Undergraduate Project Report
2022/23

The Program of Chinese Chess

Name:

School:

Class:

QMUL Student No.:

BUPT Student No.:

Programme:

Date: 27-04-2023

Table of Contents

Abstract

The Chinese Chess, also known as "Xiangqi," is an intangible cultural heritage with a long history and rich culture. It is not only a crystallization of Chinese wisdom but also an important research topic in the field of artificial intelligence and game theory. This paper presents the design and implementation of a Chinese Chess computer game program based on the Java programming language, aiming to better understand and inherit this culture while familiarizing oneself with the development process and methodology of Java projects and deepening the understanding of artificial intelligence-related knowledge.

With the rapid development of artificial intelligence, the Internet, and hardware technologies, Chinese Chess has gradually transitioned from offline to online gaming. An increasing number of chess enthusiasts are no longer satisfied with face-to-face matches, as online gaming provides them with more flexible game times and opponents of similar skill levels. Players can either compete against computer AI or participate in online matches with others through the ever-expanding Internet. In the design and implementation of this program, artificial intelligence search algorithms are employed to enable intelligent calculation in human-computer matches, allowing players to compete against computer with a certain level of chess skills.

By utilizing various interfaces provided by the Java programming language, I have designed and implemented the human-computer module and created a user-friendly graphical user interface and functional logic to enhance the gaming experience for players. After a series of tests, the search algorithm can respond to various chess moves of players and analyze different situations to find better moves. Although the AI can defeat some chess enthusiasts, there are still some shortcomings, such as longer thinking times and difficulty predicting situations four steps ahead. These deficiencies will be further optimized and improved in future research.

Keywords:

Chinese Chess ; Man-machine game; Java; Alpha-beta pruning,;Evaluation function

摘要

中国象棋，是一种拥有悠久历史和文化的非物质文化遗产。它不仅是中华民族智慧的结晶，还成为当今人工智能领域博弈研究的重要课题。本文通过设计与实现一款基于 Java 语言的中国象棋人机对战程序，旨在更好地理解和传承这一文化，同时熟悉 Java 项目的开发流程、方法并深入学习人工智能相关知识。

随着人工智能、互联网和硬件技术的飞速发展，中国象棋逐渐从线下转向线上游戏。越来越多的象棋爱好者不再满足于面对面的对弈，线上游戏为他们带来了更灵活的游戏时间和实力相近的对手。在本游戏程序的设计与实现中，采用了人工智能搜索算法实现人机对弈的智能计算，使玩家能够与具有一定棋力的计算机展开博弈。

通过利用 Java 编程语言提供的各种接口，我设计并实现了人机模块，创建了用户友好的图形用户界面和功能逻辑。经过一系列测试，搜索算法能够应对玩家的各种走棋方式，并分析不同局面以找出较优的走子。尽管搜索算法能够战胜一些象棋爱好者，但仍存在一些缺陷，如思考时间较长，难以预测四步之后的局面等。这些缺陷将在未来的研究中得到进一步优化和改进。

关键词:

中国象棋;人机对战;Java;Alpha-beta 剪枝;评估函数

Chapter 1: Introduction

For centuries, people have been placing the Four Arts of the Chinese scholar - music, chess, calligraphy, and painting - on an equal footing, and the art of Chinese chess is considered one of the treasures of the Chinese nation. Combining elements from various domains, including sports, art, science, and culture, chess art is a unique flower of traditional Chinese culture. Due to its simplicity and strong appeal, Chinese chess has become a popular chess activity worldwide and is recognized as one of China's official individual sports. In ancient China, chess was regarded as an art of self-cultivation for literati, while chess games were considered as beneficial activities for both body and mind. In the course of chess games, people can draw certain philosophies from the complex changes in relationships such as offense and defense, reality and deception, and whole and part. It is evident that in many respects, chess art is incomparable to other forms of art[1]. In recent years, with the rapid development of computer science and artificial intelligence, chess algorithms and games based on various platforms have emerged. Some of these works have been recognized and loved by a large number of chess enthusiasts. In order to better understand and pass on this culture, as well as to familiarize oneself with the development process and methods of Java projects and to delve into knowledge related to artificial intelligence, this paper makes full use of existing technology and algorithms to develop a human-computer chess game based on the Java language.

Java, as a cross-platform programming language, has been widely favored by programming enthusiasts due to its platform diversity. It has become one of the mainstream programming languages, maintaining an unshakable position even in the current context of various new and excellent programming languages emerging one after another. In game development, Java has provided great convenience for many developers in their development process. In simple terms, it is a programming language that can be used to write cross-platform applications. Like C++, it is also an object-oriented programming language. Generally, when people refer to Java, they are referring to the programming language and the Java platform (i.e. JavaSE, JavaEE, JavaME) introduced by Sun Microsystems in May 1995. Java technology has superior general performance, and its efficiency, platform portability, and security provide favorable conditions for its promotion. Currently, there are billions of devices running Java programs worldwide, spanning various fields such as personal computing, data centers, game consoles, scientific supercomputers, mobile phones, and the Internet[2]. At the same time, with

the rapid development of computer science and the continuous improvement of hardware performance, together with the development of emerging disciplines such as cloud computing and artificial intelligence in the global Internet industry, Java's series of characteristics have given it significant advantages and broad application prospects. The Chinese Chess game designed and implemented in this paper is developed based on the Java language.

In modern society, numerous enthusiasts and researchers of Chinese chess have begun to explore the algorithmic essence of this traditional game. In the process of exploration, an increasing number of algorithm implementations have been applied to this game of intelligence, among which the most representative is artificial intelligence (AI), which was initially proposed at the Dartmouth Conference in 1956. Since then, researchers worldwide have developed numerous related theories and principles, and the concept of AI has also expanded. Its popular definition is a new technology science that studies and develops theories, methods, techniques, and application systems for simulating, extending, and expanding human intelligence[3]. Simply put, the goal of AI is to enable computers to perform problem analysis and self-centered reasoning similar to humans. AI is an important branch of computer science, which, by gaining a deep understanding of the nature of intelligence, produces intelligent machines that can respond in ways extremely similar to human intelligence. Currently, the main research work in this field includes robotics, speech recognition, image recognition, natural language processing, and other directions[4]. In the design and implementation of this program, algorithms are the core content of the human-machine battle part and also one of the most challenging development stages of this game.

Currently, many game companies at home and abroad have developed similar products, among which mobile device games are the most common, such as mobile games based on the IOS and Android systems. Some well-known examples include "Tencent Chess", which mainly focuses on online multiplayer battles, but also includes a single-player mode, making it an important reference for the development of this game. In desktop applications, single-player games are more common, such as "Chess Wizard". These games are written in different programming languages, but their results and functions are very similar. In contrast, in web games, due to their network activity, they mainly adopt network battle mode. In many of these Chinese chess games, a large number of chess manuals and empirical results are used in the human-machine battle mode, and only a few use dynamic algorithms (such as AI algorithms) to dynamically respond to game user operations. In the Chinese chess game designed and implemented in this paper, artificial intelligence algorithms are directly integrated into the

human-machine battle mode using the Java language, without relying on external chess manuals.

On the basis of previous research, this paper focuses on studying commonly used design methods and approaches in game design, particularly the various operations of classes and the operation design of the Swing interface for graphical user interfaces. The work is centered on enhancing the user experience and designing and implementing algorithms, and a new game interface for Chinese Chess is redesigned. Based on game requirements, the implementation methods for each functional module are detailed, and the core part of the game implementation for human-machine competition is designed and implemented. During the algorithm design and implementation process, the search algorithm was particularly considered. Drawing on game theory from Go and Gomoku, a comprehensive suite of algorithms suitable for Chinese Chess was developed. By combining the human-machine module with game interface controls, the system is able to simulate the movement of chess pieces and dynamically alter the game board. Overall, the work done in this paper fully utilizes limited reference resources and experiences, and implements a Chinese Chess game for human-machine competition using artificial intelligence-related theories.

The following structure will be presented in this paper:

Chapter 2: Background Introduction. This chapter will provide a detailed introduce the rules and gameplay of Chinese chess, provide an overview of Java technology, and discuss the artificial intelligence algorithm used in this study.

Chapter 3: Technical Design and Implementation. This chapter will provide a detailed introduction to the specific characteristics of Chinese chess and technical design aspects, including user interaction, game module division, and AI module design. Additionally, this chapter will focus on the process of technical implementation.

Chapter 4: Testing and Result Analysis. This chapter will briefly introduce the testing methods and results of the game and analyze the conclusions drawn from the study.

Chapter 5: Conclusion and Future Work. This chapter will summarize the main contributions of this paper and propose possible future work.

Finally, the acknowledgments section will be included in this paper.

Chapter 2: Background

The design and implementation of this program involve various techniques and knowledge in the field of computer science. This chapter provides a detailed introduction to these fundamental concepts. Firstly, as the game design and rules are closely related to gameplay and winning strategies, the rules and relevant winning strategies of Chinese chess will be briefly introduced. Secondly, game development is built on the Java platform, therefore, it is necessary to have a clear understanding of this fundamental platform before the project begins. Finally, the two major modules of game design - player interaction and AI module - use Java Swing technology and search algorithm techniques in artificial intelligence. Therefore, a detailed introduction to the fundamental knowledge of these two core modules will be provided.

2.1 The Gameplay of Chinese Chess

Understanding the rules and gameplay of Chinese chess is crucial for the design and implementation of human-machine chess playing projects. Chinese chess is a two-player board game consisting of 16 pieces for each side, including seven types: rook, knight, cannon, bishop, guard, king, and pawn. The board is a square with 90 intersections formed by nine horizontal and ten vertical lines. Players move their pieces to attack their opponent's pieces and protect their own pieces, ultimately capturing their opponent's king and winning the game. Each piece has specific movement rules, such as the rook can only move in straight lines, and the cannon must jump over one piece to capture another. Additionally, Chinese chess has special rules such as check and double check. A deep understanding of these rules and strategies is necessary to succeed in the game. In the design and implementation of a Chinese chess project, a clear understanding of the game rules and gameplay is essential. For instance, the use of Java's inheritance and polymorphism can accommodate the various types of chess pieces, and a chessboard model can be designed based on the board's dimensions. Furthermore, the movement of the pieces can be restricted according to their specific rules.

2.2 Introduction to Java Platform

People usually refer to the Java platform as consisting of the Java Virtual Machine and the Java core classes. It provides a unified programming interface for pure programs to run as a virtual machine on top of the underlying operating system, regardless of what the operating system is. This also explains the fundamental reason for cross-platform compatibility, which refers to a programming language that can run on different operating systems such as

Windows, Unix, etc. At the application level, a platform typically refers to the system software that runs on various terminals such as servers, mobile devices, and embedded devices. In addition, the language is usually referred to as Java[5].

2.2.1 Java System Architecture

Java architecture mainly includes several distinct but interrelated technologies, which are Java Virtual Machine technology, Java class file, Java Application Programming Interface (API), and Java programming language. The relationship among these four components is illustrated in Figure 1: Java Architecture.

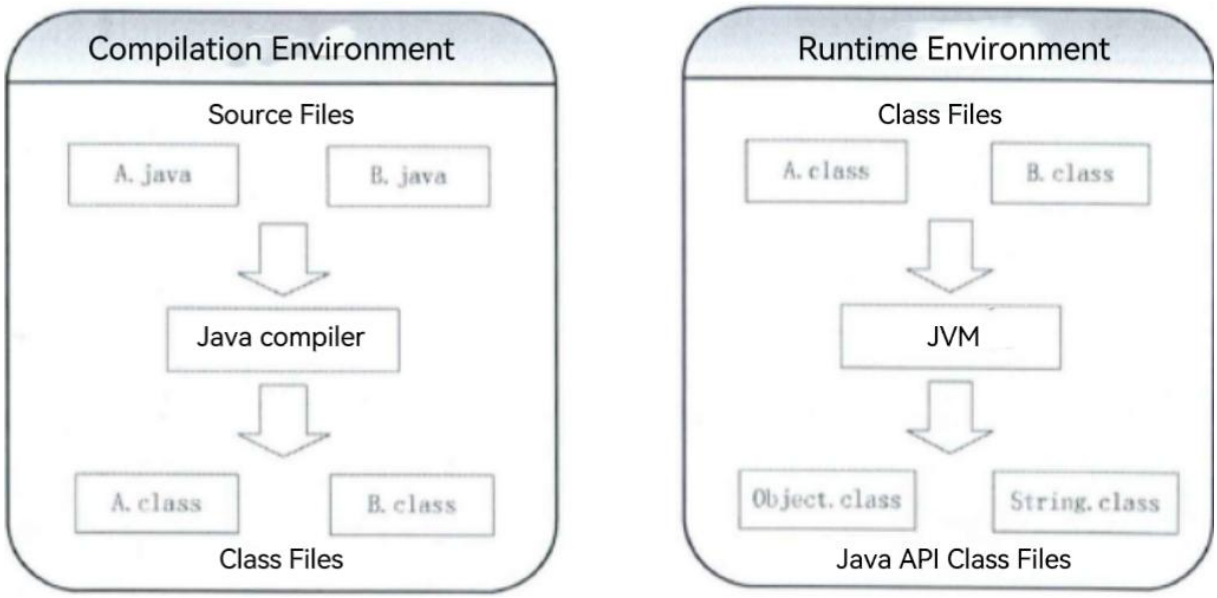


Figure 1: Java Architecture

(1) JVM

The Java Virtual Machine (JVM) is the core technology that enables Java programs to run on different platforms. Instead of directly executing on the local operating system of the machine, Java programs are interpreted into local machine language and executed using machine language on the local operating system. Essentially, the JVM is an abstract computer built on top of actual processors and serves as the heart of program execution. Many of the JVM's excellent features stem from its concept and implementation. For example, it ensures the portability of code across different operating systems, such as Windows and Unix. However, achieving this benefit comes at a cost of slower running speed compared to code that does not require JVM interpretation. Nevertheless, researchers unanimously believe that the time cost is acceptable given the practical problem of increasingly large software and rapidly improving

computer hardware performance, as well as the booming application development. The implementation of the JVM is highly flexible, with the JVM specification defining the characteristics that each JVM should have. Platform developers can use different technologies to implement JVM functionality.

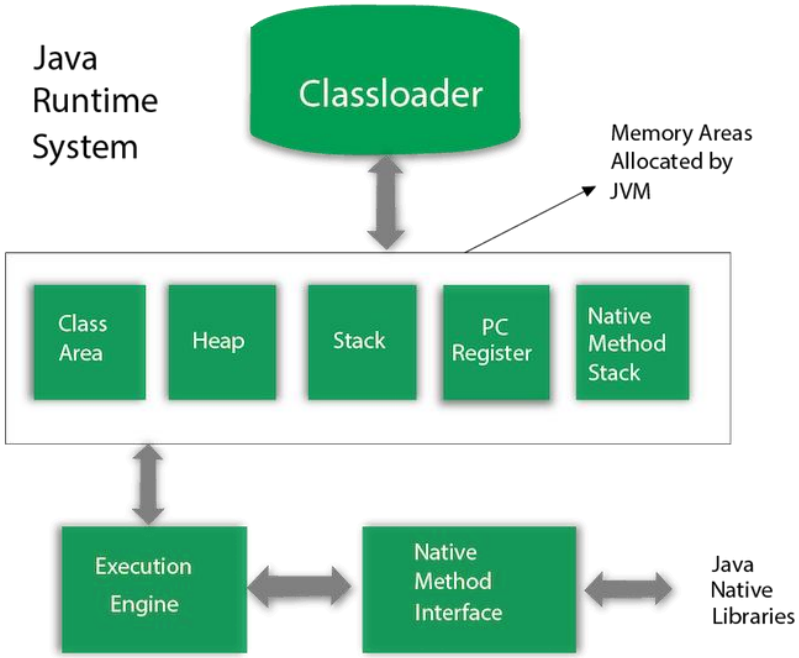


Figure 2: JVM Architecture

The primary function of the JVM is to load class files and execute their bytecode. As shown in Figure 2: JVM Architecture , the JVM includes a class loader that is responsible for loading user program class files and class files in the Java library. During program execution, only these class files need to be loaded onto the JVM, and the bytecode contained in the class files is executed in the execution machine.

(2) Java Class Files

Java class files are the binary representation of Java programs. Each class file represents a class or interface, and it is impossible to include multiple classes or interfaces in a single class file. This allows class files to be generated on one platform and executed on any other platform. Although class files are part of the Java architecture, they are not inherently tied to the Java language. Other programming languages can also be compiled into class files, and Java programs can be compiled into other binary formats. Java class files contain all the information about classes and interfaces that is needed by the virtual machine.

(3) Java API

API (Java Application Interface) is an application programming interface developed by Sun Microsystems, which consists of a series of programs that are readily available for use by programmers. As part of the architecture, APIs are not associated with these interfaces, but rather represent a ready-made library provided directly to developers for use in programming[6]. Without these directly callable APIs, development would become much more difficult, as different developers would have to use different methods to implement certain functions, making it difficult to establish a standardized development convention as it exists today. Similar to Java class files, APIs encapsulate many functions, providing only information such as function names and parameters without providing specific entities, but users can still use them. The API provides a standardized programming interface, allowing developers to use the methods provided by the API without worrying about underlying implementation. The API includes various common functional modules, such as graphical interfaces, network communications, and data storage, enabling developers to quickly develop feature-rich and maintainable applications. Finally, it should be noted that because APIs are open source, developers can directly view the specific implementation of various methods in the library, providing great convenience for deeper understanding of the underlying implementation.

(4) Java Program Language

Java, a versatile programming language, combines simplicity, object-orientation, distribution, interpretation, resilience, security, portability, and efficient multithreading. Launched in 1995, it quickly captivated developers across the globe with its unique features and capabilities. Over the past three decades, Java has evolved into a more mature and high-performing language.

