Hexagonal War Chess Bot Using Memorized Search and Greedy strategy

Songqiang Xu[1], Hongbin Ma[*2],Weipu Zhang[3]

[1]School of Information and Electronics, Beijing Institute of Technology, Beijing, 10008
E-mail: 1120190856@bit.edu.cn
[*2] School of Automation, Beijing Institute of Technology, Beijing, 10008
E-mail: mathmhb@bit.edu.cn
[3]School of Xuteli, Beijing Institute of Technology, Beijing, 10008
E-mail: 1120191516@bit.edu.cn

Abstract. In this paper, we present a robot program for a special hexagonal war chess game that can expand, explore and attack automatically. First, we introduce the rules and process of the hexagonal war chess game. Then, we theoretically analyze the requirements and characteristics of the robot algorithm on the basis of the rules of the game, and then implement the robot by memorizing search and greedy strategy. Through the experiment of simulating the battle between each other, it is verified that our robots have strong fighting power and have high rapidity and accuracy in attacking. To sum up, the memorized search and greedy strategy are practical and effective to improve the aggression of the robot in this game.

Keywords: Memorized search, Greedy strategy, Breadth-first search, Robots

1. INTRODUCTION

The design of intelligent robots for strategy games is a complex problem [1]. Some games have a fixed state so it is appropriate to use a computer to pre-process a large number of future board states. Some games are more random and have more complex states, making it difficult to make decisions by enumerating states. One solution is to use informed search. There have been some previous researches [2–10] on the informed search already. The value of this strategy is that it reduces complexity by directing the search in the most promising direction. In this paper, memory search and greedy strategy are applied to improve the attacking efficiency of the robot in the game.

Greedy strategy is a technique for solving some optimal problems quickly. It has spawned a variety of algorithms [11-13]. Its characteristic is to proceed step by step, based on the current state, make the optimal choice according to a certain optimization condition, without considering all possible overall situation. This saves a lot of time that would have been spent enumerating all the possibilities. The greedy strategy uses the top-down method to make successive greedy choices, and each time the choice is made, the problem is reduced to a smaller subproblem. Through greedy selection at each step, an optimal solution of the problem can be obtained. Although each step is intended to ensure that can obtain the local optimal solution, but sometimes the resulting global solutions are not necessarily optimal. In the game in question, greedy strategy can achieve better results with less computation.

Several major contributions of this paper can be highlighted as follows:

1. This paper briefly introduces the rules of hexagonal war chess and makes a theoretical analysis of winning ideas. The basic abilities required by a winning robot and the basic strategies to be followed are summarized.

2. According to the above requirements, the corresponding robot will be programmed to achieve. We optimize for a variety of special situations, so that the robot can efficiently expand, explore and attack. Through the use of memorized search and greedy strategy, the robot has a high efficiency when attacking.

3. Demonstrating the validity of this robot algorithm in aggressive behavior by simulating battle between robots with different logics.

The rest of the paper is organized as follows. Section 2 introduces the basic rules of hexagonal general chess game. In Section 3, we make a theoretical analysis of the winning ideas based on the rules of the game, and summarize the abilities and strategies that the robot needs to have. In Section 4, according to the theoretical analysis above, we implement the robot program, and use memorized search and greedy strategy to improve the aggression of the robot. The effectiveness of the results is verified by comparing with the simulated battle between ordinary robots. Section 5 gives a brief conclusion and summary of the results.

2. RULES OF THE HEXAGONAL WAR CHESS

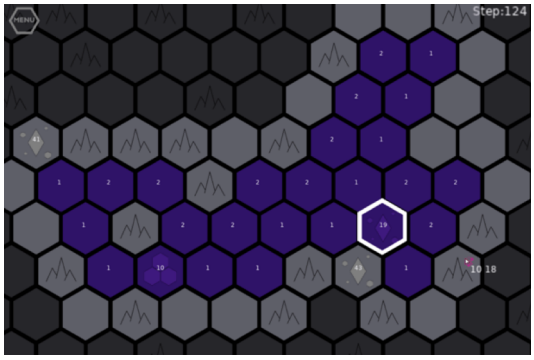

Fig. 1: The map

The hexagonal war chess is a real time strategy game with war chess elements. In the game, you need to control your army to occupy land, occupy cities, build up forces and defeat all enemies, and when there is only one player left, that player wins.

