. At the same time, the algorithm adds some winning skills, such as death chess, hunting chess, and so on. The whole Hex game system mainly includes the combat platform and game design. The front-end display interface is combined with the back-end game progression. Among them, game design is the core of the whole game system. The specific design is shown in Figure 1.

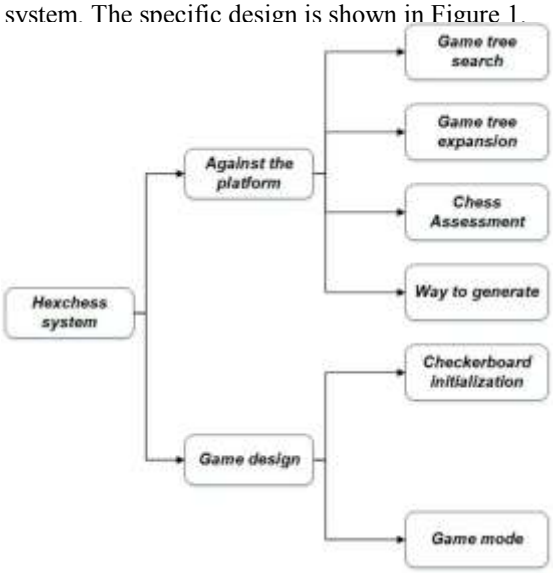

Figure 1. The design system of Game

The rest of this paper is organized as follows. We introduce the background in Section 2. Then, the UCT algorithm formed by the UCB formula combined with Monte Carlo algorithm random algorithm is applied to the hex chess game system in Section 3. In Section 4, we introduce the system of the proposed algorithm. In Section 5, the experiments are considered. The main conclusions of this paper are given in Section 6. Finally, section 7 is the future work.

2. Background

2.1 Hex

Hex, also known as Six Squares, first appeared in an article published by The Danish newspaper Politicen, then known as Ploygon [7]. It was independently invented by John Nasi again in 1948, and fans immediately called it Nasho. The Parker Brothers later released a version called Hex in 1952, after which the name stuck. Because Hex is popular in recent years, compared with chess, Go, gobang, and other ancient world chess, scholars have less research on it, so Hex has greater research potential and is more worthy of an in-depth study by scholars.

The Hex board has a diamond shape and is usually composed of 10×10, 11×11, and 14×14 hexagonal cells, while in domestic Hex games, the board is composed of 11×11 boards. The entire board has four boundaries, symmetric boundaries of the same color. In addition, there are two additional functions, restart and repent. Reopening is an essential feature of a match so that the match can be restarted at the end. The purpose of repentance is to prevent one side from making mistakes or breaking rules when playing chess in Figure 2.

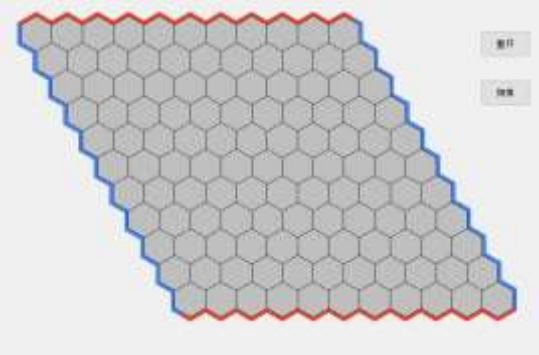

Figure 2. The board and pieces in Hex's chess game

2.2 The rules of the game

Hex is a relatively simple game. (1) Before the game starts, both sides choose the corresponding color of the chess pieces, corresponding to the boundary. After each side holds a colored chess piece, carries on the successive hand selection. (2) After the hand selection is complete, the game begins, the first hand (the player who plays chess first) and the second hand (the player who plays chess last) will take turns throwing the pieces on the board. Only one piece can be played at a time. Each time, take up space and put a piece of your color in the space. (3) The last time a game is played, the side whose color is connected to the line is the winner [8]. As shown in Figure 3, in the chessboard, red is connected first and red wins. The red piece goes first, so when the eleventh step piece falls, the piece that represents its color is connected to the boundary.