Java's platform independence sets it apart from other programming languages, allowing it to run on various platforms. This is achieved through the implementation of a virtual machine, which acts as an interface between different platforms and enables Java programs to be shared worldwide. Additionally, Java supports machine-independent data types.

Security is another strength of Java, as it does not rely on direct memory address operations using C++ pointers. Instead, memory allocation is managed by the operating system, shielding the system from potential virus attacks via pointers. Java also incorporates a security manager to prevent unauthorized program access.

Inheriting object-oriented concepts from C++, Java encapsulates data within classes,

leveraging the benefits of classes to ensure simplicity and maintainability. Class features such as encapsulation and inheritance enable Java code to be compiled once and reused. Developers can then concentrate on designing and implementing classes and interfaces. Java offers numerous generic object classes, which can be utilized through inheritance to access parent class methods. In Java, the inheritance relationship between classes is single and non-multiple, meaning a subclass can have only one parent class, which in turn may have another parent class. Java's Object class and its subclasses form an inverted tree structure, with the Object class at the root, providing powerful functionality when used alongside derived subclasses.

Java is also designed to work on the extended TCP/IP network platform, with its library functions offering methods for sending and receiving information via HTTP and FTP protocols. This enables programmers to access network files as easily as local ones.

Lastly, Java possesses the capability to identify errors within programs at both compilation and execution stages. By conducting type checking, it uncovers numerous mistakes that arise in the initial phase of development. Java's inherent management of memory minimizes the likelihood of memory-related issues. Moreover, it utilizes genuine arrays, thereby eliminating the risk of data being inadvertently overwritten. These beneficial features significantly enhance the efficiency and life cycle of program development and deployment.

2.2.2 Java Technology Architecture

According to different usage scenarios, Java platform provides different versions for developers to choose from, including Java Enterprise Edition (JEE), Java Standard Edition (JSE), and Java Micro Edition (JME). These three versions have different applications. JEE is mainly used for developing distributed network programs and is usually used for server-side programming. JME is a mini version suitable for small application development, mainly used for desktop application software programming. JSE is mainly used for embedded system development, such as various mobile device programming, especially for mobile application development.

It is worth mentioning that the Java Development Kit (JDK) is a powerful tool used for Java development. JDK is widely used in the development of the above three technology branches. JDK is the core of the entire Java platform, including the runtime environment, tools, and basic libraries. Starting from Java 5, JDK provides practical features such as generics, and its

version is constantly updated, greatly improving the running efficiency. Another tool worth mentioning is the Java Runtime Environment (JRE), which is usually integrated into JDK. However, if the user only needs to execute the program without developing the code, installing JRE is sufficient.

2.3 Introduction to Alpha-Beta Search Algorithm

Artificial intelligence has gone through a long development process, during which researchers have continuously pushed for technological advancements, enabling AI to simulate human intelligence. Various chess games have played an important role in the development of AI, such as AlphaGo, which is considered one of the milestones in the history of AI. AlphaGo used deep reinforcement learning and search algorithms to defeat the world champion of Go, Lee Sedol, demonstrating the powerful ability of AI in solving complex decision-making problems.

Therefore, this paper uses AI algorithms to implement man-machine game playing. The research on AI algorithms has developed from basic to advanced, and many excellent algorithms have been developed by different researchers, which together form the core of the entire field of AI. Search algorithms are an important type of AI algorithm, mainly applied in chess games. In the platform-based Chinese chess game designed in this paper, the man-machine game playing mode adopts the basic ideas of search algorithms. Therefore, it is necessary to introduce the basic principles of this algorithm.

It is worth noting that this paper uses traditional search algorithms instead of machine learning or deep learning. Despite the continuous advancement of technology by researchers, search algorithms remain an important application in chess games and are one of the core algorithms in the field of AI. Therefore, in the platform-based Chinese chess game designed in this paper, we adopt the basic ideas of search algorithms as the core of the man-machine game playing mode, to introduce the basic principles of search algorithms.

2.3.1 The Basis of The Algorithm

Before introducing the search algorithm, it is necessary to mention the "Minimax" algorithm. This algorithm uses a tree search to represent a search function. For any given Chinese chess board, the current board is viewed as the root of the tree, and each move is seen as a branch of the tree, forming a very large tree. Starting from the root node, selecting any move will create a new board position through a particular branch. Each new board position will have many

new branches, each representing a move in that new position. Chinese chess can be used as an example to explain this tree model. In the Chinese chess tree model, each position is represented as a node, with red nodes indicating it is Red's turn to move, and black nodes indicating it is Black's turn to move. The lines connecting the nodes represent different moves, with each move creating a new position and forming a new node in the tree[7].

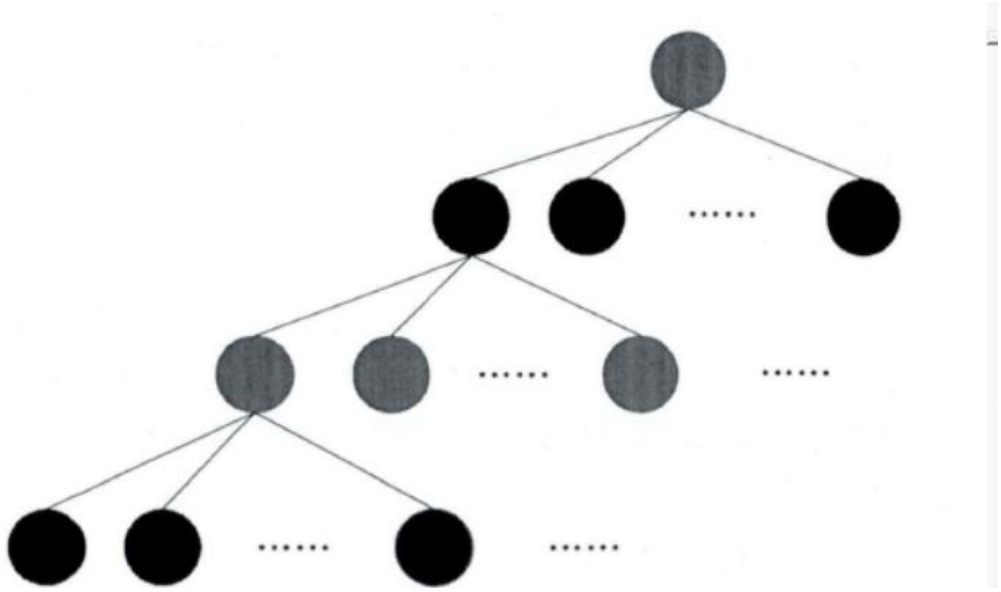


Figure 3: N-branch tree

In this paper, we consider using a score value as an analysis criterion. Assuming that there are two players, A and B, in a game of Chinese chess, we assign a very large positive number to the winning position of A and a very large negative value to the winning position of B. For the tied position, we assume a value with an absolute value within a certain range, such as when the score value is less than 0.1, it is considered a tied game. Suppose a score value is assigned to each position in the tree. According to the above assumptions, when A moves, he will try his best to make the score value of the position as large as possible, while B will try to make the score value of the position as small as possible. This is the basic idea of the "Minimax" algorithm. In Figure 3: N-branch tree, taking the red side as an example and the black side as the opponent, since A wants the score value on the chessboard to be as large as possible, he will choose the node with the maximum score value each time he moves. Similarly, B wants the score value to be as small as possible, so he will choose the node with the minimum score value when he moves. However, in Chinese chess,the average number of moves in the mid-game is about 40. Therefore, searching 4 plies requires checking 2.5 million routes, searching 5 plies requires checking 100 million routes, and searching 6 plies requires checking 4 billion routes[7]. Due to the large number of pieces and flexible moves, the constructed tree can be

very large, and searching all the paths of the entire tree will require a lot of time and resources. Therefore, to reduce the time requirement, we need to improve the "Minimax" algorithm and reduce the branches of the tree, which is known as the Alpha-Beta search algorithm.

2.3.2 The Core of The Algorithm

The Alpha-Beta search algorithm is a development of the “Minimax" algorithm. Compared with the minimax algorithm, it reduces the number of search branches by using the following basic idea: if a good choice has already been made, there is no need to determine the exact score of other choices if it can be determined that they cannot be better than the current choice. At the same time, any choice that is not better than the best choice found can be considered sufficiently poor and can be abandoned directly. Therefore, the main improvement of the Alpha-Beta algorithm compared with the Minimax algorithm is to reduce the number of search branches.

Two parameter values, Alpha and Beta, are needed in the search process. The Alpha value is the best value that has been found in the search, because it is the best value that can be obtained by the current strategy, and any value smaller than it will not be used. The Beta value is the worst value for the opponent. If a value returned during the search process is better than or equal to Beta, it is good enough and the other side will not have the opportunity to use this strategy.

During the search for moves, each move searched returns a value associated with it, and the relationship between Alpha and Beta values is very important, which may cause the search to stop and return. If the score of a move is less than or equal to Beta, it is a very poor move and can be abandoned directly. If the score of a move is greater than or equal to Alpha, the entire node is discarded because the opponent has other moves to avoid reaching this situation. If the score of a move is greater than Beta but less than Alpha, the move is one that the player can consider, but it needs to be constantly updated to reflect new situations. Sometimes, a reasonable move may have a score that is not greater than Beta, in which case another move should be chosen at the previous level of the game tree to avoid this situation[8].

To clarify the essence of the algorithm, an example is presented below (see Figure 4:Alpha-Beta Pruning Case). Suppose that the game board has developed to node a, and it is A's turn to move. First, examine the child nodes of the node. The layer to which node b belongs is the situation where the black side is moving first. First, consider each child node b of node a in turn. The first child node of node b returns 10, the second child node returns 6, and the third

child node returns 2. Since the layer of node b is selected by B, whose goal is to make the score of the game board as small as possible, he will choose the node corresponding to the smallest return value, which is 2. This value is finally transmitted back from node b.

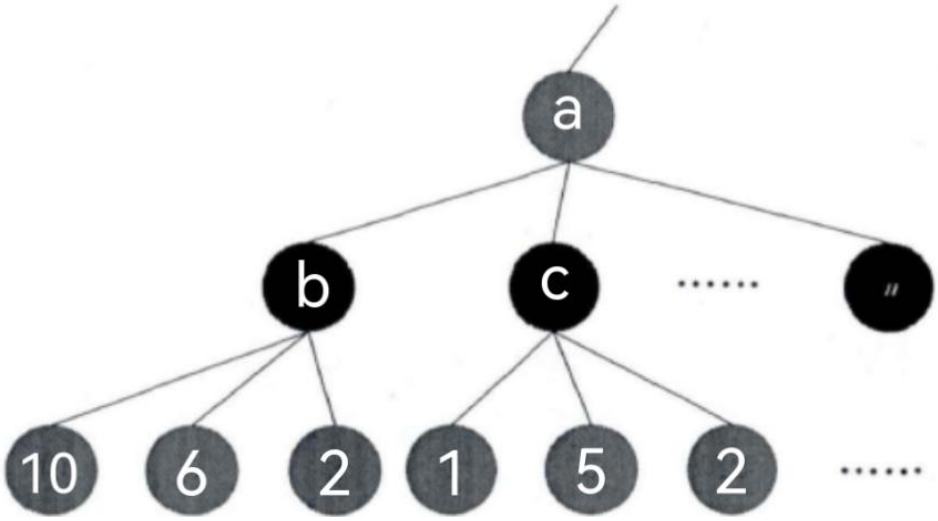


Figure 4:Alpha-Beta Pruning Case

Next, A analyzes the second child node c of the node. Node c is also a situation where it is black's turn to make a move. Examine the score values of each child node of node c in turn, and the first child node returns 1. Now the "pruning" function can be used. There is no need to consider the remaining child nodes of node c because node c is already very bad and can only return a maximum of 1. Compared with the value returned by node b, it is clear that the red side does not want to see the value of 1 appear in the game board. Therefore, the red side will not choose a move that leads to the development of the game to node c, because if they do, the next move of black will bring a game board score of no more than 1 for the red side. In this way, the "useless" child nodes of node c are avoided without affecting the quality of the search, which saves valuable time and provides the possibility of searching more layers under the same machine configuration.

2.4 Introduction to Evaluation Function

During the alpha-beta pruning search, each leaf node of the game tree has a value used to evaluate the current board position, which is called the evaluation function. The evaluation function scores the board position at a fixed search depth and helps the search engine determine the selection of nodes and make correct choices. The evaluation function is one of

the most critical components of computer game systems, and its quality can significantly affect the development trend of the game. If the search engine enables the program to see further, then the evaluation function allows the program to see more accurately.

2.4.1 The Game Performance with Evaluation Function

Chinese chess game programs can be divided into two categories: knowledge-based programs and speed-based programs. Knowledge-based programs are designed to compensate for the shortcomings of search algorithms by incorporating a large amount of chess knowledge into complex evaluation functions, resulting in higher performance as the knowledge base increases. On the other hand, speed-based programs use simple evaluation functions to achieve deeper search depth and win through speed, resulting in higher performance as the search depth increases at the same level of knowledge. Figure 5:The Relationship between Knowledge and Speed illustrates the relationship between speed and knowledge, which are inversely proportional and contribute to the overall chess-playing ability of the program.

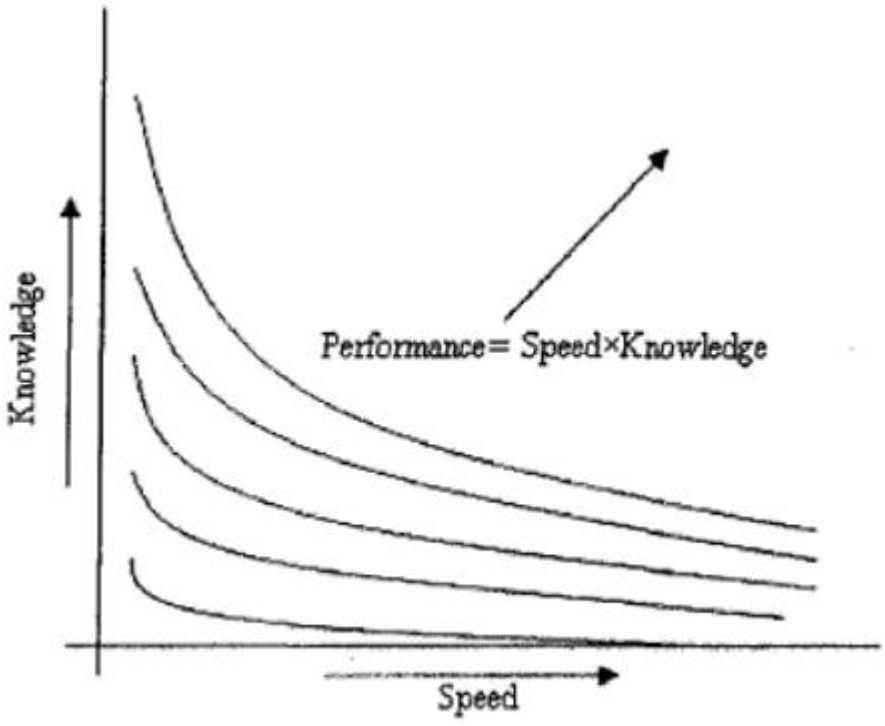


Figure 5:The Relationship between Knowledge and Speed

It is evident that the amount of knowledge and the speed are inversely proportional, and the chess-playing ability of the program is the product of the two[9].That is:

$$Performance=Speed*Knowledge \quad (1)$$

This formula has theoretical significance, indicating that both overly complex and overly simple evaluation functions are not ideal. Achieving a balance between speed and knowledge is the goal of program designers, but it is usually difficult to determine the appropriate level of knowledge for an evaluation function. Designers need to conduct extensive experiments to determine which knowledge to incorporate into the evaluation function.

2.4.2 Factors of Evaluation Functions

Given the complexity and flexibility of Chinese chess, the construction of an evaluation function requires the consideration of multiple factors for a comprehensive assessment. Such factors include the intrinsic values and flexibility of each chess piece, their positional impact on the overall board state, as well as the mutual interactions among different pieces.

(1) The evaluation of fixed pieces value

The concept of pieces value refers to assigning a fixed value to each chess piece on the board based on its importance to the situation and the richness of its moves. The pieces value is the primary factor in measuring the advantage or disadvantage of a situation, with more pieces value generally indicating an advantage. For example, based on experience, the combat power of the chariot is the strongest, and its value is slightly higher than that of the cannon and horse. The value of the cannon and horse is roughly equivalent to that of the advisor and elephant, and the value of the two advisors and elephants is equivalent to that of the double pawns, and so on. Additionally, in Chinese chess, the pieces value of the king and general is assigned an infinite value, usually much higher than the pieces value of other pieces, because the goal of the game is typically to capture the opponent's king or general. Due to different strategies and habits chosen by players, there may be some differences in the estimation of pieces value, even for the same piece. For example, a pawn may be as valuable as a horse or cannon in the eyes of a skilled player. Therefore, the evaluation of pieces value should fully consider the characteristics of algorithms and programs. Through reviewing literature, this program sets the fixed pieces values of each chess piece, as shown in Table 1 Fixed Piece Values for each Chess Piece.

Table 1 Fixed Piece Values for each Chess Piece.