2.1. Map introduction

The map of the game is a random honeycomb map composed of countless hexagonal grids. Each hexagon represents a region (Fig.1).
These hexagonal regions can be divided into four different types, which are (Fig.2-5):
1. BLANK:
The most basic type. It is initially owned by no player and has a force of 0. When a player moves to an area of this type, the area automatically belongs to that player and increases by 1 force every 25 seconds.
2. HILL:
Type of area that cannot be passed or occupied by any player.
3. FORT:
It is initially not owned by any player, has a certain number of troops, but does not increase. When occupied by the player, it belongs to the player and increases the number of troops by 1 per second. The map is now shown as a HILL type when not detected by the player.
4. KING:
The player's starting point. Each player has only one KING, which increases by 1 per second. When occupied by another player, the area type is changed to FORT, the occupied player loses, and all areas and forces occupied belong to the attacker. When only one player's KING is left on the map, that player wins the game.

2.2. Game features

2.2.1. Forces growth

The number of troops in a player's territory on the map increases over time, with different types of territory growing at different rates. - All KING's and occupied FORT's forces increase by 1 per second; All BLANK forces occupied by the player increase by 1 every 25 seconds.

2.2.2. Movement and attack

When a player moves troops from an area (the source area) to an adjacent area that is not his or her own (the target area), as long as the target area is not of type HILL, it


Fig. 2: BLANK area


Fig. 3: HILL area


Fig. 4: FORT area


Fig. 5: KING area

automatically attacks. If the force in the source area is greater than the force in the target area, the target area belongs to the player in the source area. Otherwise, the force in the target area is not changed and the force in the target area is reduced to (target area force - Source area force +1). The source area retains 1 force regardless of whether the force in the source area is changed or not.

When a player moves troops from one area (the source area) to an adjacent area that is his or her own (the target area), the source area force becomes 1 and the target area force becomes (the target area force + the source area force -1).

Players can only do one action per second on their territory.

2.2.3. Visible area

In the game map, players get a view of the area they occupy and six adjacent areas, and can learn the types, affiliations and forces of these areas; Other areas are considered to have no field of view and become darker in color, the player can't get their ownership and troops, KING areas are shown as BLANK, FORT areas are shown as HILL, and only when the field of view is obtained will they appear as FORT and KING on the player's map.

2.2.4. Victory and defeat

When a player's KING is occupied by another player, the occupied player is forfeited, and the KING is converted to a FORT and belongs to the occupier, along with the BLANK and FORT areas previously owned. When there is only one player left on the map, that player wins the game.

3. ANALYSIS OF GAME STRATEGY AND ROBOT DESIGN REQUIREMENTS

3.1. Basic principles of optimization strategy

From the rules of the game, we can easily conclude the basic principles that the optimal game strategy must follow. In order to win the game as much as possible, we must follow the following principles. These principles are obvious and therefore their derivation will not be discussed in this paper.
Principle 1 Occupy as much of the FORT as you can, in order to get rapid replenishment.
Principle 2 Occupy as much of the BLANK as you can, in order to get slow replenishment.
Principle 3 Plan the path of troop movement so that the number of steps required to achieve the same result is minimal.

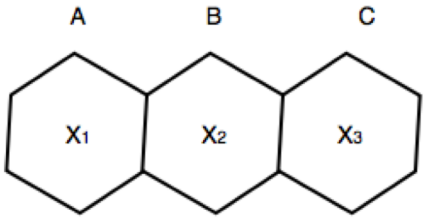

Fig. 6: Three adjacent hexagonal areas.

3.2. The logic of the ideal robot

Based on the above basic principles, we can design a general logic of action for the ideal robot. Note that the action logic at this point is still non-specific and will be added in detail later.