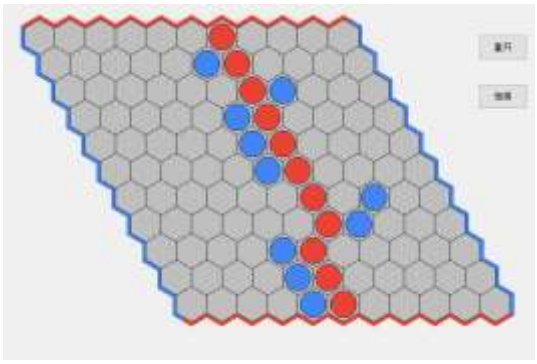

Figure 3. Red victory

3. Algorithm design

3.1 UCT algorithm

The computer game algorithm is mainly divided into two parts [9], the first part is game control, and the second part is chess game evaluation. Game control can be divided into three parts: game tree search, game tree expansion, and move generation. In the process of the game, the core of the chess layout lies the searching tree and the game evaluation. Traditional search algorithms Including the minimax algorithm in the most popular exhaustive search algorithm, alpha-beta pruning algorithm in the clipping search algorithm, hash table heuristic algorithm in the heuristic algorithm, etc. These algorithms all have a common feature. When searching to a certain depth, they evaluate the value of each branch to obtain the branch with the greatest value.

The traditional Monte Carlo algorithm tree search algorithm is a kind of tree search algorithm, called MCTS. It is itself a heuristic search algorithm for some decision-making processes and is more efficient in the process of large search space [10]. From a global perspective, the main goal of the Monte Carlo algorithm tree search is to select the best next step given a game state [11]. Alpha Go, as it is now known, uses Monte Carlo algorithm tree search. Every time Alpha Go makes a move, it runs thousands of these simulations using Monte Carlo algorithm algorithms to determine which move has the best chance of winning. But the actual implementation, there are still a lot of difficulties.

UCT algorithm is a popular search algorithm, it is based on the Monte Carlo algorithm. The Monte Carlo algorithm is combined with the UCB algorithm to obtain the optimal node by simulating a large number of nodes [12]. In the search process, the search depth for different branches can be different, so the search efficiency can be effectively improved [13]. The in-tree selection calculation of the UCT algorithm is mainly accomplished by the following formula:

$$\mathrm{Value} = \mathrm{V_i} + \mathrm{C} * \sqrt{2\mathrm{In(N)}/\mathrm{N_i}} \qquad (1)$$

where N is the number of visits of all nodes, and the value of C is a constant, which can be arbitrarily chosen, but the best value of C is generally obtained through the experimental process; $\mathrm{V_i}$

represents the average value of all simulation results with the current node as the root node, reflecting the expectation of the return value that can be provided by the node observed according to the current simulation results; N_i is the number of times the current node was visited, which is the number of times it was selected in the search tree.

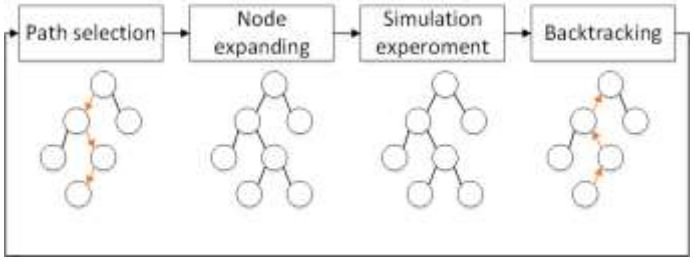

Figure 4. Search tree search process

The realization process of the UCT algorithm is mainly completed by four steps: node selection, expansion, simulation, and backpropagation. As follows:

(1) Path selection. Starting from the root node, each child node that can be selected is recursively selected and the Value of the node is calculated. The node with the largest value is selected as the beginning of the next recursive selection, until the leaf node. The selection process is shown in the orange arrow in Figure 4.

(2) Node expansion. The optimal node selected in the previous step is added to the search tree as a new leaf node with appropriate initial values for its V and N.

(3) simulation. An appropriate simulation strategy should be able to accommodate all of the moves. So according to the new game tree, the Monte Carlo algorithm is used to choose the next action uniformly and randomly. Then all the leaf nodes are evaluated, and the evaluation strategy can be simply set as 1 if your side wins, 0 if your opponent wins. Through multiple simulations, a better Value can be obtained.

(4) backtracking. As can be seen from the backtracking section in Figure 4, when leaf nodes obtain new values and access times through simulation, the UCT algorithm updates the values of all V and access T on the search path through result return. The calculation formula is:

$$N = \sum_i N_i \qquad (2)$$

$$V = \sum_i V_i N_i \Big/ T \qquad (3)$$

That is, the number of visits N of the parent node is the sum of the number of visits T of all leaf nodes, V is the weighted average of all leaf nodes V under the current node, and the weight is the ratio of the number of visits of child nodes [14]. The results are then sent back, starting at the leaf node and updating the network along the search path to the root node. See Figure 4.