Type	Chariot	Knight	Cannon	Elephant	Guard	Soldier	King
Value	1300	490	610	200	200	100	10000

(2) The evaluation of position value

The position occupied by a chess piece on the board is an important piece of information in the evaluation function, as the same piece can have different effects depending on its location. Generally, a piece is considered more valuable if it is closer to the opponent's "palace," as it exerts greater control over the opponent's pieces and poses a greater threat. For example, the attack power and threat to the opponent of a pawn before and after crossing the river are clearly different, and a pawn that enters the opponent's "palace" may be worth more than a rook. The threat posed by a piece to the opponent's pieces also varies depending on its position on the board. A rook that stays in the corner of the board is not very effective in controlling the situation, and a horse that is on one's own side of the board may not pose much threat to the opponent, but can pose a serious threat to the opponent's king or general near their "palace." A cannon can pose a serious threat to the opponent when it is in the central position or on the same line as the opponent's king or general. Moreover, the impact of the same piece on the situation can be different in the opening, middle, and endgame stages of the game. For example, a pawn in the endgame can even checkmate the opponent if used properly, and the number of pawns that have crossed the river is critical in the middle game and endgame. These pieces of information should all be reflected in the evaluation function. Figure 6 The position score of black pawn in middle game and Figure 7 The position score of black pawn in end game show the position values assigned to the black side's pawns in the middle and endgame in the design of the evaluation function in this program.

```
public  final int[] blackSoldierAttach ={

        0  ,0  ,0  ,0  ,0  ,0  ,0  ,0  ,0
        ,0  ,0  ,0  ,0  ,0  ,0  ,0  ,0  ,0
        ,0  ,0  ,0  ,0  ,0  ,0  ,0  ,0  ,0
        ,0  ,0  ,0  ,0  ,+10,0  ,0  ,0  ,0
        ,+20,0  ,+45,0  ,+35,0  ,+45,0  ,+20
        ,+80,+100,+110 ,+110,+110,+110,+110,+100,+80
        ,+100,+120,+140 ,+160,+160,+160,+140,+120,+100
        ,+100,+150,+190,+220,+220,+220,+190,+150,+100
        ,+100,+150,+200,+250,+280,+250,+200,+150,+100
        ,+100,+100,+100,+100,+100,+100,+100,+100,+100
};
```

Figure 6 The position score of black pawn in middle game

```
public  final int[] blackSoldierAttach ={

        0  ,0  ,0  ,0  ,0  ,0  ,0  ,0  ,0
        ,0  ,0  ,0  ,0  ,0  ,0  ,0  ,0  ,0
        ,0  ,0  ,0  ,0  ,0  ,0  ,0  ,0  ,0
        ,0  ,0  ,0  ,0  ,0  ,0  ,0  ,0  ,0
        ,+55,0  ,+60,0  ,+60,0  ,+60,0  ,+55
        ,+90,+110,+110 ,+120,+120,+120,+110,+110,+90
        ,+110,+150,+150,+180,+200,+180,+150,+150,+110
        ,+110,+180,+220,+230,+250,+230,+220,+160,+110
        ,+110,+180,+220,+260,+300,+260,+220,+160,+110
        ,+100,+100,+100,+100,+100,+100,+100,+100,+100
};
```

Figure 7 The position score of black pawn in end game

(3) The evaluation of flexible value

The flexibility of a chess piece refers to its range of movement in a specific position, that is, the positions it can move to. The flexibility of a piece directly affects its power, so the greater the flexibility, the better. The flexibility of the chariot, knight, and cannon in particular have a significant impact on the game. For example, obstacles on a chariot's straight path or knights blocked by surrounding pieces reduce their flexibility and limit their power.

The calculation of a chess piece's flexibility involves summing up all the legal moves that the

piece can make on the chessboard. In a given situation p, the flexibility evaluation expression is:

$$F1(p) = \sum mi(p) \quad (2)$$

where f1(p) represents the total evaluation value of flexibility for player 1, and mi(p) represents the number of legal moves that piece i can make in the current situation p.

This program sets a minimum threshold for the flexibility value for each type of chess piece. If the flexibility value is below this threshold, a penalty is applied based on the flexibility coefficient. The flexibility coefficient is also determined based on practical experience. For instance, while the horse can move to a limited number of positions, it often poses a great threat to the opponent, thus receiving a higher flexibility coefficient. Conversely, the car and cannon can move to many positions, but these positions often hold similar significance in the game. Therefore, considering the pieces' threat to the opponent, a lower flexibility coefficient is assigned to the car and cannon. Table 2 Flexible coefficient shows the flexibility coefficients for the car, horse, and cannon.

Table 2 Flexible coefficient

Type	Chariot	Knight	Cannon
coefficient	5	12	2

(4) Evaluation of special chess patterns

The evaluation of special chess patterns refers to specific patterns such as the empty cannon, sunken cannon, and separated-carriage. Generally, the higher the position of the empty cannon, the greater the threat to the opponent and the higher the point value. Based on experience, the empty cannon is most powerful when located in the middle of the player's own palace. In the program, the distance between the empty cannon and the opponent's general is determined, and if it is greater than three points, an additional 100 points are added to the threat value. Additionally, 100 points and 50 points are respectively added for the sunken cannon and separated-carriage.

(5) Other factors influence evaluation value

There are other factors that influence the evaluation function in addition to the factors mentioned above. These include the evaluation of tactics, the evaluation of scoring by region, and the dynamic adjustment of evaluation scores. Typical tactics include double capture tactics, swift attack tactics, and sacrificing pieces for the sake of strategy. When such tactics appear in a game of Chinese chess played by a computer, appropriate additional values are given. Scoring by region refers to dividing the chessboard into multiple zones, such as six zones, which are the left, center, and right zones for attacking and defending relative to both players' generals. The evaluation scores are calculated separately for different zones, which is advantageous for developing an offensive game. Dynamic score adjustment generally adjusts the evaluation score of a single piece according to the number of surviving pieces. For example, the value of a cannon is high when the opponent has all their guards and elephants, but its effectiveness is limited when the opponent does not have these pieces, resulting in a decrease in evaluation score.

Chapter 3: Design and Implementation

3.1 The General System Design and Game Flow

3.1.1 The general design

In the game, based on the functional requirements, three major functional modules were mainly designed: the user interface module, the game logic module, and the artificial intelligence(AI) module. Before delving into the detailed design, the general system architecture was first depicted in a UML diagram, as shown in Figure 8 General System Design UML.This UML diagram illustrates the package structure and class relationships that should be present in this program.

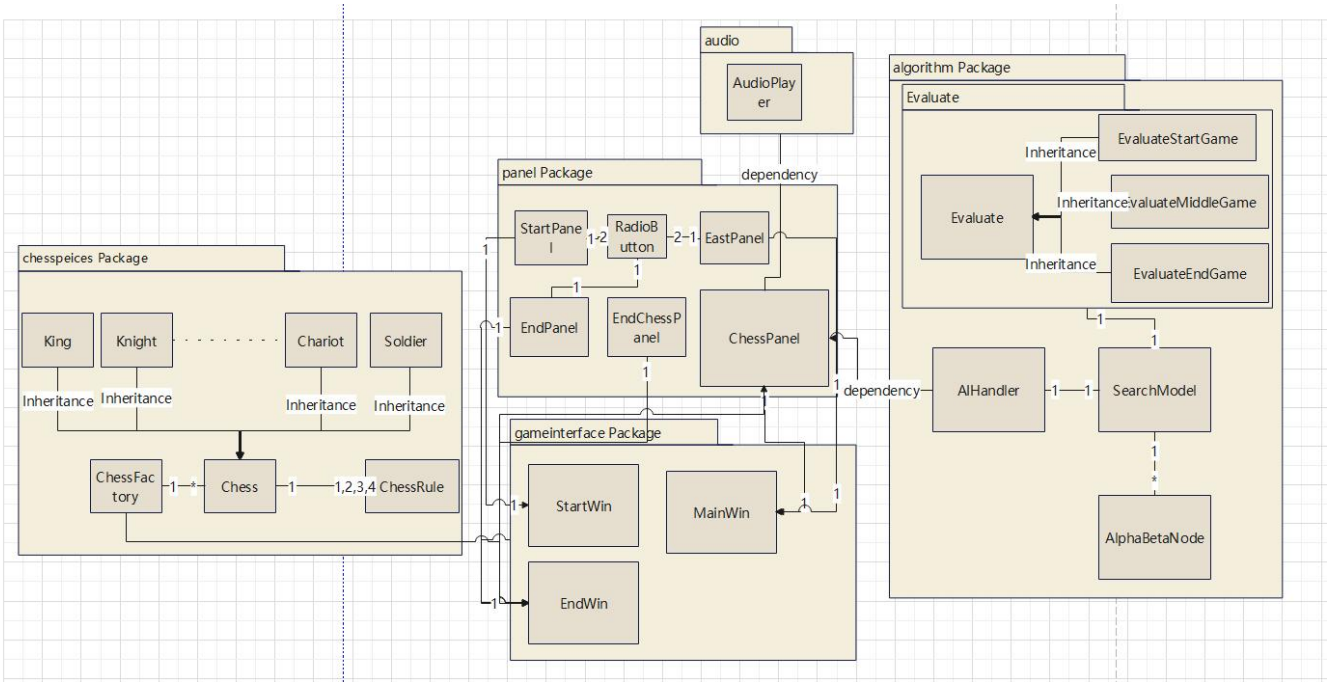


Figure 8 General System Design UML

To better understand the relationships between the classes and the overall structure of the program, the following sections will provide a detailed description of the design and implementation of the three major functional modules.On this basis, each function was refined and more small implementation branches were distinguished, as shown in Figure 9 System Module General Design Diagram.

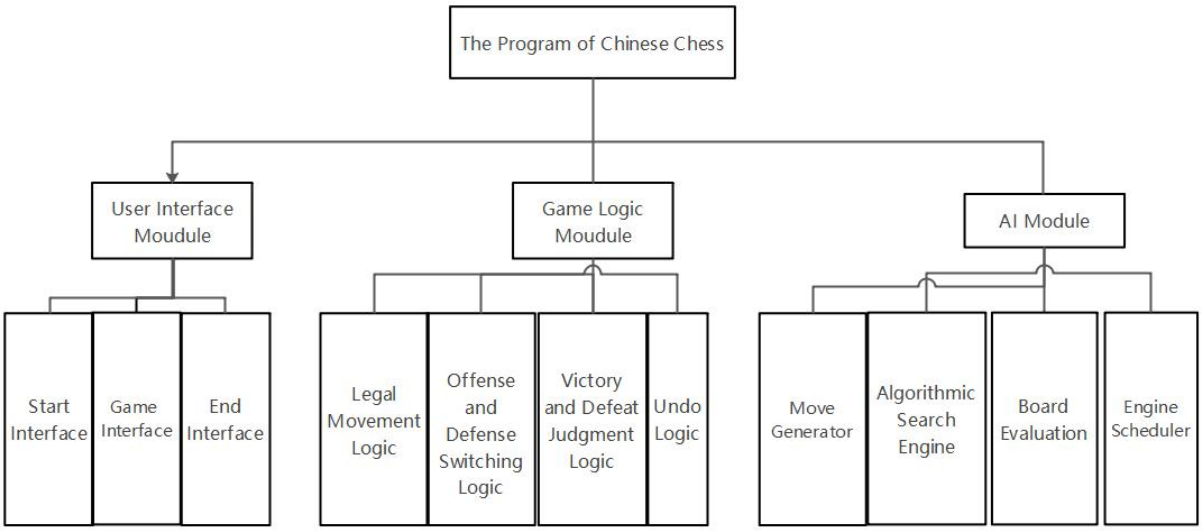


Figure 9 System Module General Design Diagram

3.1.2 The game flow

Based on Figure 9 System Module General Design Diagram, an analysis can be made of the user interface module, game logic module, and artificial intelligence module. In the user interface module, the user first enters the start interface, selects to be the attacking or defending side, and then starts a new game. They then enter the main interface to play against the computer, and the game logic module is used to enforce the rules. Meanwhile, the computer uses search algorithms for intelligent calculation. After the player makes a move, they can choose to retract the move. By default, the computer allows the player to retract the move, which restores the board to the previous state. Of course, the player can also choose to start a new game at any time, in which case all previous moves will not be saved. Following this flow, the game ends when one side wins. After the game ends, the game ending interface will appear, which displays the winner and the state of the board. It also allows the user to start a new game. The user case diagram is shown in Figure 10 User Case Diagram.

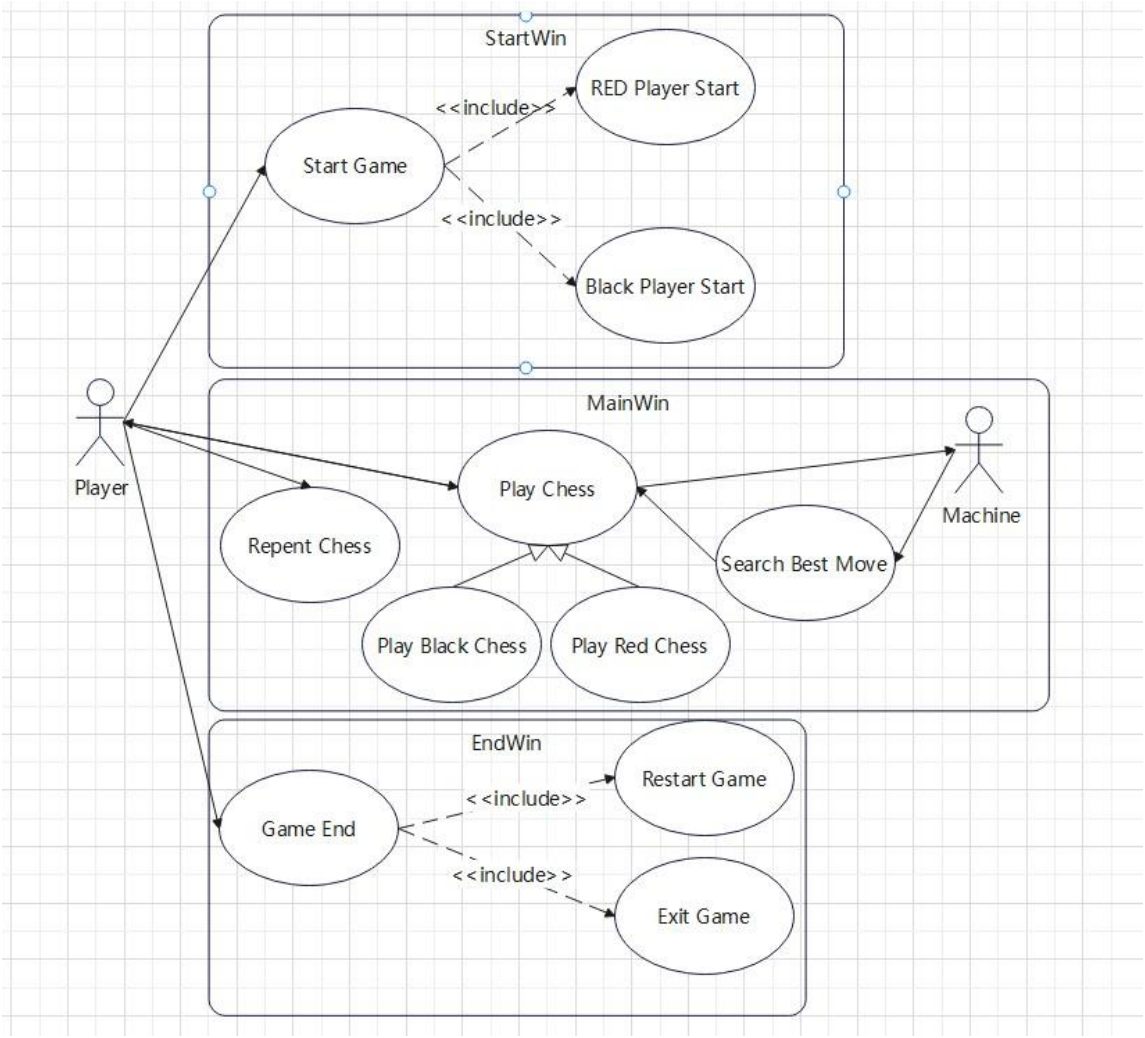


Figure 10 User Case Diagram

3.2 The Detailed Design and Implementation

After completing the general system design, there is a preliminary understanding of the system's overall framework and model. On this basis, detailed planning and design of the system can be carried out. Detailed design is one of the most important steps in software development. The main purpose of this stage is to refine the content of the overall design, that is, to design the specific implementation algorithms and local structures required for each module in detail.

In the Chinese chess game system discussed in this paper, around the three core functional modules, it is necessary to first abstract each component of Chinese chess and design specific storage models, which in Java language means designing independent classes to facilitate the storage and management of the basic components such as chessboard and chess pieces. Second, a user-friendly game main interface needs to be designed, which, in addition to the

chessboard and chess pieces loaded in image form in the main interface, also requires the design of various specific operation controls according to specific operational needs. Third, after completing the above two steps and the overall design closely related to the entire system, it is necessary to implement the intelligent operation of the computer for the human-computer competition module, which requires the specific implementation of search algorithms. Subsequent sections will elaborate on the detailed design and implement of each module based on the overall design: user interface module, game logic module, and artificial intelligence module.

3.3 The details of user interface module

An excellent game application often requires a user-friendly interface that players enjoy. Such an interface is usually referred to as a user-friendly interface. In addition to avoiding challenges to user operating habits, a friendly game main interface plays a crucial role in application promotion and advertising, catching users' attention immediately and often being a common feature of many successful cases. In the design process of the Chinese chess game, the JFrame class of the Java library was used as the basis for the game interface. On this basis, various Java controls were flexibly applied, which coordinated with each other to achieve a beautiful and atmospheric game main interface.

According to the functional requirements of the system, the following three globally-used controls need to be designed first, including the game background, two circular chess buttons for mode selection, undoing and restarting functions, and an audio control for playing game sound effects. The circular chess button shown in Figure 11 Undo Button and Figure 12 Restart Button is used for restarting and undoing game. The button control inherits from the JBotton class, extends the button to a circular shape, and adds custom-designed chess images. Additionally, the button is equipped with click feedback by enlarging the circular area's shadow upon clicking, creating a three-dimensional clicking experience.