Step 1 Expand the army from the KING area to the surrounding areas, increasing the size of territory.

Step 2 Occupy as much of the FORT as it can, and get enough troops for itself.

Step 3 Mobilize troops to attack enemy's territory, looking for the enemy's KING area.

Step 4 Occupy enemy's KING area.

3.3. Core algorithm for moving troops

The essence of this game is the movement of troops, whether it is expansion, exploration, defense or attack, all involve the movement of troops. Therefore, how to quickly move enough troops to the designated position is one of the important problems that the robot needs to solve. To discuss this problem, we use three adjacent hexagonal regions as examples (Fig. 6).

Consider three contiguous regions, A, B, and C, from left to right. They have forces X_1, X_2, and X_3. Now we want to move both A and B's forces to C.

At this point, we have two ways to move our forces. One is to move A's force to C first, and the other is to move B's force to C first.

Calculate the number of steps required for each method:

A move first:
A→B B→C
2 steps
B move first:
B→C
A→B B→C
3 steps

Obviously, it takes fewer steps to move A first, because the path to move A includes the path to move B. So, in a real game, we need to take advantage of this inclusion relationship between paths. Move the forces farther away first.

To achieve this, we use the breadth-first search method (We will refer to it as BFS for short). BFS was invented in the late 1950s by E. F. Moore [14], and discovered independently by C. Y. Lee as a wire routing algorithm [15 16]. By taking the target location of carrying troops as the root node and carrying out the BFS, the adjacent areas can be searched in the order from near to far. When we refer to BFS later, we actually build a binary tree based on the results of the search to record the path from the child node to the parent node.

3.4. The application of memorized search and greedy strategy

In order to effectively win the game, robots should have efficient and accurate attack modules. Make sure to find the enemy's KING area as soon as possible and avoid pointless exploration in enemy territory. Therefore, we used memorized search and greedy strategy in the attacking part of the robot.

First, let's introduce a new parameter, **regional potential**, which is defined as the number of inaccessible views in the six adjacent regions of a certain region. The higher this number, the larger the field of view that can be added after occupying the area, and the easier it is to find the enemy KING area. Let's take the following state as an example (Fig.7).

The blue area in the picture is our territory, and the green area is the enemy's territory. According to the rules of the game, HILL is an obstacle that does not belong to any player, and the gray area marked with a question mark is an unknown area that we cannot see at the moment. By definition we can figure out that A and B 's **regional potential** is 1, C and D 's **regional potential** is 3. Therefore, we believe that C and D are more suitable targets for attack. In theory, occupying C and D makes more of the area visible, thus increasing the chance of finding the enemy's KING. From the human experience of playing these kinds of games, we also tend to explore in the middle. Therefore, this strategy is consistent with the actual needs.

The specific content of our greedy strategy is that before each operation, all enemy territory in the field of vision is counted, the **regional potential** of each territory is calculated, and the territory with the highest **regional potential** is selected as the target of attack. With this algorithm, our robot can quickly find the enemy KING area, and occupy less non-king areas, reducing the loss

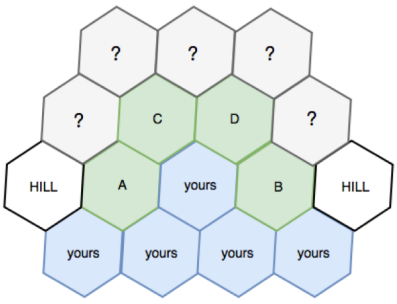

Fig. 7: An example of an attack scene.

of troops.

The specific content of our memorized search is that after each occupation of a territory, the coordinates of that territory are marked to indicate that it has been occupied. When choosing targets, we will compare the **regional potential** of each enemy's territory and determine whether they have already been occupied. If the enemy territory has been occupied by us before, lower the priority of the area and attack the unoccupied area first. The purpose of using memorized search is that the enemy's territory we once occupied may be taken away by the other side. But our ultimate goal is to find the KING of the opposing team, so next time we attack the unoccpied area first.