When the recursion is complete, it means that the next piece is selected and placed in the cell. At this point, the UCT search process is in a new state, all the values in the previous recursive process are newly initialized, and the UCT search process restarts from the first expansion step.

When the recursion is complete, it means that the next piece is selected and placed in the cell. At this point, the UCT search process is in a new state, all the values in the previous recursive process are newly initialized, and the UCT search process restarts from the first expansion step.

3.2 Special type system

Compared with the traditional minimax algorithm and Monte Carlo algorithm, the UCT algorithm is improved in the search algorithm [15]. For some special moves that often appear in chess games, we can directly respond to the situation. The Special type system is mainly aimed at some Special chess types. When these chess types appear, it can give priority to them directly and give corresponding Special moves according to their particularity, which can greatly reduce the algorithm search time. In general, the Special type system mainly considers the death type, hunting type, and must-win type [16].

（1）Death mode. At some point in the game, there will be a situation where the current local move will not work for any player. This means that neither side can influence the outcome of the game. That is to say, no matter which side places the pieces in these positions, it does not affect the outcome of the game. See Figure 5.

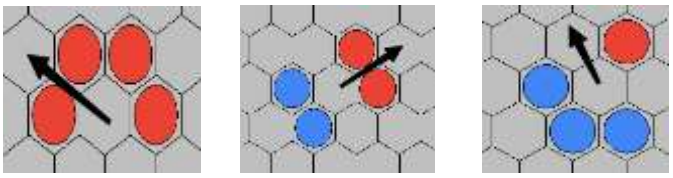

Figure 5. Death pattern

（2）Hunting chess. When the following situation occurs in a chess game, take red as an example. When the blue moves in one of the two marked positions on the chart, the red moves immediately to the other position to keep the surrounding pieces connected. The red side doesn't have to play in either of these positions, it can choose any position, and the blue side has to play in one of these positions before it connects. This can be interpreted as one side being closely watched by the other when a hunting position is formed in Figure 6.

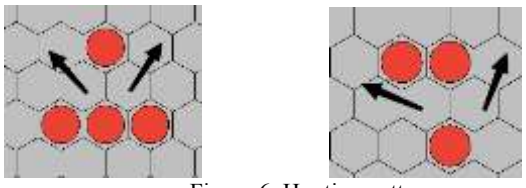

Figure 6. Hunting pattern

（3）Win pattern. In the course of the game, both sides of the game must look for opportunities to deploy the board shown below, as shown below. as shown in the picture below. When one side deploys five similar pieces on a path, the other side is bound to lose. It relates to what we were talking about in the hunting game, where wherever the blue side is playing, the red side is going to play on the other side to ensure connectivity.

However, when the successive pieces are deployed, Red Convenience is sure to win. in Figure 7.

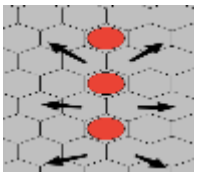

Figure 7. Win pattern

The above three types of chess can be fundamentally avoided. When one player finishes the game, the other should watch closely to prevent it from forming the above three situations. Conversely, it gives us a way to improve our winning percentage. For example, when an arresting move occurs, when we fill in any marker position, if the opponent does not fill in another position, the piece is connected. At this point, we can break its hunting pattern and change it from passive to active. Not all Hex programs take into account particular shapes and offer solutions. In addition, the first step is usually placed in the middle of the board, and experiments can prove that the side that places the first piece in the middle is easy to win.

4. System Implementation

4.1 The realization of everyone versus everyone

The game player's control of the chess pieces requires the occurrence of mouse click events. Clicking on a free call with the mouse to move a piece requires the piece and the board to communicate information directly and interact. The signals and slots mechanism in QT Creator solves this problem effectively. The connect（） function is required to transport signals and slots. Semaphores and slots are defined as signal（） and slot（）, respectively. Each time a key is defined, a connect（） is executed so that the clock signal triggers the defined slot function to pass the parameters.

Signal and slot declaration:

```
void on_Click(int row,int col);
void clickEmptyPoint(int row,int col);
```

Send a signal:

```
QObject:connect(ui.canvas,SIGNAL(clickEmptyPoint(int,int)),
this,SLOT(on_Click(int,int)));
```

Receive a signal:

```
void app:on_Click(int row,int col)
{PlayerSetPiece(row,col);}
```

4.2 Realization of man-machine war

（1） Defined parameters: In terms of the preservation of checkerboard data, we define the checkerboard as a two-dimensional array of chess [11]. Everything in the array is 0; Because 0 means the current board is empty. Similarly, when red plays, red is represented by an array of zeros; Blue is represented as a 1 when blue plays. Specifically, the piece is defined as a class, with Nexpnanded-access times of the node; Children indicates the child node of the node. Index Indicates basic parameters such as the index value of the current node.