Figure 11 Undo Button

Figure 12 Restart Button

The unified modeling approach was employed to construct the UML model, as shown in the diagram, for building a consistent Chinese chess game interface. This model serves as the basis for the overall design of the game interface, providing unified guidance for the subsequent design and implementation of the program. Additionally, the relationships between the classes in the model correspond to the actual class relationships in the programming implementation. Through this unified design approach, the system becomes more coherent, complete, and clear, providing effective guidance for software development. As shown in Figure 13 UML of User Interface, the logical relationships between the three game interfaces and the game panel are clearly displayed.

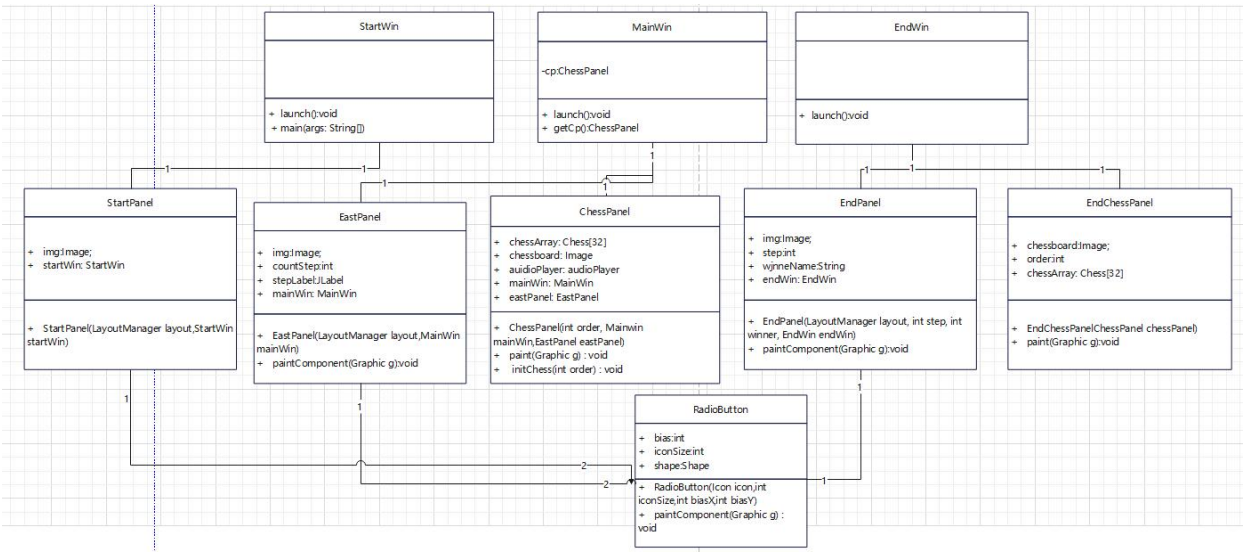


Figure 13 UML of User Interface

To better demonstrate the design effect of the game interface, the following three interface design diagrams are used to illustrate the design at three different stages. Figure 14 Start Interface shows the design of the start interface, Figure 15 Game Interface shows the design of the game interface, and Figure 17 End Interface shows the design of the game over interface.

3.3.1 The details of Start Interface

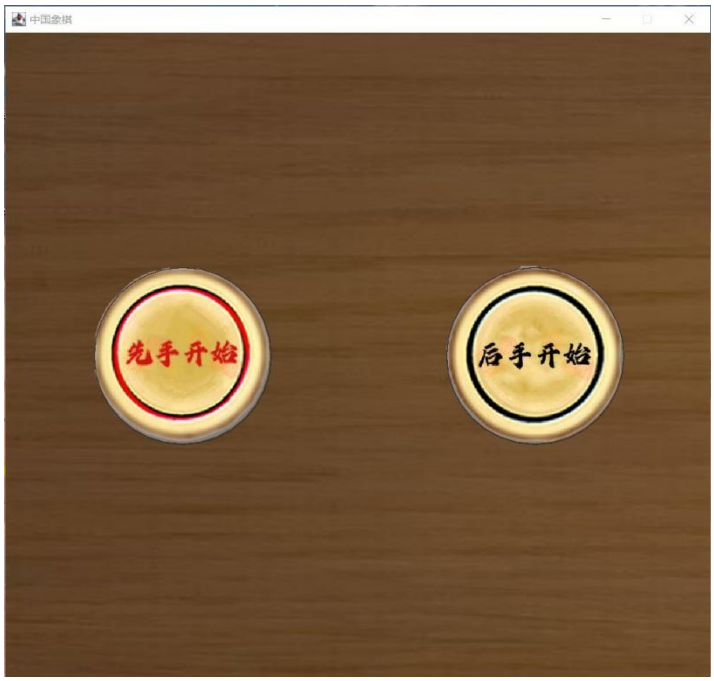


Figure 14 Start Interface

The design of the start interface is concise and straightforward, with no excessive elements. It adopts two Chinese chess buttons combined with a background panel to achieve the function of players choosing the opening move. The first button is used for players to open as red, and the second button is used for players to open as black. After clicking the button, an int type order parameter will be passed in, and the game interface will initialize the chess pieces and chessboard according to this order parameter.

3.3.2 The details of Game Interface

The game interface is the core interface of the user interface module. It includes two main panels, the chessboard panel and the east functional panel. The chessboard panel is used for players to play against the AI, while the functional panel is used for functions such as undo, restart, and displaying the current game status.

(1) Chess Class Design

Before designing the chessboard panel, the design of the chess piece class should be discussed. In Chinese chess, the chess piece is the core of the game. To represent various types of chess pieces, it is necessary to design the Chess superclass, which should include attributes such as the piece's grid coordinates p, the corresponding pixel coordinates x, y of the piece on the chessboard image, the piece name, and the unique identifier of the piece. The use of grid

coordinates p facilitates the maintenance of the piece's position. The conversion between grid coordinates and chessboard pixel coordinates can be achieved through a linear representation by simply multiplying a coefficient and adjusting the offset. For example, to convert a coordinate point to pixel coordinates, the following formula can be used:

$$x = BREAK * (p.x - 1) - MARGIN;$$
$$y = BREAK * (p.y - 1) - MARGIN; \quad (3)$$

Where x and y are pixel coordinates in px, BREAK represents the distance parameter between coordinate points, and MARGIN is the offset of the image on the edge. The pointToXY() method is implemented in the code for coordinate conversion.

By obtaining these coordinate values, the initial game board will assign them based on the initial position of different types of chess pieces. The name attribute is used to represent the type of chess piece, with a total of 14 types in Chinese including ″車″, ″车″, ″马″, ″馬″, ″相″, ″象″, ″仕″, ″士″, ″将″, ″帅″, ″炮″, ″黑炮″, ″卒″, and ″兵″. These pieces are stored through the array, and the corresponding array numbers with the initial position of the chessboard are shown in the below Table 3 Array numbers and initial position.

Table 3 Array numbers and initial position

Name	id	p.x	p.y
帅(King)	0	5	10
士(Mandarins)	1	4	10
士(Mandarins)	2	6	10
相(Bishop)	3	3	10
相(Bishop)	4	7	10
车(Chariot)	5	1	10
车(Chariot)	6	10	10
马(Horse)	7	2	10
马(Horse)	8	8	10

炮(Cannons)	9	2	8
炮(Cannons)	10	8	8
兵(Soldier)	11	1	7
兵(Soldier)	12	3	7
兵(Soldier)	13	5	7
兵(Soldier)	14	7	7
兵(Soldier)	15	9	7
将(General)	16	5	1
仕(Guards)	17	4	1
仕(Guards)	18	6	1
象(Elephant)	19	3	1
象(Elephant)	20	7	1
車(Castle)	21	1	1
車(Castle)	22	9	1
馬(Knight)	23	2	1
馬(Knight)	24	8	1
黑炮(BlackCannons)	25	2	3
黑炮(BlackCannons)	26	8	3
卒(Pawns)	27	1	4
卒(Pawns)	28	3	4
卒(Pawns)	29	5	4
卒(Pawns)	30	7	4
卒(Pawns)	31	9	4

This attribute is mainly used as a reference for the movement rules of different types of chess pieces. In addition, the name attribute is also used to read the object chess image.

The SIZE attribute is used to adjust the size of the chess pieces, which are also loaded as images. Adjusting the size can facilitate displaying changes in the chess pieces, such as prompting by flashing. RES is the image resource bound to each chess piece, and a unique image is bound to each piece when it is created for display on the game board. The name represents the name of the piece.

The isDead attribute is a boolean attribute. When it is true, it indicates that the piece has been eaten by the opponent. When it is false, it indicates that the piece is still alive and should be displayed on the game board. By modifying this attribute, the operation of eating a piece can be conveniently displayed during the game.

In the class, setter and getter methods should be designed for the above attributes to provide interfaces for modifying and accessing the attributes. In addition, there are some interface methods, such as drawFeedback and drawPieces, which are used to draw the chess pieces and the special effects of the interaction between the chess pieces.For subclasses of the Chess class, it is only necessary to override the name, id, and other attributes of the Chess class. This makes it easier to maintain the code in the future.

Figure 14 Game Interface shows the layout of the chess pieces after initialization in the chess panel. We can see that there are 32 chess pieces of various types on the chessboard. If the chess pieces are directly initialized, it will cause a high code coupling degree, with the chess panel class depending on each chess piece class. Therefore, in the early design stage, I considered using a simple chess piece factory class for the initialization of the chess pieces.The ChessFactory class has a static createChess() method, which takes in parameters such as the name of the chess piece, the order in which the piece is placed, the string s, integer i, and id. Based on the name of the chess piece, a switch case is used to create an instance of the specific Chess subclass with the given properties. If the name does not match any of the predefined cases, null is returned.

Figure 16 UML Diagram of Chess shows the UML diagram of the chess-related classes, where the ChessRules class is responsible for the movement rules of the chess pieces, which will be discussed in section 3.4.

Figure 15 Game Interface

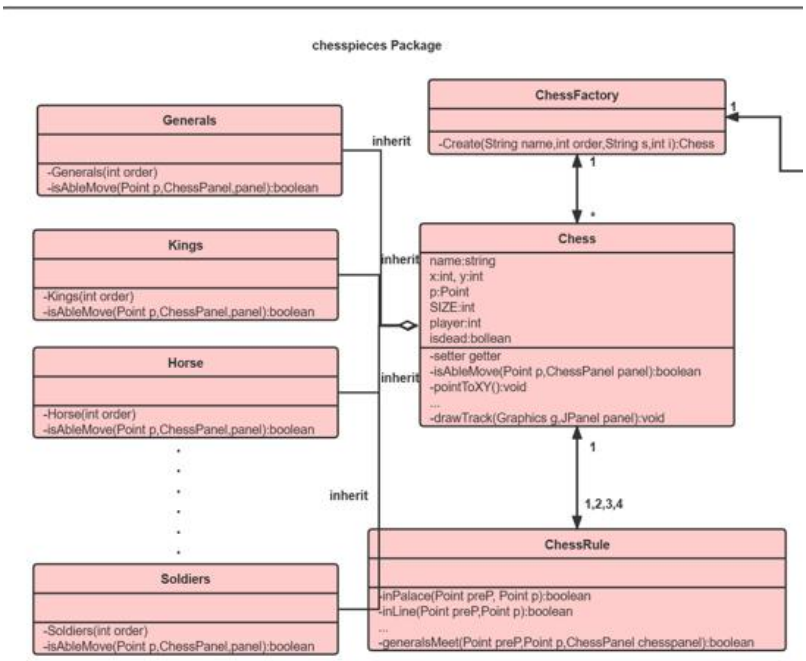


Figure 16 UML Diagram of Chess

(2) EastPanel Design

EastPanel is a functional panel located on the right side of Figure 15 Game Interface. This class contains two circular chess buttons for "undo" and "restart", as well as two JLabel tags for "Step 0" and "Red or Black Move". These four controls provide auxiliary functions other than chess playing, such as prompting the player to move when it's their turn, or warning the

player not to give a check when they are in check. This design makes the interface more user-friendly.

(3) AudioPlayer Design

Using auditory interactions in addition to visual interactions in game user interfaces can enhance the gaming experience for players. Audio effects can increase the realism of the game by providing players with a more immersive experience. For example, when a chess piece is captured, playing a sound effect can make the experience more tangible for players, thus making the game feel more real.

Moreover, audio effects can help players better understand the game state. For instance, playing a check sound effect when a player's king is threatened can help players more easily detect check positions, thereby making it easier for them to understand the game state.

To implement audio effects in the game, an AudioPlayer class can be designed using Java's built-in audio interface to transmit audio data streams. The audioPlayer(str) method can be used to play sound effects by passing in a string that represents the type of sound effect. Additionally, creating a new playAudio thread can synchronize the movement of chess pieces with the playing of sound effects.

3.3.3 The details of End Interface

The End interface consists of two key panels: EndPanel and EndChessPanel, as shown in Figure 16 End Interface.Through the comparison of the Game Interface and End Interface, it can be observed that most of the interface controls on the game main interface are completely identical, with only a few specific module features for different game modes showing some degree of difference. Considering the extensibility and game design difficulty, such a unified design across multiple interfaces is undoubtedly the most suitable choice. On the one hand, it aims to provide players with a better and more unified visual experience for the entire game main interface, while on the other hand, the coordinated and unified interface can to some extent reduce the difficulty of game development and design, greatly improving the reuse rate of the code.The EndPanel is similar to the east functional panel in the Game Interface, displaying the winning player, counting the number of steps, and providing a restart button to start the game again. On the other hand, the EndChessPanel obtains the chessboard information from the ChessPanel instance and displays the final chessboard state, making it easier for players to review the game and its outcome.

Figure 17 End Interface

3.4 The details of Game Logic module

The game logic module consists of four parts, including Legal Movement Logic, Offense and Defense Switching Logic, Victory and Defeat Judgement Logic, and Undo Logic. These logic modules play a crucial role in the normal operation of the game and ensure the correctness of the game rules.

3.4.1 The details of legal movement logic

The Legal Movement Logic module is responsible for generating valid moves for players or AI, ensuring that chess pieces are moved in accordance with the rules of Chinese chess. This module includes the design of the Chess Rules and actions of moving chess pieces.

(1)The Design of Chess Rules

Due to the complexity of Chinese chess rules, it is proposed to create a dedicated class to represent the move rules, named ChessRule. This class will record the potential move rules for different types of chess pieces, such as "obstructing the horse's leg" and "moving in a straight line." These methods are generally not exclusive to a single chess piece but are shared by all pieces. For example, the straight-line movement rule is applicable to rook, knight, general, and soldier pieces.

When moving a specific type of chess piece, multiple rules from the ChessRule class will be

considered simultaneously as the final move rule for the piece. For example, when moving the rook and cannon pieces, the inLine() method must be used to ensure that the target point and the piece are on the same straight line. Additionally, the countBlock() method is utilized to calculate whether there are any obstacles between the chess piece and the target point. If the returned value is 0, it indicates no obstacles, and the rook can move. However, for the cannon, the countBlock() method should return a value of 1, as the cannon can attack with a piece in between.

Whenever the user makes a move, the computer will check whether the move complies with the rules. Invalid moves will trigger a prompt message. At the same time, during the computer's intelligent analysis, it will also refer to valid move rules to find the optimal move. Therefore, a function called isAbleMove(Point p) is designed as one of the methods in the Chess class. This method will call multiple methods from the ChessRule class to determine whether the move complies with the rules.

This function retrieves the target point, searches for the move rules of the chess piece, and makes a judgment. If it complies with the rules, it returns true; if it does not, it returns false . It is important to emphasize that the movement rules for all types of chess pieces in this class are defined using mathematical calculation methods and are based on the corresponding chessboard model as shown in the Figure 15 Game Interface.

(4) The move action

The design of the movement actions in Chinese chess should take into account two aspects: obtaining the player's movement position and implementing the movement effects. Obtaining the player's movement position can be achieved through logical judgments based on the location point clicked by the user's mouse. The specific operations are shown in Table 3 User Valid Click Event and Corresponding Effect.

Table 4 User Valid Click Event and Corresponding Effect

Numbers of click	Place	Operation	Display Effect
1	On Chess	Save the Chess	The chess piece becomes larger and a halo is displayed.

2	Valid Target Point	Change the Point of the Chess	The chess piece move to the target position and a sound effect is played to indicate the movement.
2	Another Chess in same side	Save the new Chess	The appearance of the previous chess piece is restored, while the new chess piece becomes larger and displays a halo.
2	Another Chess in different side	Change the chess point to the new point and remove the original chess from chessboard	Chess piece moves to the target point, it captures an opponent's chess piece, displays the movement path, and plays a sound effect indicating the capture.

From the table ,When the player clicks a chess piece for the first time, the chess piece is stored and a click effect is displayed. When the player clicks a chess piece for the second time, three scenarios are distinguished: clicking on a chess piece in the same camp/faction to select a new chess piece, clicking on a blank area to move a chess piece, or clicking on an opponent's chess piece to capture it. Before performing the latter two operations, a compliance check for the movement must be conducted, which involves calling the isAbleMove(P) method mentioned in the chess rules.

3.4.2 The details of Offense and Defense Switching Logic

After designing the move function, it is important to consider that a side cannot move continuously.Chinese chess is played in turns, meaning that the red and black sides take turns to move their chess pieces. To implement this feature, a method for switching the control of the chessboard between the attacking and defending sides must be designed, allowing for the switching of control between the current player and the computer AI. Specifically, a boolean type isPlayer property is added to determine whether it is the player's turn. After the player finishes their move, the isPlayer property is set to false. After the AI completes its move, the property is set to true. In addition, this method can also be designed to include other features to enhance the player's gaming experience, such as incrementing the game steps through this method and switching between "Red's turn" and "Black's turn" prompts.