4. PROGRAM IMPLEMENTATION

4.1. Implementation of main functions

4.1.1. Expansion of territory

The purpose of territory expansion is to increase the area of BLANK areas and the number of FORT areas. Among them, it is more important to increase the number of FORT areas. Therefore, we rejudge whether there are any FORT that can be occupied in the field of vision in each turn. If there are, we will try to occupy the FORT; otherwise, we will try to occupy the BLANK.

In the operation of FORT occupation, we need to move a sufficient number of troops from nearby territories to the FORT area. In order to improve the speed of occupation, we adopt the breadth-first search order when mobilizing troops, and make traversal with target FORT as the root node, and accumulate the available forces in our territory along the traversal process. The traversal stops when the accumulated forces exceed the FORT forces. Finally, according to the traversal order, the forces in each region are moved to its parent node to realize the occupation of FORT.

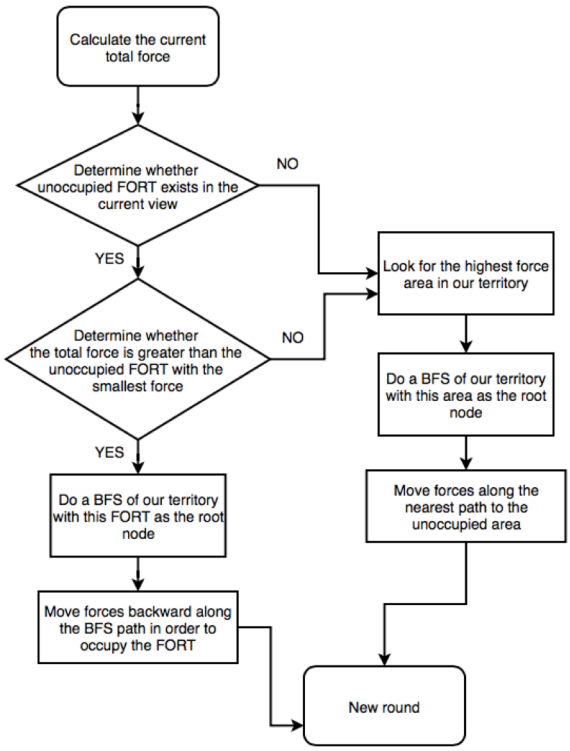

Fig. 8: Flowchart of expansion behavior.

When executing the operation to occupy BLANK, the expansion method is different than when occupying the FORT area. We choose the area with the highest military force in our territory, take this area as the root node to conduct a width-first search around, find the BLANK area nearest to this area, and record the path to this area. Moving troops along this path makes the overall force spread around more evenly, making the territory area rapidly larger.

4.1.2. Gather the army

In order to improve the effectiveness of attack, we divide the whole process of attack into two parts: **gather** and **attack**. **Gather** means to concentrate a large number of troops in one area of our territory, **attack** means to move the forces in that area towards the enemy territory. Only the implementation of **gather** is discussed in this section. The steps for **gather** are as follows:

Step 1 Choose one of our territories as a place to gather forces (*troop location*).

Step 2 Calculate our total forces, and set 80% of our forces as a target for gathering army (*target forces*).

Step 3 Take the *troop location* as the root node to conduct a BFS of our territory, and accumulate the territory forces traversed. The traversal stops when the accumulated force exceeds the *target forces*.

Step 4 Start with the node at the end of the queue. Move its forces toward its parent node.

Step 5 Determine whether the force at the *troop location* has reached the *target forces*. If so, enter the **attack** mode. If not, go back to step 1

4.1.3. Attack mode

This section describes the **attack** mode after a successful **gather** mode. The steps of the **attack** are as follows:

Step 1 Choose an enemy territory in view as the target of attack (*attack target*).

Step 2 With the *attack target* as the root node, make a BFS of our territory, stopping when it has traversed the *troop location*.