（2）Judge if it can play chess now: Judge whether the current is the most advantageous: When the player drops a piece, the whole board is searched as follows: First, the whole board is searched for blank pieces to find the squares that can be played. Second, after finding the position, through a random function to obtain a position, and then check whether the current position can play chess. If yes, select, expand, simulate, and traceback from the current node. If not, continue to call a random function to find another location.

（3）Judge if the current is the most advantageous:Computer control of chess pieces mainly depends on the dynamic establishment of game tree, with the help of UCT search algorithm optimization completed. The following functions are mainly used: selection function （）, expansion function （）, simulation（） and back（）.

```
void selection(Node *& current, disjointset::IDisjointSet *uf);
void expand (Node *& current, disjointset::IDisjointSet *uf);
color::Color simulation(Node *& current, disjointset::IDisjointSet *uf);
void back(Node *& current, disjointset::IDisjointSet *uf, const color:: color winner);
```

The entire search process is mainly implemented through these four functions. If the current node has the highest chance of evaluation, it increments Win and Total and returns row arguments. Notice that at the end of one search, the parameter is reinitialized to zero on the next search.

5. Experiments

5.1 Experimental scheme

Nowadays, there are more and more chess algorithms. The application of the UCT algorithm in the Hex chess game is verified by comparing it with the Hex chess game system which adopts the traditional Monte Carlo algorithm as the search algorithm. To obtain clear experimental results, the experiment times were set as 1, 10, 50, and 100 times respectively, and then the experiment was carried out in sequence. In addition, the experiment compares the winning rate of the first and last hand of the UCT algorithm in chess. In this experiment, the coefficient C in the UCT formula is 2.

5.2 Experimental results

As shown in Table 1, when the system uses the UCT algorithm for firsthand, when the number of experiments is 1, 10, and 100, the success rate is 100%. When the number of trials was 100, the success rate was 98 percent. The results show that the UCT algorithm is superior to the traditional Monte Carlo algorithm when the system uses the UCT algorithm and plays chess first.

Table 1. Experimental results of UCT algorithm(first)

Number	UCT algorithm	Monte Carlo algorithm	Odds/%
1	1	0	100
10	10	0	100
50	49	1	98
100	100	0	100

As shown in Table 2, when the system uses the UCT algorithm in the last chess game, the success rate is 100% when the number of experiments is 1, 10, and 50. When the number of trials was 100, the success rate was 99 percent. The results show that the winning rate is significantly higher with the UCT algorithm and the last set than with the Monte Carlo algorithm. Combined with the experimental results in Table 1, the UCT algorithm is significantly superior to the traditional Monte Carlo algorithm in both primary and secondary chess.

Table 2. Experimental results of UCT algorithm(last)

Number	UCT algorithm	Monte Carlo algorithm	Odds/%
1	1	0	100
10	10	0	100
50	50	0	100
100	99	1	99

As shown in Table 3, both sides of the game use the UCT algorithm. When the number of tests is 1,10,50 and 100 respectively, the first-hand chess win rate fluctuates at 60%. Therefore, for the UCT algorithm game system, first-hand chess is easier to win.

Table 3. Win percentage comparison between first hand and last hand

Number	First	Last	First odds/%
1	1	0	100
10	6	4	60
50	29	21	58
100	61	39	61

Experimental data show that the UCT algorithm has a higher winning rate than the traditional Monte Carlo algorithm when the parameter of the UCT algorithm is fixed c and the first hand and the last hand are arbitrary. Compared with the traditional Monte Carlo algorithm, the UCT algorithm is proved to be relatively accurate and fast, and better than the original Monte Carlo algorithm.

6. Conclusion

This paper mainly introduces the research and design of the Hex game system. Based on the traditional Monte Carlo search algorithm, a new UCT algorithm is proposed, which can control the search depth under the controllable search time or search times, and improve the search speed of the game tree fundamentally. In addition, a strategy system is added to the UCT algorithm to further improve the search time. At the same time, the UCT algorithm can promote the development of the Hex search algorithm and promote the further development of computer games. It even is applied to other types of chess, such as checkers, Ehrenstein, etc.

In future work, we will continue to optimize the UCT search algorithm to further improve its search speed and time. In the future, the improvement of the UCT algorithm mainly focuses on the improvement of the node expansion method. For example, the simulation times of fixed nodes are compared with a certain factor to judge whether the nodes are expanded or not. In addition, UCT algorithms can be compared not only to traditional Monte Carlo trees but also to players and other search algorithms.