3.4.3 The details of victory and defeat judgement logic

In addition to implementing the move and turn switching functions, the logic for judging the winner of the game needs to be designed. There are two main ways to determine the outcome of the game: one is to win by capturing the opponent's general, and the other is to win by putting the opponent's general in a "checkmate" state, which means that no matter what defensive strategy the opponent uses in the next move, their general will be captured.These two scenarios are illustrated in Figure 18 Two different logic to judge whether the game is end.

Figure 18 Two different method to judge whether the game is end

The first method is relatively simple to implement. After capturing a chess piece, it is checked whether the piece is the general by its name "帅" or "将" to end the game. However, in this case, it is possible for a player to overlook the situation and be checkmated, meaning that they lose the game even if they were not actually in a checkmate position. In real-life chess games, this situation can be resolved through communication between the players, but in the case of a chess program, it can lead to a decrease in user experience, which contradicts the original intention of providing a platform for players to practice chess. Therefore, the program ultimately decides to use the second method to determine the winning logic. In this case, two new methods will be designed in the chessPanel class to determine the checkmate situation. The first method is used to determine whether a chess piece is in check. The implementation is to traverse the positions of all the pieces on one side and use the isAbleMove(p) method, where p is the position of the general, to determine whether it can be attacked by any of the opponent's chess pieces. This method can detect whether there is a checkmate situation on both sides. If there is, the corresponding sound effect is played, and after the move, the player is checked to see if they are in check to prevent them from putting themselves in a checkmate position. The second method is used to determine whether a checkmate is possible. This

method traverses all the positions of the pieces and attempts to move them to all possible positions. For each position, the first method is called to determine whether it is in check, which can determine whether there is a position that can resolve the checkmate state. If there is, then there is no checkmate, otherwise there is.

3.4.4 The details of undo logic

In Chinese chess game, the design of the undo function is to facilitate players to correct and undo errors during the chess game. Since each chess piece in Chinese chess has a different way of moving, once a chess piece is moved, it cannot be returned to its original position, which means that each move is irreversible. Therefore, the undo function can help players to correct or undo the previous chess piece movement in case of errors or dissatisfaction, making it easier for players to find the best moves. In addition, this function can also help players avoid some negligence or accidental mistakes, thus improving their chess skills.

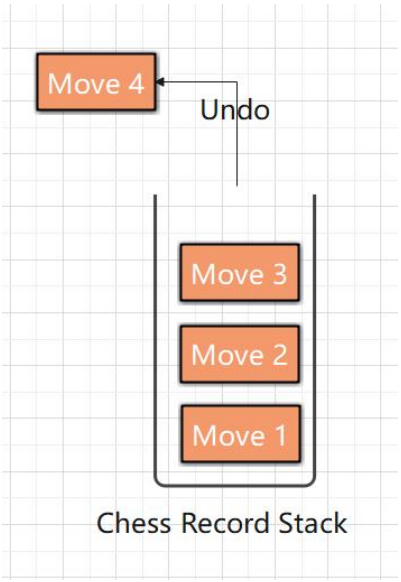


Figure 19 Use Stack to Undo the Chess

In the design of the undo feature, a stack can be used as the data structure to store the records of chess moves. As shown in Figure 19 Use Stack to Undo the Chess. The "last in, first out" characteristic of a stack is particularly suitable for the needs of the undo operation, where the last chess move can be undone by popping an element from the stack. In addition, using a stack data structure has the following advantages: low space complexity, as the stack only needs to store the records of each move, so it occupies relatively little space; low time complexity, as the stack operations (push/pop) are both constant time complexity, making implementation relatively simple and efficient; and convenience in implementation, as Java

has a built-in Stack class which is a stack data structure and is easy to use.

When implementing the undo function in the existing program, the following aspects need to be considered:Firstly, the undo function should only be provided during the player-controlled chess phase and cannot be used when the AI is moving.Secondly, the undo operation should remove two pieces at once, representing the player and AI moves respectively. The player should not be allowed to undo AI moves.Thirdly, the user experience of the undo function should be improved. For example, when the undo function is successfully used, the step count displayed in the eastPanel panel should decrease by one. In addition, using sound and visual effects to inform the user of a successful undo operation can enhance the interactive experience.

3.5 The details of AI module

The game algorithm is one of the most critical components of the man-machine battle. In the design of this game, the system has adopted the well-known Minimax algorithm and the Alpha-Beta Pruning algorithm in artificial intelligence algorithms.

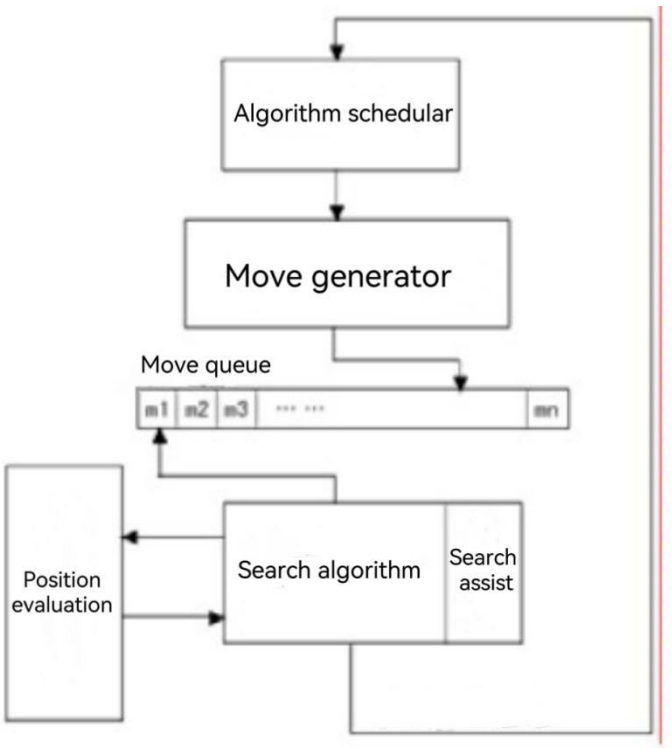


Figure 20 Module Structure

This module's system flow can be reflected by the structural diagram Figure 20 Module Structure.The algorithm scheduler first determines the current game stage and selects the appropriate search algorithm depth and evaluation function. Then, the move generator is used to generate all possible moves for the current game state and adds them to the move queue.

The search algorithm recursively searches all moves to find the best move. During the search process, the board evaluation function is used to evaluate the quality of the game state to quickly determine which moves are better. In addition, search assist such as the alpha-beta pruning optimization algorithm used in this article are needed to improve the efficiency of the algorithm. Finally, the best move generated is outputted, allowing the system to make the next chess move.

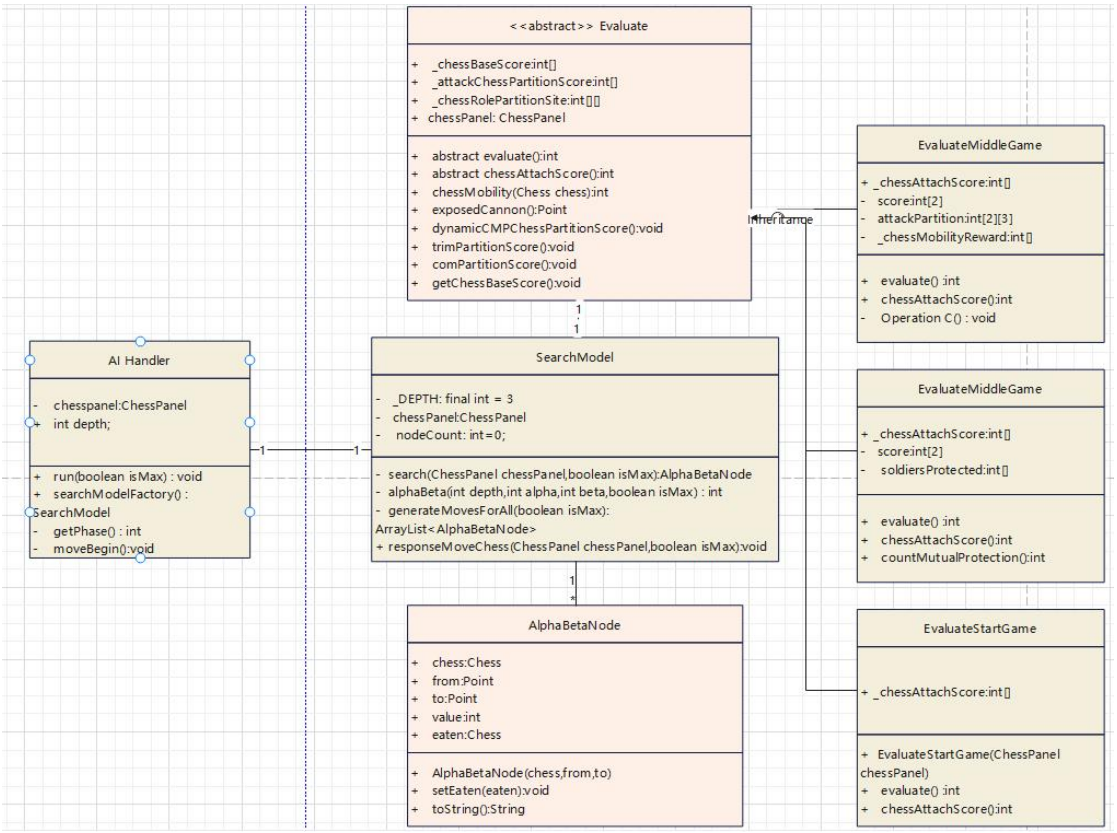


Figure 21 UML diagram of AI module

The UML diagram shown in Figure 21 UML diagram of AI module, provides a detailed design of the various classes and their dependencies in the system. The AlphaBeta class is responsible for representing all the nodes in the game tree and stores information on the survival and starting positions of each chess piece. The Evaluate abstract class defines the methods for computing the evaluation function value, which are implemented in its subclasses to calculate the evaluation function value for a given node. The AI Handler class selects the appropriate evaluation function class based on the current game phase. The SearchModel class generates all valid moves and stores them in AlphaBeta nodes. It applies the minimax algorithm with alpha-beta pruning to traverse the game tree. Once the search reaches the best node, the program retrieves its starting and capture positions to make the optimal move for the AI.

3.5.1 The details of move generator

In Chinese Chess, the move generator is a critical algorithm that generates all legal moves that can be made by either black or red player in the current game state. This is essential for the subsequent search algorithms to traverse and search the game tree. In AI gameplay, search algorithms depend on move generators to obtain all possible moves and search for the optimal solution. The use of move generators ensures the completeness and correctness of the game tree and improves the efficiency and accuracy of the search algorithms.In the specific implementation, a method is implemented in the SearchModel class, which accepts a Boolean parameter isMax to specify whether to generate moves for maximum or minimum value search. Based on the value of isMax, this method selectively generates all valid moves for either the red or black side. For each chess piece, the method can traverse and determine all the points it can reach by using the pointByRule method in the ChessRule class, and create a new AlphaBetaNode object with the original position as well as the new position as parameters. This object represents a child node in the game tree. Finally, the method returns an ArrayList list containing all child nodes.The specific implementation can refer to Figure 21 Move GeneratorFigure 22 Move Generator.

```
private ArrayList<AlphaBetaNode> generateMovesForAll(boolean isMax) {//生成当前局面黑方或红方的所有着法
    ArrayList<AlphaBetaNode> moves = new ArrayList<>();
    for (Chess chess : chessPanel.getChessArray()) {
        if (isMax && chess.getPlayer() == 1) continue;//最大值搜索，则生成红子所有着法
        if (!isMax && chess.getPlayer() == 0) continue;//最小值搜索，生成黑子所有着法
        for (Point nxn : ChessRule.pointByRule(chess,chessPanel,isMax)) {
            moves.add(new AlphaBetaNode(chess, chess.getP(), nxn));//每个着法为博弈树的一个子节点
        }
    }
    return moves;
}
```

Figure 22 Move Generator

3.5.2 The details of algorithmic search engine

In this paper, the alpha-beta search algorithm is used as the AI algorithm for the game of Chinese Chess. As such, a dedicated SearchModel class was designed to implement the main functionalities of the search algorithm, with the external interfaces called by the algorithm modified to fit the specific application context of Chinese Chess. The most important attribute

of this class is a boolean isMax, which specifies whether the search is for maximum or minimum values. Another critical attribute is the int depth, which represents the search depth and directly reflects the computational complexity and quality of the search results. This variable is dynamically adjusted by the AIHandler class based on the current state of the game. In terms of method implementation, the search(ChessPanel chessPanel, boolean isMax) function is designed to carry out the core search process of the algorithm. This function searches all possible moves and returns the best node by evaluating the alpha-beta values of each node.The specific implementation can refer to Figure 23 Core method in SearchModel.

```
private AlphaBetaNode search(ChessPanel chessPanel,boolean isMax) {//对每一个根节点进行alpha-beta搜索，返回最佳
    this.chessPanel = chessPanel;
    long startTime = System.currentTimeMillis();
    AlphaBetaNode best = null;
    //生成所有着法
    ArrayList<AlphaBetaNode> moves = generateMovesForAll(isMax);
    for (AlphaBetaNode node : moves) {
        //移动
        Chess eaten=chessPanel.nextMove(node.chess, node.to);
        node.setEaten(eaten);
        //计算根节点评估函数
        node.value = alphaBeta(DEPTH, Integer.MIN_VALUE, Integer.MAX_VALUE, !isMax);
        //最佳根节点评估函数
        if(isMax) {
            if (best == null || node.value >= best.value)
                best = node;
        }else{
            if (best == null || node.value <= best.value)
                best = node;
        }
        //返回
        chessPanel.backMove(node,eaten);
    }
    long finishTime = System.currentTimeMillis();
    System.out.println(best.toString());
    System.out.println("耗时："+(finishTime - startTime)+"毫秒\t 分数:"+best.value+"\t叶子节点："+nodeCount);
    return best;
}
```

Figure 23 Core method in SearchModel

3.5.3 The details of board evaluation

According to section 2.4.2 in the Background, a good design strategy for the evaluation function is provided. In practice, I designed different evaluation function classes for the opening, middle game, and endgame phases of the game to handle the different values of chess pieces in different stages. Overall, the design of the evaluation function includes the following scores: (1) the basic score of the chess pieces, (2) the positional score of the chess pieces, (3) the attack score and defense score of the chess pieces in different regions, (4) the

flexibility score of the chess pieces, and (5) the score for special tactics. Evaluating the game based on these scores can make the algorithm more reliable.

In the implementation, the key methods are the evaluate() and chessAttachScore() methods. The former computes the overall score by summing up the scores of each component, while the latter calculates the score of a single chess piece based on its role and location. The evaluate() method first calculates the basic scores of the chess pieces, then computes the scores for each region by calling the compPartitionScore() method, which calculates the score for each region based on the surviving chess pieces. It then computes the score for each player based on the attack and defense scores of their chess pieces in each region. Other factors such as mobility, special tactics, and weaknesses are also considered in the evaluation. Finally, the difference between the scores of the two players is returned as the overall score.

3.5.4 The details of Engine Scheduler

The main purpose of the algorithm scheduling class is to select appropriate evaluation functions and search depths based on the different stages of the current game situation, thereby improving the efficiency and accuracy of the AI engine. In a game of Chinese chess, different stages of the game may require different evaluation functions to assess the value and position of the chess pieces, in order to better guide the AI engine in making the next move.

For example, in the opening stage, most moves have similarities, so a generic position function can be used for the initial layout. In the middle game stage, however, the focus is more on the overall pattern of the chessboard and attacking strategies, thus different evaluation functions may be needed. Additionally, in the endgame stage, it may be necessary to increase the search depth to cope with the complex variations.

In implementation, the key functionality of the scheduling class is to select the correct evaluation function and search depth based on the game state (opening, middle game or endgame), in order to enable the computer opponent to make better moves. This class should include a factory method for creating a SearchModel object that utilizes the evaluation function to implement a specific search algorithm. The scheduling class also includes some special heuristic search strategies, such as the rising value of the pawn with the decrease of attack power, the increasing value of the horse when the number of chess pieces is small, and the decreasing value of the cannon, etc. Additionally, the class includes some auxiliary functions, such as the getPhase() function, which determines the current game state to select the appropriate evaluation function and search depth. The moveBegin() function also contains

some logic for adjusting the value of chess pieces, to better adapt to the computer opponent's decision making in different stages of the game.The following Table 4 Determination of the game phase. shows the decision logic of the getPhase() method for evaluating the current game phase.

Table 5 Determination of the game phase

Game Phase	Judging Criteria
Start (START_GAME)	Total number of chess pieces ≥ 29
Middle (MIDDLE_GAME)	Total number of chess pieces < 29 and (number of pawns/soldiers > 3 and number of cannons/chariots/horses > 6 and total number of chess pieces > 16)
End (END_GAME)	Total number of chess pieces < 29 and (number of pawns/soldiers ⩽ 3 or number of cannons/chariots/horses ⩽ 6)

Chapter 4: Results and Discussion

Upon testing the three modules of the game, it has met the fundamental requirements.
The user interface module performs well, with accurate display of chess piece positions, smooth interface transitions, and seamless user interaction. The game logic module has undergone thorough testing, confirming that various chess pieces' movements adhere to the rules, and both offense-defense transitions and undo functions are operating as expected. The AI module functions effectively, with the algorithm scheduler accurately determining the game stage and selecting appropriate evaluation functions and search depths, yielding optimal moves with a certain level of chess strength.