Step 3 Move the forces from *troop location* to its parent node and use the coordinates of the parent node as the new *troop location*.

Step 4 Determine if the forces of *troop location* are greater than 1. If so, proceed to the next round. If not, it is considered that the forces are exhausted and the next turn will be in **gather** mode.

5. SIMULATION RESULTS OF THE ROBOT

To test the effectiveness of our robot, we put our robot in simulated battles with robots that didn't use this algorithm. We arranged three groups of tests, pitting

Tab. 1: Simulation results.
(Green team means our bot in the third group)

	Normal bot v. normal bot	Our bot v. our bot	Our bot v. normal bot
Green team win rate	0.68	0.41	0.93
Blue team win rate	0.32	0.59	0.07
Average steps	1376.23	634.17	968.72

normal robot against normal robot, our robot against our robot, and our robot against normal robot. Play 100 battles per team. 100 randomly generated maps will be used in the battle. Each battle record records the number of moves used in the game and battles won or lost. The statistical results are shown in Tab.1

Fig. 10: Beginning of the battle.
(Normal bot v. normal bot)

Fig. 11: Middle of the battle.
(Normal bot v. normal bot)

Fig. 12: Beginning of the battle.
(Our bot v. our bot)

Fig. 13: Middle of the battle.
(Our bot v. our bot)

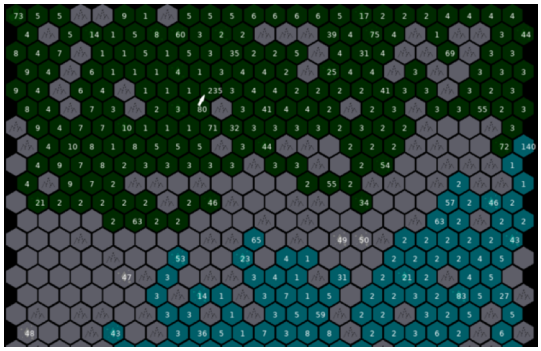

Fig. 14: Beginning of the battle.
(Our bot v. normal bot)

Fig. 15: Middle of the battle.
(Our bot v. normal bot)

To further validate our design, we observed three games with the same map from three sets of tests.

Test 1 Normal robot against normal robot
Compare the difference between the two images of the blue side. Its attack pattern is to cover large areas of enemy's territory. It's not targeted. The total number of steps in this game is 1,416, which is relatively long. The results show that the normal robot attack speed is not sharp enough, often get into a stalemate, it takes a long time to determine the winner.

Test 2 Our robot against our robot
Compare the difference between the two images of the blue side. Its attack pattern is a serpentine exploration of enemy's territory. But there is no enemy's territory out of sight. The total number of steps in the game is 807, and the duration is relatively short. The results show that our robots are highly aggressive, and in a two-game war, one will soon be overrun by the other.

Test 3 Our robot against normal robot
Compare the differences between the two images and attacking patterns of the blue and green teams. You can see that the blue team (normal robot) is still covering enemy's territory, and the green team (our robot) is snake exploring enemy's territory. In the end our robot won. The total number of steps in this game is 1102. The results show that our robot can quickly and accurately attack enemy's territory.

6. CONCLUSIONS AND FUTURE WORK

In order to design the robot program that can have efficient attack ability in hexagonal war chess. We first made a theoretical analysis of the rules of the game. Summarize the basic principles that must be followed to win the game according to the rules of the game. Then we designed the general behavior structure of the robot according to the basic principles. In the implementation of the program, we encountered some difficult problems. So we use the memorized search and greedy strategy to solve the robot attack inefficient, stationary and other problems. Then, the algorithm was optimized to solve the problem of chaotic behavior of the robot after the territory was divided and the problem that the robot did not attack immediately after finding the enemy's KING area. Finally, the effectiveness of the robot program is verified by simulation battle.