Conflict of Interest

The authors declare that they have no competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Author Contributions

Lingling Wang and Yiyang Wei designed the experiments, implemented the models, performed the experiments, analyzed the experiment results, and wrote the paper; Feng Li guided, revised, and fine-tuned the paper.

Acknowledgment

This work was supported in part by the Natural Science Foundation of the Higher Education Institutions of Anhui Province under Grant No. KJ2020A0011, Innovation Support Program for Returned Overseas Students in Anhui Province under Grant No. 2021LCX032. the Science Research Project of Anhui University of Finance and Economics under grant No. ACKYC20085, the Research Project of Anhui Provincial Education Department under Grant No. 2020jyxm0037, No. 2020zyrc020, and No. cxjhjyzda 1803.

References

[1] Cazenave, Tristan, Mark HM Winands, and Abdallah Saffidine, eds. Computer games. Springer International Publishing, 2018.

[2] S. Ben-David and E. Blais, A New Minimax Theorem for Randomized Algorithms (Extended Abstract), 2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS), 2020, pp. 403-411.

[3] Zhang, D., Lindholm, G., & Ratnaweera, H. (2018). Use long short-term memory to enhance Internet of Things for combined sewer overflow monitoring. Journal of Hydrology, 556, 409-418.

[4] Zhang, R., Jiang, X., & Li, R. (2017). Decomposition based multiobjective spectrum allocation algorithm for cognitive vehicular networks. Paper presented at the 17th IEEE

International Conference on Communication Technology, ICCT 2017, October 27, 2017.
[5] G. Yıldızdan and Ö. K. Baykan, A Novel Artificial Jellyfish Search Algorithm Improved with Detailed Local Search Strategy, 2021 6th International Conference on Computer Science and Engineering (UBMK), 2021, pp. 180-185.
[6] Enzenberger M, Müller M, Arneson B, Segal R. Fuego—an open-source framework for board games and Go engine based on Monte Carlo tree search. IEEE Transactions on Computational Intelligence and AI in Games. 2010 Oct 14;2(4):259-70.
[7] Z. Li, H. Liu, Y. Wang, J. Zuo and Z. Liu, Application of Monte Carlo Tree Optimization Algorithm on Hex Chess, 2020 Chinese Control And Decision Conference (CCDC), 2020, pp.3538-3542.
[8] M. Lu and X. Li, Deep reinforcement learning policy in Hex game system, 2018 Chinese Control And Decision Conference (CCDC), 2018, pp. 6623-6626.
[9] Q. Du, J. Zhao, L. Shi and L. Wang, Research on the two-dimensional face image feature extraction method, 2012 3rd International Conference on System Science, Engineering Design and Manufacturing Informatization, 2012, pp. 251-254.
[10] Alhejali, A. M., & Lucas, S. M. (2013). Using genetic programming to evolve heuristics for a Monte Carlo algorithm Tree Search Ms Pac-Man agent. Paper presented at the 2013 IEEE Conference on Computational Intelligence in Games, August 11, 2013, Niagara Falls, Canada.
[11] Kato, H., & Takeuchi, I. (2010). Parallel Monte-Carlo tree search with simulation servers. Paper presented at the Proceedings - International Conference on Technologies and Applications of Artificial Intelligence, TAAI 2010.
[12] Hou, F., He, H., Xiao, N., Liu, F., & Zhong, G. (2010). Incremental hash tree for disk authentication. Paper presented at the 15th IEEE Symposium on Computers and Communications, ISCC 2010, June 22, 2010 - June 25, 2010, Riccione, Italy.
[13] De Temino, L. A. M. R., Berardinelli, G., Frattasi, S., & Mogensen, P. (2008). Channel-aware scheduling algorithms for SC-FDMA in LTE uplink. Paper presented at the 2008 IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC 2008, September 15, 2008 - September 18, 2008, Poznan, Poland.
[14] Y. Fu, S. Li and Y. Qi, Amazon game system design and realization based on staged UCT algorithm, 2018 Chinese Control And Decision Conference (CCDC), 2018, pp.5929-5932.
[15] Looney, C. G. (2006). Intelligent battle gaming pragmatics with belief network trees. Paper presented at the 2006 IEEE Symposium on Computational Intelligence and Games, CIG'06, May 22, 2006 - May 24, 2006, Lake Tahoe, NV, United states.
[16] Cai, H., Kulkarni, S. R., & Verdu, S. (2006). An algorithm for universal lossless compression with side information. IEEE Transactions on Information Theory, 52(9), 4008-4016.