4.1 Testing methods of AI Module

In this section, I will briefly introduce the testing methods used for the AI module, including unit testing and system testing.More testing cases please look up in Appendix.

4.1.1 Unit testing

The most critical part of the AI module to be tested is the evaluation function. As mentioned in Section 2.4 of the report, a piece's evaluation function ultimately depends on multiple factors, including its position score, flexibility score, and special strategy score. To ensure that these scores are effective for the AI's decision-making, I conducted unit testing using Junit4. Junit4 is an open-source Java testing framework that simplifies the testing process and provides rich assertion methods and other features to facilitate the writing and execution of test cases. In the unit testing, I separated the different scores of the evaluation function and tested their effectiveness one by one. The evaluation function class and methods that underwent unit testing are shown in the table below:

Table 6 Unit testing

Class	Method	Description
EvaluateTest	testChessMobility():void	Test the flexibility of a certain piece at a position on the board

	testExposedCannon():void	Test whether there is a exposed cannon on the board
	testRestChariot():void	Test whether there is a rest chariot on the board
	testBottomCannon():void	Test whether there is a bottom cannon on the board
EvaluateMiddleGameTest	testEvaluate():void	Test the evaluation function in a certain situation in the middle game.
	testPartitionScore():void	Test the partition scores in the middle game of chess
	testAttachScore():void	Test the game position scores in the middle game of chess
EvaluateEndGameTest	testMutualProtect():void	Test the number of protection between the soldiers

4.1.2 System testing

System testing involved testing the overall performance of the AI module during gameplay. Here, I present the methods and results of the AI module's overall performance testing. I tested the AI module's performance by playing multiple rounds against a human player and recording the response time, leaf node count, and optimal evaluation function for each move. Response time is a critical indicator of the AI module's speed of response. I added up the time taken for each operation and calculated the average response time as a performance parameter. Leaf node count and response time are positively correlated. In general, the more leaf nodes, the deeper the AI thinks about the situation, and the better the decision made. However, response time also increases. The optimal evaluation function represents the situation of both sides in a particular move. If the red side is dominant, the score increases. Conversely, if the black side is dominant, the score decreases. I obtained the specific parameters by printing the relevant parameters in the console. The Table 7 System testing shows the changes in these three critical parameters during the opening, middle, and endgame of a game:

Table 7 System testing

Game stage	Time(ms)	Leaf node	Evaluation value
Opening	4480	32032	50

	2182	16566	75
	3068	23475	110
	2417	18372	105
	2788	20763	130
	2983	22336	150
	4274	31983	220
	4706	36345	245
	3872	29347	240
	2969	22398	313
MiddleGame	9746	19434	295
	11146	20994	362
	10580	19653	389
	6886	12853	449
	10141	17121	338
	11547	20858	481
	13948	28469	430
	10555	19108	366
	10808	21115	-243
	16397	28404	-159
	14144	27658	281
	12171	24096	225
	9809	18948	436
	7659	15387	631
	6543	13635	1134
	5125	11675	1436
Endgame	10842	96194	1333
	10616	93100	1288
	6976	63127	1288
	14108	123607	1218
	13966	202186	1165
	11637	101913	10991

After conducting ten rounds of testing, I recorded the average response time, average leaf node count of the AI module during the opening, middle, and endgame. The specific data are

shown in the Table 8 Average response time and leaf node below:

Table 8 Average response time and leaf node

Game Stage	Average response time	Average leaf node
Opening	3298ms	25291
Middle game	6114ms	16834
Endgame	10854ms	110505

4.2 Result and Discussion

These test results demonstrate that some drawbacks exist. For instance, the AI might experience prolonged delays (exceeding 10 seconds) during the middle and endgame stages, with an average response time of around 7.5 seconds, adversely affecting the gameplay experience for speed-focused players. Reducing search depth would lead to a decline in algorithm strength, primarily due to an incomplete understanding of the chess algorithm during the initial design phase, emphasizing the enhancement of evaluation functions while neglecting pruning algorithms and dedicated storage data structures for chess games.

Moreover the AI may occasionally choose a suboptimal move when facing checkmate, rendering the sole reliance on "checkmate" as the termination criterion insufficient. To ensure precise control over the game's conclusion, I incorporated capturing the opponent's king as a winning condition in the player's victory logic, utilizing a dual winning logic approach.

Despite these shortcomings, the overall project is still successful. To address these issues, future research can involve improvements such as employing speculative move generation, null move pruning, and history heuristic algorithms to enhance pruning efficiency. Additionally, bitboards can be used as a data structure for storing chess pieces, processing game rule-related logic through bitwise operations without iterating through the chess array, significantly boosting algorithmic computation speed.

Chapter 5: Conclusion and Further Work

In conclusion, this project successfully developed a Chinese Chess game using the Java language and incorporating artificial intelligence algorithms, which met the fundamental requirements. The user interface, game logic, and AI modules performed well, providing a smooth gameplay experience. However, some limitations were identified, such as the AI's occasional suboptimal moves when facing checkmate, and the prolonged response time during the middle and endgame stages.

Despite these limitations, the overall project is deemed successful, and future work can focus on addressing the identified issues. For example, enhancing pruning efficiency by employing speculative move generation, null move pruning, and history heuristic algorithms, as well as adopting bitboards as a data structure for storing chess pieces to significantly boost algorithmic computation speed.

The study of Chinese Chess, its rules, gameplay, and the application of artificial intelligence algorithms, combined with the development process and methods of Java projects, has expanded the understanding of traditional Chinese culture and the practical application of emerging technologies. The project demonstrates the potential for further improvements and optimization in subsequent research, ultimately contributing to the advancement of Chinese Chess games and the promotion of traditional Chinese culture worldwide.

In terms of future work, the following areas can be explored:

Enhancing the AI's performance by incorporating more advanced machine learning techniques or deep learning algorithms, potentially improving the game's overall challenge and appeal to a wider audience.

Expanding the game's functionality to support online multiplayer capabilities, allowing players to compete against each other remotely and fostering a more engaging and interactive gameplay experience.

Implementing adaptive difficulty levels based on player skill, ensuring a more balanced and enjoyable experience for players of all skill levels.

Investigating the potential for developing mobile or web-based versions of the game to increase accessibility and reach a broader audience.

In summary, this project serves as a solid foundation for future research and development in

the field of Chinese Chess games, with ample opportunities for improvement and expansion, ultimately contributing to the enrichment of traditional Chinese culture in the digital age.

References

[1] Ma, Z. F. (2003). Introduction to Chinese Chess. (M). JinDun Press.

[2] Li, Z., & Pang, Y. (2011). Java programming and project practice. Beijing: Publishing House of Electronics Industry.

[3] Xie, G. (2008). Study on data structure design and search algorithm for Chinese chess machine game (Doctoral dissertation). Xi'an University of Technology, China.

[4] Liu, H. Y., Wu, I. C., Liao, T. F., et al. (2013). Software framework for generic game development in CGDG. In Proceedings of the Computer Graphics, Imaging and Visualization (pp. 219-229). Springer Berlin Heidelberg.

[5] Gao, W., Guo, J., & Zhang, H. (2007). Design and Implementation of Chinese Chess Game Based on Java Technology. Dalian Nationalities University Journal, 9(5), 17-19.

[6] Tarongoy, M.A. and Purgatorio, J.B. (2013). Application of Alpha-beta Searching Algorithm and Intelligent Material Evaluation on the Artificial Intelligence Engine of Dama: The Game. CLOUD, 1(1).

[7] Chen, Y. (2012). Design and Implementation of Human-Computer Chess Games based on Alpha-Beta Search Algorithm. Computer CD Software and Applications, 04, 197-199.

[8] Li, H. (2007). Game tree search algorithm. Journal of Changchun University of Technology, (2), 12-20.

[9] Junghanns, A., & Schaeffer, J. (1997). Search versus Knowledge in Game-Playing Programs Revisited. In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97) (pp. 692-697).

[10] Du, X. R. (2012). Analysis of evaluation function in Chinese Chess. Tianjin Vocational College Joint Journal, 14(02), 87-89.

[11] Song, Y. Q. (2019). Application of UML diagrams in software requirement specification. Shanxi Electronic Technology, (206) (05), 61-63.

[12] Yu, C. (2011). Application of game algorithm in Chinese Chess (Doctoral dissertation). Ocean University of China.

[13] Lu, R. (2008). Application of hybrid game tree algorithm in man-machine game of Chinese Chess (Doctoral dissertation). Dalian Jiaotong University.

[14] Zhang, Z. (2006). Study on quadratic valuation method and its optimization in computer Chinese Chess game (Doctoral dissertation). Northeastern University.

[15] Yuan, T. X., & Fu, Y. Q. (1990). Tree search algorithm in Chinese Chess game program. Journal of Shanghai Jiaotong University, (04), 86-88. DOI: 10.16183/j.cnki.jsjtu.1990.04.012.

[16] Wang, Z. Y., Xu, H. Q., Liu, Q. W., et al. (2020). Development and implementation of computer game Chinese Chess. Computer Knowledge and Technology, 16(07), 63-64+90. DOI: 10.14004/j.cnki.ckt.2020.0761.

Acknowledgement

I am deeply grateful to my undergraduate thesis supervisor for his invaluable guidance and support throughout my project. His expertise and encouragement helped shape my research direction, and his feedback on my report and design diagrams was instrumental in my development.

I am also thankful for the opportunity to work on a project that aligns with my interests and stimulates my curiosity. Being selected as a candidate for this project is a great honor, and I appreciate my supervisor for recognizing my potential.

Appendix

Disclaimer

This report is submitted as part requirement for the undergraduate degree programme at XXX.It is the product of my own labour except where indicated in the text.

Signature:

Date:27-04-2023

Project specification

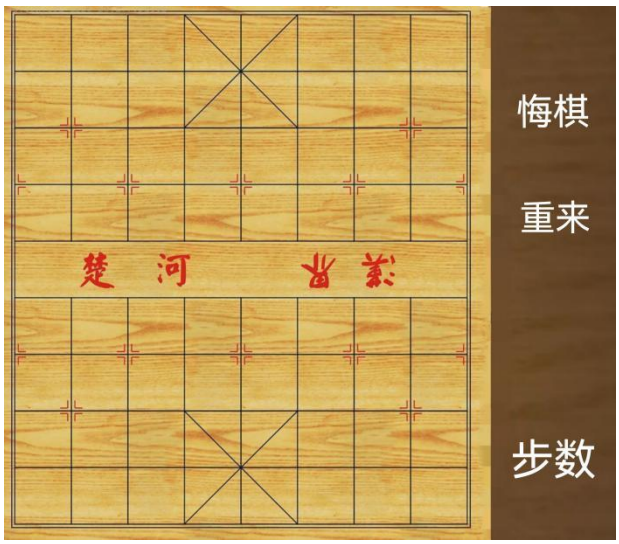
本科毕业设计（论文）任务书

Project Specification Form

Part 1 – Supervisor

论文题目 Project Title	The program of Chinese Chess		
题目分类 Scope	Software Development	Implementation	Software
主要内容 Project description	Chinese Chess is a very popular chess game in China. According to their own understanding of the form of the chess game and their mastery of the chess skills, the chess players mobilize their chess pieces such as chariots and horses, organize their troops and coordinate their operations. The function of this program is to simulate the chessboard and chess pieces on the computer screen, and both parties can play by using the mouse to operate their own chess pieces. The outline design of the program: Chinese Chess consists of three elements: chessboard, pieces and rules. These three elements can be subdivided into four aspects: chessboard, the place where chess can be placed, chess pieces, and chess rules. Chessboard: When drawing a chessboard, the program draws horizontal lines, vertical lines, diagonal lines, etc. A chessboard here is a figure on the computer screen. Placement point: a point on the chessboard where chess pieces should be placed. Chess piece: including the piece's name and affiliated party. Chess rules: each chess pieces can only be placed and fought according to their own rules.		
关键词 Keywords	Chinese Chess, game, programming		
主要任务 Main tasks	**1** To code chessboard and chess pieces management module, responsible for human-machine interface, storing and displaying chessboard and chess pieces.		
	2 To code the playing rules module which stores the playing rules		
	3 To code the winning strategy module which stores the must win or priority strategies.		
	4 To code the placement decision module which calculates the placement point according to the playing rules and the winning strategy		
主要成果 Measurable outcomes	**1** 1. The program code including the above four modules which meet the design goal. And relative technical documentation.		
	2 Complete experiment and test environment with software deployment.		
	3 Interactive game demo of the software		

本科毕业设计（论文）任务书
Project Specification Form
Part 2 - Student

学院 School		**专业 Programme**			
姓 Family name		**名 First Name**			
BUPT 学号 BUPT number		**QM 学号 QM number**		**班级 Class**	
论文题目 Project Title	The program of Chinese Chess				
论文概述 Project outline **Write about 500-800 words** **Please refer to Project Student Handbook section 3.2**	Chinese Chess is a very popular chess game in China. The function of this program is to simulate the chessboard and chess pieces on the computer screen, and both parties can play by using the mouse to operate their own chess pieces. My project involves the use of advanced programming languages(Mainly java) and artificial intelligence algorithms(Mainly decision tree learning) to achieve chess man-machine combat. The user can use the mouse to operate one of them on the computer, and click the chess pieces to move and eat. The other side will be operated by AI. The main tasks are as follows: **1. Draw a main interface of the game.** This interface will only draw using the java swing API. Its bottom container is jFrame, and its upper container is composed of two jPanels. The jPanel in the center is the main part, which is used to draw chessboards and pieces. It is the place for future games. The chessboard consists of ten rows and nine columns. The fifth and sixth rows are separated by the Chu River Han border. There are many kinds of chesspieces, which are respectively red: chariots, horses, guns, Xiang, soldiers, soldiers, and black: chariots, horses, guns, elephants, officials, generals, soldiers. The jPanel on the right is used to place buttons for some auxiliary functions, such as repentance, restart. In addition, it can be used to display some necessary information, such as counting steps. In terms of layout, BorderLayerout is used for the overall layout, and gridLayerout is used for the right function panel. The following is a simple prototype diagram.  **2. Chess's logic of moving, eating, and winning rules.** In this task, I will implement the basic operations of chess. Java will be used to implement the background code. Firstly, the basic mouse click event monitoring is used to				

	obtain the click location. Secondly, judge whether there are chess pieces according to the click position. If there are chess pieces, implement the function of walking and eating. Then, add restrictions on walking chess pieces and eating pieces. Chess pieces need to move according to the rules, such as Ma Zouri and Xiang Zoutian. At the design level, a chess class will be used as the parent class of all chess pieces. In addition, a simple factory design pattern will be used to initialize all pieces. In these ways, reduce code coupling and enhances code reusability. **3. Realize a low-level AI that can only play chess simplely** The machine side will be supported by the decision tree learning algorithm to implement the man-machine combat function. There are many ways to achieve human-computer combat. The most commonly used algorithm is the game tree α - β Pruning, minimax search method. It is very likely that I will use this algorithm to achieve human-computer combat.Specifically, the evaluation algorithm of Chinese chess mainly includes the evaluation of chess pieces, flexibility evaluation and chess piece relationship evaluation. Chess evaluation: according to experience, one car can be valued at 500, one horse 300, one soldier 100, and one general 10,000. Flexibility evaluation: If a chess piece can have multiple moves, the corresponding value is higher. Evaluation of chess pieces relationship: A chess piece will increase its value if it threatens the opponent's chess pieces. The search technology is a game tree, which will recursively generate a search tree containing all the game processes from the root down. Each step will select the best way according to the valuation through the minimax algorithm. In order to increase the efficiency of the algorithm, the tree will be pruned. In this step of the task, a two-layer game tree will be implemented, which may lead to weak chess power but high speed. **4. To improve the computing power of the machine.** The last step will optimize the algorithm of the third task, and optimize its performance. Specifically, the valuation function will be recalculated and the number of game tree layers will be increased. In addition, may use other optimized algorithms. The machine needs to have entry-level chess computing power.
道德规范 Ethics **Please discuss ethical issues with your supervisor using the ethics checklist in Project Handbook Appendix 1.**	Please confirm by checking the box: ☒ I confirm that I have discussed ethical issues with my supervisor.
	Summary of ethical issues: (write “None” if no ethical issues) None

中期目标 **Mid-term target.** **It must be tangible outcomes, E.g. software, hardware or simulation.** **It will be assessed at the mid-term oral.**	Mid-term target is tasks 1、2 and 3 outlined in the project. That is: **1. Draw a beautiful main interface of the game.** **2. Chess's logic of moving, eating, and winning rules.** **3. Realize a low-level AI that can only play chess without training**

Work Plan (Gantt Chart)

Fill in the sub-tasks and insert a letter X in the cells to show the extent of each task

	Nov 1-15	Nov 16-30	Dec 1-15	Dec 16-31	Jan 1-15	Jan 16-31	Feb 1-15	Feb 16-28	Mar 1-15	Mar 16-31	Apr 1-15	Apr 16-30
4. Task 1 [Draw a beautiful main interface of the game.]												
Draw a beautiful Chinese chess board with obvious column lines, row lines and diagonal lines. The junction point is used to place the pieces	X											
Draw chess pieces. The chess pieces are initially placed on the standard position of chess	X											
There needs to be a desktop or a container outside the chessboard to place other required button components	X											
Enhance interactive experience, click feedback, sound effect.	X	X	X	X	X	X	X					
5. Task 2 [Chess's logic of moving, eating, and winning rules.]												
Basic event monitoring		X	X									
Chess moves according to rules and eats		X	X									
Chessboard Rules and Winning Rules			X	X	X							
Optimize code		X	X	X	X							
5. Task 3 [Realize a low-level AI that can only play chess simply]												

Calculate valuation function				X	X	X						
Implement Game Tree				X	X	X						
Enable maximum and minimum value search				X	X	X						
Implement α-β prune					X	X	X					
6. Task 4 [To improve the computing power of the machine.]												
Study other papers and find the method or algorithm to optimize the valuation function (unknown now)								X	X	X	X	
Increase the number of layers of the game tree (this can increase the computing power of the machine, but may cause the algorithm to be slow ,whether to increase will depend on its calculation time)								X	X	X	X	

Early-term progress report

本科毕业设计（论文）初期进度报告
Project Early-term Progress Report

学院 School		**专业 Programme**			
姓 Family nam		**名 First Name**			
BUPT 学号 BUPT number		**QM 学号 QM number**		**班级 Class**	
论文题目 Project Title	The program of Chinese Chess				

已完成工作 Finished work:

1. Summary of literature review：

In the early term of the project, I looked up 4 relative papers. Since the topic of the project is about Chinese Chess, I chose the references in China National Knowledge Infrastructure(CNKI) that included more precise paper than Science Citation Index(SCI). I list them in the Reference List.