Of course, there are still some problems with our robots. Because of the greedy strategy, although the computational efficiency and local optimality are guaranteed, the global optimality cannot be guaranteed. This paper only studies how to improve the attacking efficiency of the robot. Expanding and defending effectively is one of the challenging issues for the future. Besides, intelligent battle gaming pragmatics with belief network trees, which was raised by C. G. Looney [17 18] might be more effective in solving the decision problems of complex games since this algorithm can make move decisions based on the incomplete and uncertain model.

REFERENCES:

[1] C. Tang, Z. Wang, X. Sima and L. Zhang, "Research on Artificial Intelligence Algorithm and Its Application in Games," 2020 2nd International Conference on Artificial Intelligence and Advanced Manufacture (AIAM), 2020, pp. 386-389.

[2] P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, July 1968.

[3] T. Joppen, M. U. Moneke, N. Schröder, C. Wirth and J. Fürnkranz, "Informed Hybrid Game Tree Search for General Video Game Playing," in IEEE Transactions on Games, vol. 10, no. 1, pp. 78-90, March 2018.

[4] A. K. Mishra and P. C. Siddalingaswamy, "Analysis of tree based search techniques for solving 8-puzzle problem," 2017 Innovations in Power and Advanced Computing Technologies (i-PACT), 2017, pp. 1-5.

[5] M. Neshat, "A Hybrid Method in Informed Search: Fuzzy Simplified Memory-Bounded A* Approach," 2010 International Conference on Computational Intelligence and Communication Networks, 2010, pp. 105-109.

[6] C. B. Browne, "A survey of Monte Carlo tree search methods", IEEE Trans. Comput. Intell. AI Games, vol. 4, no. 1, pp. 1-43, Mar. 2012.

[7] M. P. Strub and J. D. Gammell, "Adaptively Informed Trees (AIT*): Fast Asymptotically Optimal Path Planning through Adaptive Heuristics," 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 3191-3198.

[8] A. P. Gerdelan and N. H. Reyes, "Synthesizing Adaptive Navigational Robot Behaviors using a Hybrid Fuzzy A* Approach" in Advances in Soft Computing: Computational Intelligence: Theory and Applications, Berlin & Heidelberg:Springer, pp. 699-710, 2006.

[9] S. Koenig and M. Likhachev, "Adaptive A*", Proceedings of the International Converence on Autonomous Agents and Multiagent Systems (AAMAS), pp. 1311-1312, 2005.

[10] J. D. Gammell, S. S. Srinivasa and T. D. Barfoot, "Batch Informed Trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," 2015 IEEE International Conference on Robotics and Automation (ICRA), 2015, pp. 3067-3074.

[11] Tan Shan Yan, "Greedy algorithm based on Hybrid Genetic Algorithm for Solv¬ing 0 / 1 knapsack poblem", Research and Development, vol. 7, 2006.

[12] K. Wang, W. Shang, M. Liu, W. Lin and H. Fu, "A Greedy and Genetic Fusion Algorithm for Solving Course Timetabling Problem," *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*, 2018, pp. 344-349.

[13] Murat Kalender et al., "A greedy gradient-simulated annealing hyper-heuristic for a curriculum-based course timetabling problem", *12th UK Workshop on Computational Intelligence (UKCI)*, 2012.

[14] C. E. Leiserson and Tao B Schardl, "A Work-Efficient Parallel Breadth-First Search Algorithm (or How to Cope with the Nondeterminism of Reducers)", *ACM Symp. on Parallelism in Algorithms and Architectures*, 2010.

[15] C. Y. Lee, "An Algorithm for Path Connections and Its Applications", *IRE Transactions on Electronic Computers*, 1961.

[16] S. Russel and P. Norvig, "Artificial Intelligence: A Modern Approach (2nd ed.)", *Prentice Hall*, 2003.

[17] C. G. Looney, "Intelligent Battle Gaming Pragmatics with Belief Network Trees," *2006 IEEE Symposium on Computational Intelligence and Games*, 2006, pp. 243-248.

[18] C. G. Looney and L. R. Liang, "Cognitive situation and threat assessments of ground battlespaces", *Info. Fusion*, vol. 4, pp. 297-308, 2003.