The first paper(Reference[1]) title is Design and Implementation of Chinese Chess Game Based on Java. Its content about 4 modules: Relative Technologies Introduction, System Requirement Analysis,System Design and Implement and Test.I mainly studied System Design Module .They are Interface Design, Chinese Chess Class Design,Chinese Chess Move Rule Design, Network Connection Design and Chinese Chess Game Storage Design and Battle Algorithm Design. This paper enlighten me have implemented the game interface and game rule module. In addition, I learned the algorithms of Alpha-Beta search algorithm, game evaluation function, Substitution Table and Hash Table which will used in Man-machine Match module in the future.

The second paper(Reference[2]) title is Design and Implementation of Chinese Chess Man-machine Matches Based on the Alpha-Beta Search Algorithm. This paper mainly elaborates three related algorithms, minimum-maximum search, iterative deepening search and alpha-beta search. The first algorithm is to evaluate the score of each situation in the Chinese Chess game. The situation where A wins is a maximum, and the situation where B wins is a minimum. The disadvantage of this algorithm is that the whole search tree is too large. In the middle game of the Chinese Chess, 2.5 million routes are needed when search a 4 floors tree, which is not feasible.The idea of the iterative deepening is that to start with only one layer, if you is less time than allocated, search two layers, then three layers, and so on until the time is out. Iterative deepening search methods can interrupt search when time runs out and improve search efficiency.The alpha-beta search uses two parameters, alpha and beta. Through the recursive algorithm, each time the path with a smaller evaluation score than the alpha value and the path with a larger evaluation score than the beta value are selected for pruning, thus eliminating a large number of worthless nodes and reducing the time complexity of the algorithm.

Reference[3] is based on alpha-beta pruning, and improve the traditional pruning to enhance algorithm efficiency. It proposes to use history table or iterative deepening to optimize pruning. Because alpha-beta pruning algorithm is strongly related to the order of nodes. The history table can sort all the current nodes from good to bad by previous historical methods.According to the experimental data in Table 1 and Table 2, as the search depth increases, the history table can effectively reduce the number of search nodes, thus effectively reducing the search time. However, the reduction of nodes may also reduce the computing power of the machine. In addition, the iterative deepening algorithm does not improve the search efficiency as much as the history table. But through the program battle experiment, the experiment tests 10 rounds in total, and completes one move

表 1　搜索节点数比较

搜索深度	1	2	3	4	5
α－β	35	284	8 387	108 563	1 500 761
α－β＋历史表	33	155	2 645	12 547	217 936
α－β＋迭代加深	34	265	8 350	103 508	1 398 513

表 2　搜索时间比较　/ms

搜索深度	1	2	3	4	5
α－β	1	8	37	238	10 320
α－β＋历史表	1	7	18	184	1 118
α－β＋迭代加深	1	8	24	208	6 387

Figure 1:Experimental data of Reference[3]

every 5 seconds in each round. The iterative deepening algorithm has won 7 victories and 3 draws, and its average computing power is stronger than the historical table.

Reference[4] proposes an improved Heuristic Search Algorithm based on Transposition Table. When searching the same game tree, using the information previously searched will undoubtedly improve the search efficiency. The Transposition Table is generally used to save the previous search information, it usually uses hash technology to implement. In addition, when using the search algorithm at the beginning of the chess game, the program often seems a little overwhelmed, when the situation of both sides on the chessboard is approximately equal, the search efficiency of the chess program is very low, which wastes a lot of time. Therefore, in the opening stage, if read the database instead of searching, can not only save a lot of game time, but also avoid the system playing "stupid chess".

I have not fully understood the last three papers, I will try my best to deeply understand their process and essence with algorithm program in the next task schedule.

Reference List

[1] 杜帮国. 基于 Java 平台的中国象棋游戏的设计与实现[D].大连理工大学,2013.
[2] 陈业鹏.基于 Alpha-Beta 搜索算法的中国象棋人机对战的设计与实现[J].计算机光盘软件与应用,2012(04):197-199.
[3] 蔡岫.一种中国象棋机器博弈剪枝策略的改进方法[J].国外电子测量技术,2016,35(03):47-49.
[4] 高强,郭琛.哈希技术在中国象棋机器博弈系统中的应用研究[J].科学技术与工程,2008,(17):4869-4872.

2. Summary of work was done：

I have finished Task1 and Task2 and all sub-task in Gantt Chart(figure 2).I will summarize my work in three aspects: project management and program design and implementation.

Project management: I used Leangoo website for project management. Leangoo is a management tool used by current product R&D(Research and development) teams, project managers, HR(Human Resource) managers and other practitioners to collaborate and manage projects. Its concise design revolves around a blank "Kanban", "List" and "Card", enabling users to track, plan or organize the current project.

In addition, I used the agile development pattern to develop Chinese Chess software. This is a development method that takes requirements as the core and adopts iteration and step by step. A large

project will be divided into a number of small projects that are interrelated and can run independently, and tested separately. In the Leangoo "Kanban", the "Kanban" of large project is the product flow chart, the large project divided into multiple small project which called milestone "Kanban", and each milestone is divided into 2 iteration cycles(Sprint).Figure 3 shows the "Kanban" of Sprint1. It contains the small tasks of the first iteration cycle of the project.

	Nov 1-15	Nov 16-30	Dec 1-15	Dec 16-31	Jan 1-15	Jan 16-31	Feb 1-15	Fe 16-
4. Task 1 [Draw a beautiful main interface of the game.]								
Draw a beautiful Chinese chess board with obvious column lines, row lines and diagonal lines. The junction point is used to place the pieces	X							
Draw chess pieces. The chess pieces are initially placed on the standard position of chess	X							
There needs to be a desktop or a container outside the chessboard to place other required button components	X							
Enhance interactive experience, click feedback, sound effect.	X	X	X	X	X	X	X	
5. Task 2 [Chess's logic of moving, eating, and winning rules.]								
Basic event monitoring		X	X					
Chess moves according to rules and eats		X	X					
Chessboard Rules and Winning Rules			X	X	X			
Optimize code		X	X	X	X			

Figure 2: Gantt Chart of Task1 and Task2

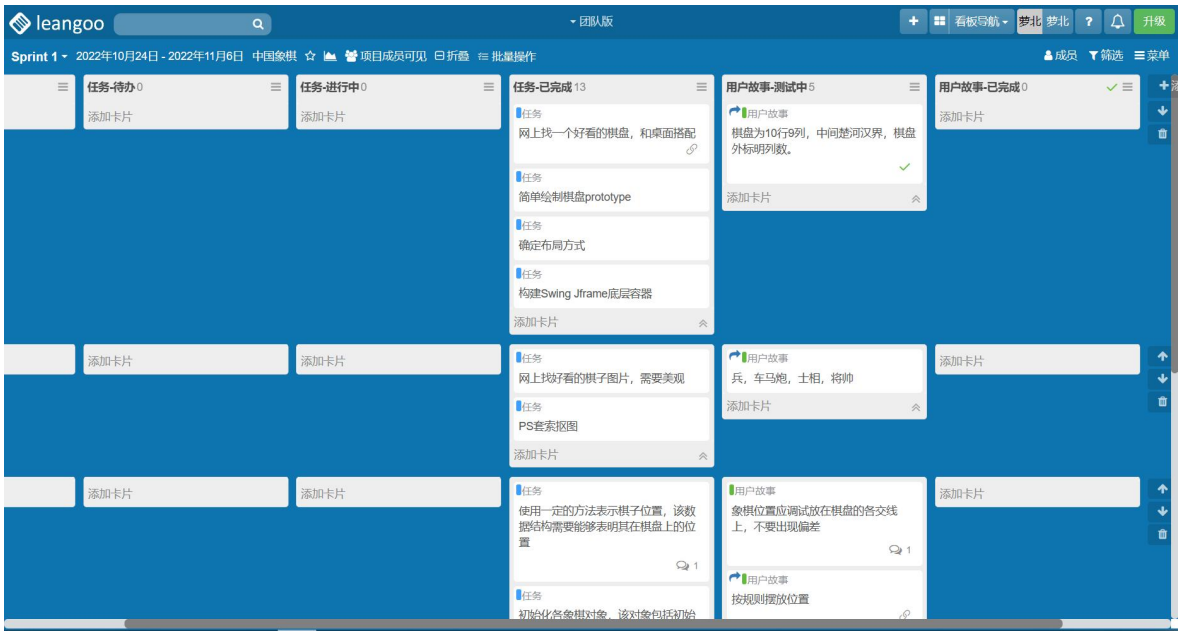


Figure 3: "Kanban" of Sprint1

Specific task details can be seen through the following hyperlink:
Project:
https://www.leangoo.com/kanban/snapshot/get/4577723/b48f7d98c152416dec469b4fd2d7f94e
Milestone 1:
https://www.leangoo.com/kanban/snapshot/get/4577722/b48f7d98c152416dd52fe64fa5d07e66
Milestone 2:

https://www.leangoo.com/kanban/snapshot/get/4577721/b48f7d98c152416da6b65dfc7ab7f165
Spring 1:
https://www.leangoo.com/kanban/snapshot/get/4577719/b48f7d98c152416ddaaab2869364af6e
Spring 2:
https://www.leangoo.com/kanban/snapshot/get/4577718/b48f7d98c152416d36f0894ffdcd5ade
Spring 3:
https://www.leangoo.com/kanban/snapshot/get/4580248/b48f7d98c152416d98a7f2885071c32d
Spring 4:
https://www.leangoo.com/kanban/snapshot/get/4580249/b48f7d98c152416d195c44adfe4d15f7

Design:

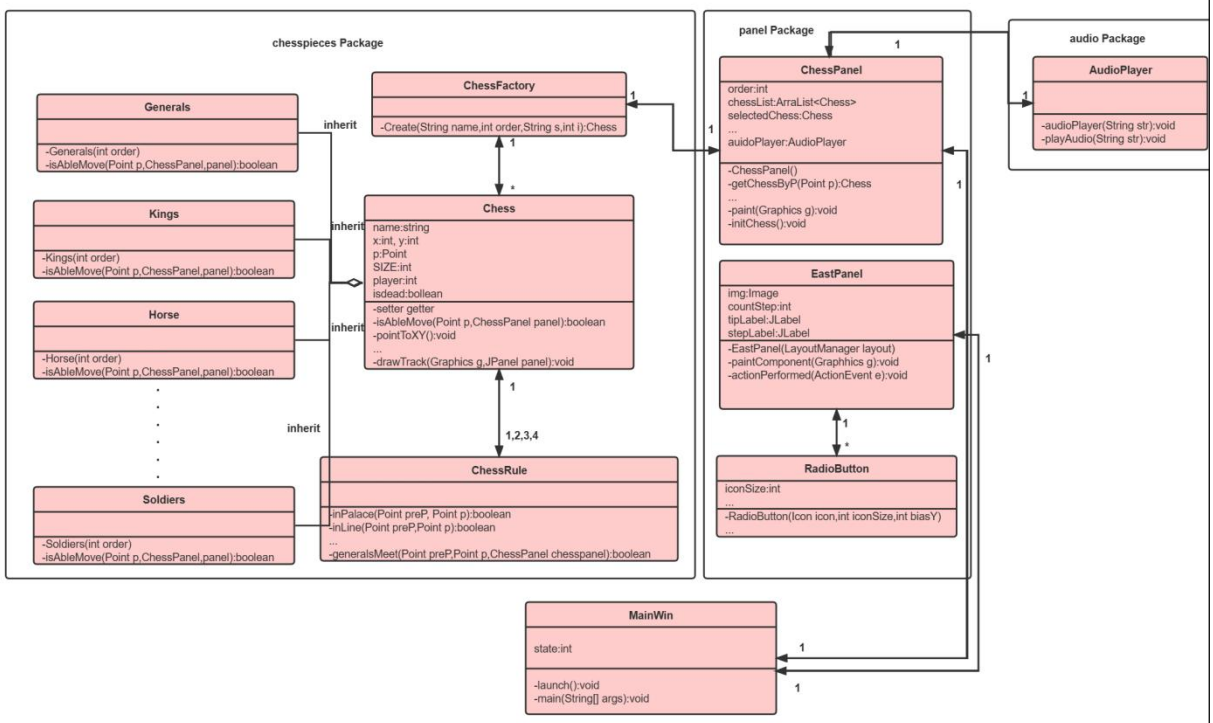


Figure 4: UML Diagram

As shown in the UML diagram, all kinds of chess pieces inherit a common parent class Chess. Each Chess uses one to four rules in the ChessRule class to ensure that the movement of chess pieces conforms to the rules. The simple factory mode is used. The ChessFactory class is used to generate all chess sub objects and will be called in the ChessPanel class. MainWin class is a program entry class, which will create two objects, ChessPanel class and EastPanel class. AudioPlayer class and RadioButton class are complementary to the program, which are used to play audio and create round buttons.

Implementation:

Programming Language: Java

Compiler：IntelliJ IDEA 2021.3.3 (Community Edition)

IDE：openjdk-17

(1) Chessboard

In the main interface of the game, the chessboard occupies the main position, and the chessboard model directly affects the chess pieces and rules, leading the whole chess program. Therefore, the chessboard part was first implemented. The appearance of the chessboard uses pictures on the Internet. Since the relative position of the chessboard and the window was not considered at the beginning of the design, in order to prevent the chessboard and chess pieces from deviating from the position despite changes in the size of the window, the fixed window width is 810 pixels and the height is 765

pixels. In addition, the initial position of chess pieces, the position of falling pieces, the position of buttons and other components are absolute positions.Order attribute is used to controls whether players play chess first or second.

(2) Chess

In order to realize the abstract model of chess pieces, Chess class,subclasses of Chess class and ChessFactory class are implemented. The key method is pointToXY(Point p) and getPFromXY(int x,int y). They are convert the grid coordinates of the chessboard into the pixel coordinates of the picture to ensure that the chess pieces fall on the grid coordinates. Player attribute is used to

(3) The movement of Chess

The key method isAbleMove().It restricts the random movement of chess pieces. This method is an empty method in Chess class. It is inherited by the subclass and calls different sub movement restriction rules in ChessRule to determine the movement rules of the final chess piece.Breaking a rule into multiple sub rules can increase code reusability. For example, the movement of soldiers is limited to a straight line instead of walking sideways, and the movement of cars is also limited to a straight line,and the cannon is also limited to a straight line. They all could call inLine() method in ChessRule class to avoid duplicate code.

(4) The main window

The main interface uses the Java swing library layout, BorderLayout as a whole layout, the chessboard is located in the center, and other additional functions, such as re-playing and repentance, are located in the east.EastPanel uses GridLayout.The state attribute is used to judge the chess game stage. 0 is the start stage, 1 is the game stage, and 2 is the end stage.

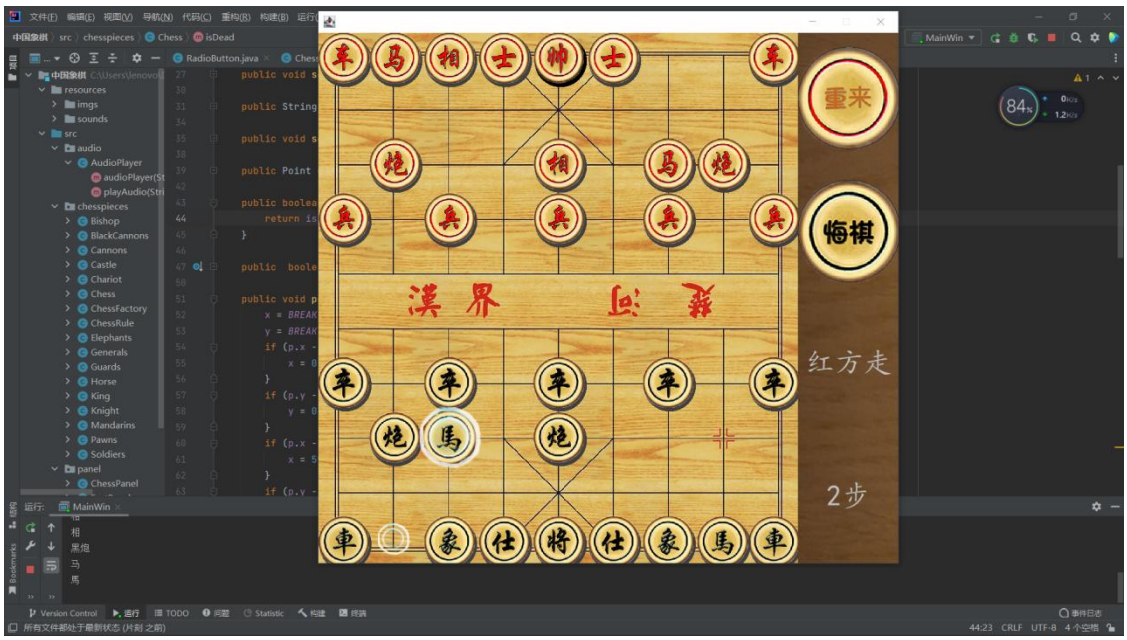


Figure 5 :Game Main Interface

3. Problem were faced

(1) Chess pieces and chessboard cannot follow the relative movement of the window.

(2) After the two tasks are completed, I found that initial interface and the end interface are not considered in the requirements.

(3) The repentance button is designed but has no function.

4. Solution were found

(1)Since there are many codes involved in modifying pixel coordinates, I do not modify the code to make the chessboard and chess pieces are in relative position to the window.Instead of fix the window size so that it cannot be changed.

(2)Create two new panel classes and draw the start and end interfaces

(3)The implementation of the repentance function needs to record the position change of the previous chess movement, so an array or set is used to store the position of each step from the game beginning to the game end.

5. The next immediate steps

(1) Solve the above problems and implement relevant codes.
(2)Follow the schedule to finish task3 on time.

是否符合进度？On schedule as per GANTT chart?

YES

下一步 Next steps:

Follow the schedule to finish task3 on time.

5. Task 3 [Realize a low-level AI that can only play chess simply]												
Calculate valuation function				X	X	X						
Implement Game Tree				X	X	X						
Enable maximum and minimum value search				X	X	X						
Implement α-β prune					X	X	X					
6. Task 4 [To improve the computing power of the machine.]												
Study other papers and find the method or algorithm to optimize the valuation function (unknown now)								X	X	X	X	
Increase the number of layers of the game tree (this can increase the computing power of the machine, but may cause the algorithm to be slow ,whether to increase will depend on its calculation time)								X	X	X	X	

Figure 6: Gantt Chart of Task3 and Task4

本科毕业设计（论文）中期进度报告
Project Mid-term Progress Report

学院 School		**专业 Programme**			
姓 Family name		**名 First Name**			
BUPT 学号 BUPT number		**QM 学号 QM number**		**班级 Class**	
论文题目 Project Title	The Program of Chinese Chess				

是否完成任务书中所定的中期目标？Targets met (as set in the Specification)?
[YES]

已完成工作 Finished work:

1. Mid-term targets:

Figure 1 is a screenshot from the Early-term Report.The Gantt chart is from the Project Specification. It shows the mid-term targets.

下一步 Next steps:

Follow the schedule to finish task3 on time.

5. Task 3 [Realize a low-level AI that can only play chess simply]												
Calculate valuation function				X	X	X						
Implement Game Tree				X	X	X						
Enable maximum and minimum value search				X	X	X						
Implement α-β prune					X	X	X					

Figure 1:Task schedule

Specifically, the mid-term goal was to implement a low-level artificial intelligence (AI) that can only play chess against a human player. The test standard was for the program to be able to beat the level of an inexperienced Tencent Chinese chess AI machine (i.e., a "novice" or "小白"). To achieve this goal, the main algorithms proposed were alpha-beta search trees and a Chinese chess evaluation function. The evaluation function was used to score the game state of a Chinese chess match, while the alpha-beta search tree was used to prune unnecessary search branches to improve search efficiency.

2. Finished work:

Project management:

In this stage, I continued to iterate on the Chinese chess program developed previously. As of today (Feb. 14, 2023), two new features have been added. The first is the introduction of the game start and end interfaces, while the second is the integration of an AI algorithm that enables the computer to play against the user. The user can choose to play as either the red or the black player.

In terms of schedule, the three user experience-related issues left over from the previous stage were addressed first. These were: (1) the addition of a start interface that offers two buttons for the user to choose whether to play as the red or the black player, (2) the improvement of the undo(悔棋) function such that clicking the "undo" button on the main interface will undo the last move made, and (3) the implementation of a game end interface that displays the winner and the number of moves, as well as a "restart" button to allow the user to start a new game.

Then, I designed and implemented a simple AI algorithm based on alpha-beta pruning and chess

evaluation functions. The game uses a three-layer alpha-beta tree from opening to endgame, and the evaluation function considers only basic scores and positional scores. Testing shows that the AI takes an average of 2-3 seconds per move during opening game(开局) and mid game(中局), and less than 1 second per move during endgame(残局). Therefore, the search depth can be increased appropriately during endgame. Additionally, the AI can defeat Tencent's beginner-level AI and some untrained chess enthusiasts.

Finally, I addressed some bugs that appeared after implementing the AI, including allowing the AI to play as the red player, starting the AI's thinking process immediately after the player moves, improving the end game interface, and integrating the undo function with the AI. It is worth noting that the updated chess program now has improved user experience features, as well as an AI algorithm that allows for a more challenging gaming experience.

I used the Leangoo product development tool to plan my project progress. By clicking on the three links below, you can view the specific subdivision of tasks and their starting times for each phase, which can provide a more in-depth understanding of the task targets and my completion status.
Task roadmap of this stage:
https://www.leangoo.com/kanban/snapshot/get/4617429/b48f7d98c152416d761e17ba3c12a44b
The fifth iteration:
https://www.leangoo.com/kanban/snapshot/get/4617428/b48f7d98c152416d46c4d7afba34905b
The sixth iteration:
https://www.leangoo.com/kanban/snapshot/get/4617430/b48f7d98c152416d2ccf0cff0323e8db

Design:

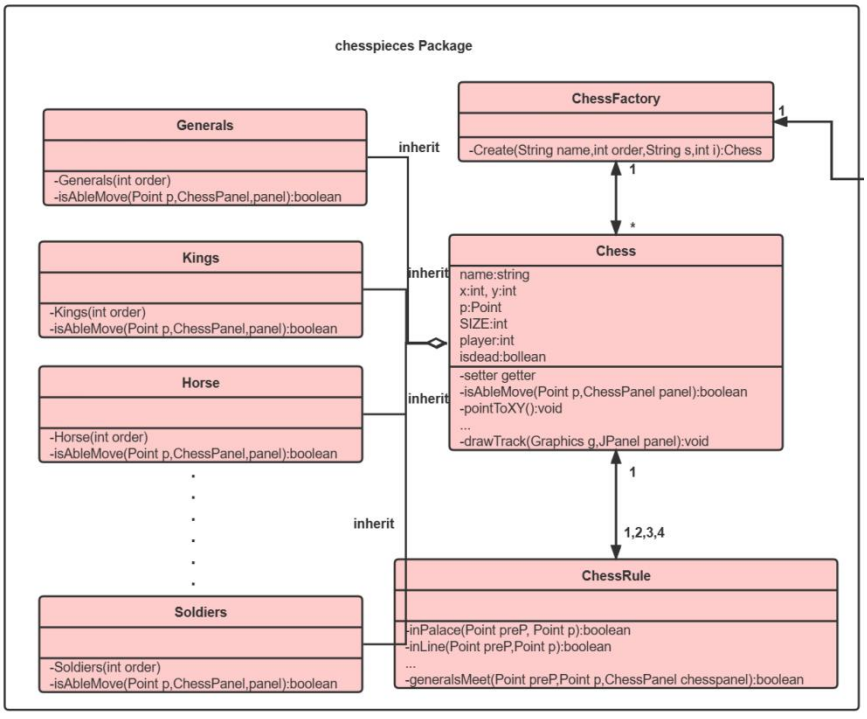


Figure 2:UML of chesspieces Package

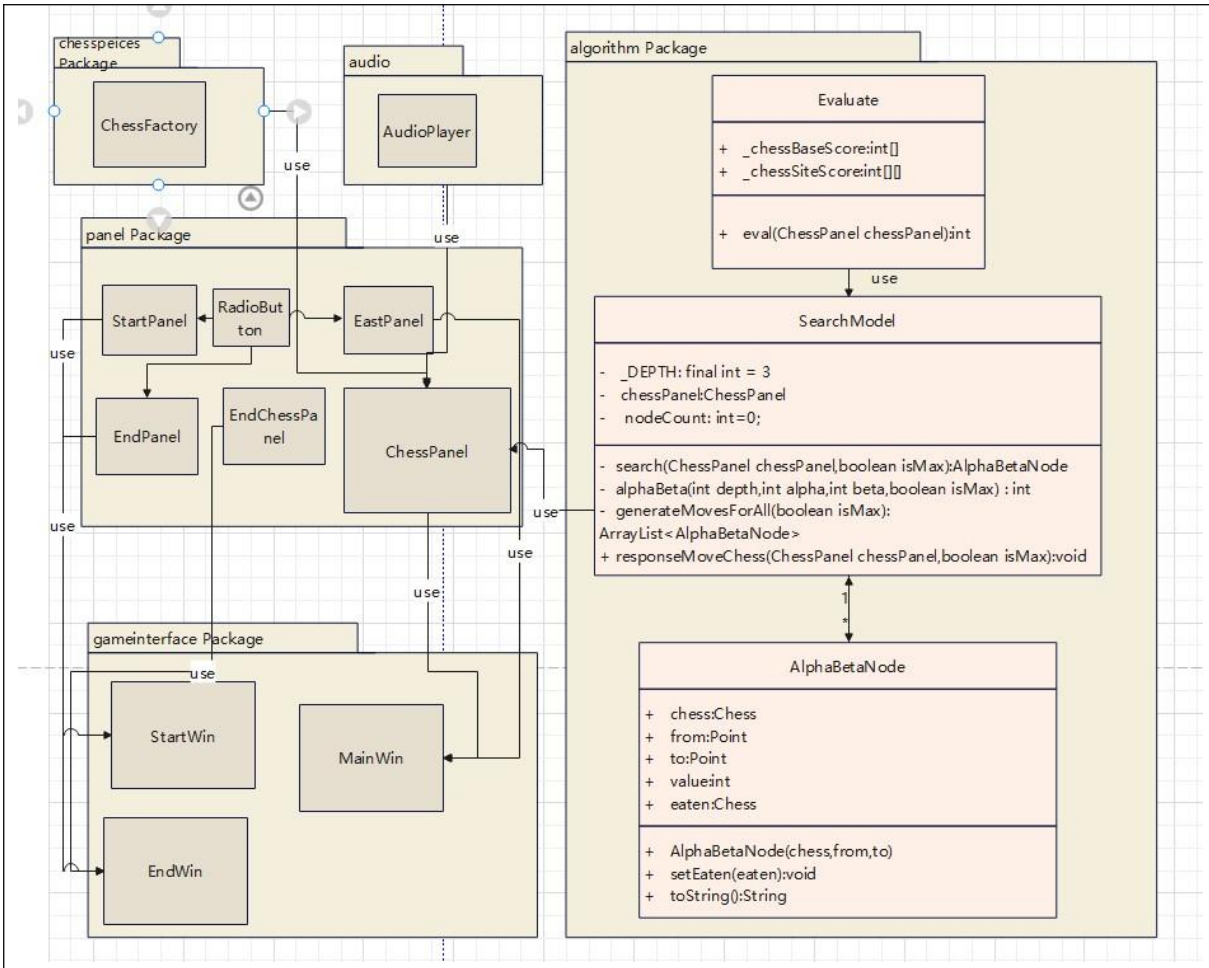


Figure 3:UML of algorithm Package and Package Relationship of Other Packages

The UML diagrams shown in Figures 2 and 3 depict the design of a chess program. Figure 3 illustrates the relationships between classes, attributes, and methods in the algorithm package. The AlphaBeta class represents all nodes in the game tree, storing information on the survival and starting positions of each Chinese chess piece. The Evaluate class computes the evaluation function value for a given node. The SearchMode class generates all valid moves and stores them in AlphaBeta nodes, applying the minimax algorithm with alpha-beta pruning to traverse the game tree. When the search reaches the best node, the program retrieves its starting and capture positions to make the optimal move for the AI.

On the left side of Figure 3, other program packages are displayed, with simplified class diagrams depicting their relationships. In contrast to the initial UML diagram, this design includes two user interfaces: StartWin and EndWin. StartWin uses the StartPanel class in the panel package, while EndWin uses the EndPanel and EndChessPanel classes. Although EndChessPanel uses some attributes and methods of the ChessPanel class, there is no inheritance relationship between them. As the chess panel class became the most important class in the program, it was modified to integrate AI functionality, including the AI move function and the undo function. The implementation details will be discussed in the following section.The object relationship diagram below provides a more comprehensive representation of the main relationships between objects.

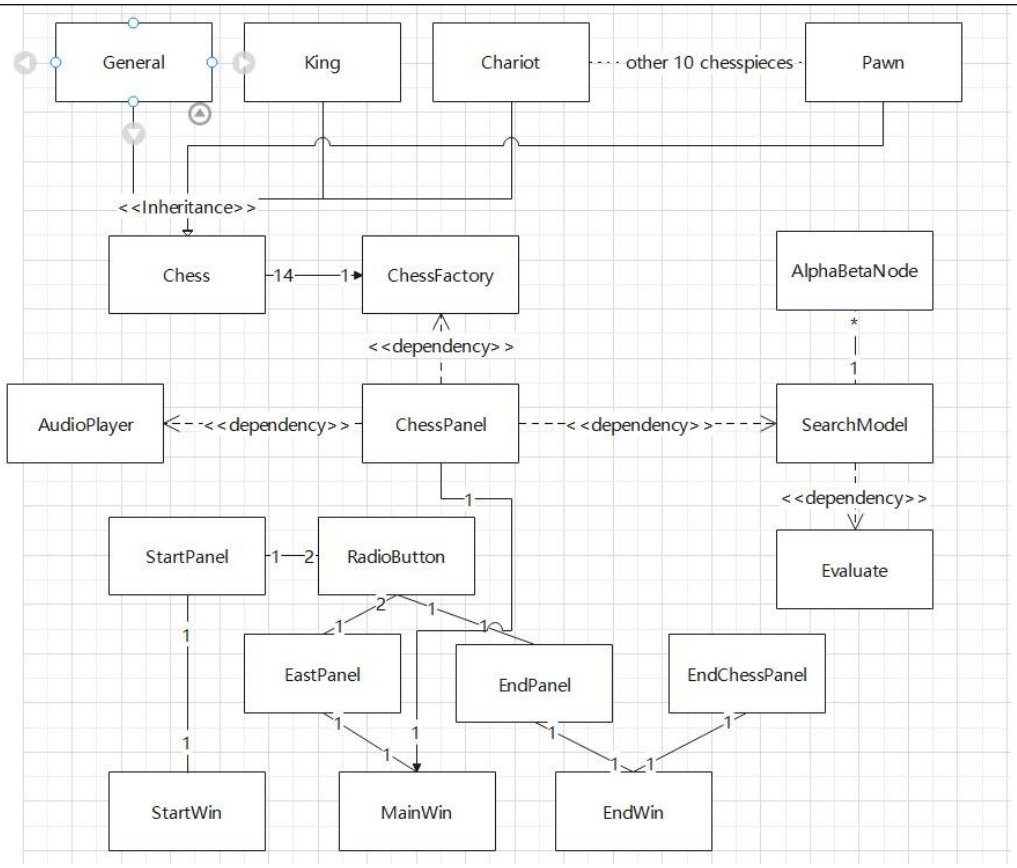


Figure 4: Object relationship

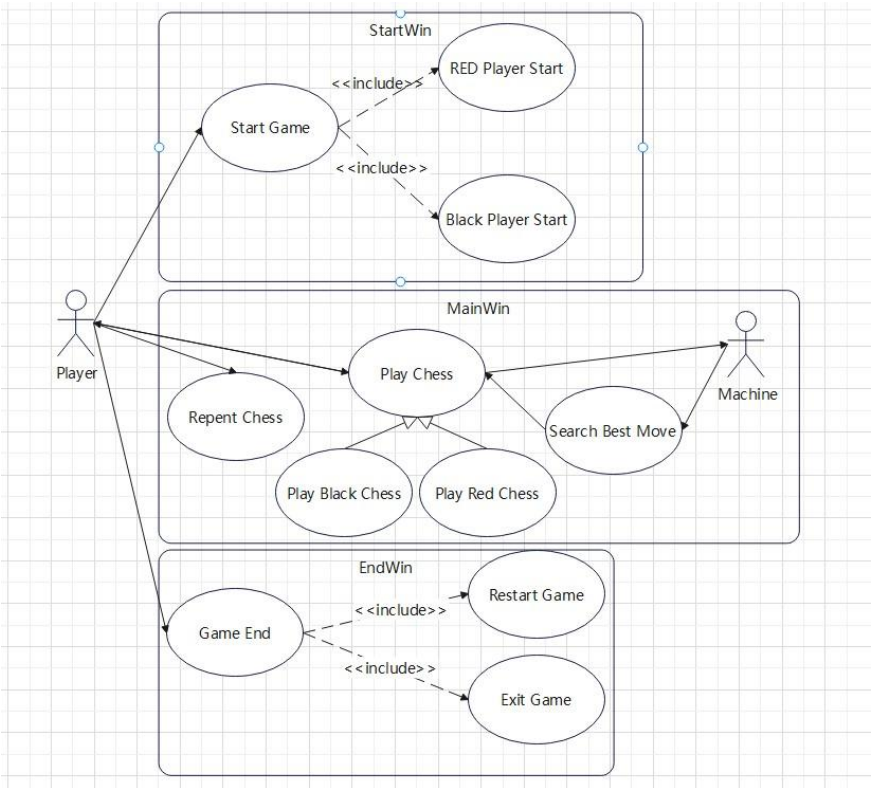


Figure 5: User case diagram

The user case diagram shown in Figure 5 depicts the process by which users utilize the Chinese chess software. Users may select to play as the red or black side at the StartWin. In the MainWin, users can play against the computer, select a piece by clicking it, then select a target location to move the piece and capture the opponent's pieces. After making a move, users may choose to undo the move. At the EndWin, users may start a new game or exit the software.

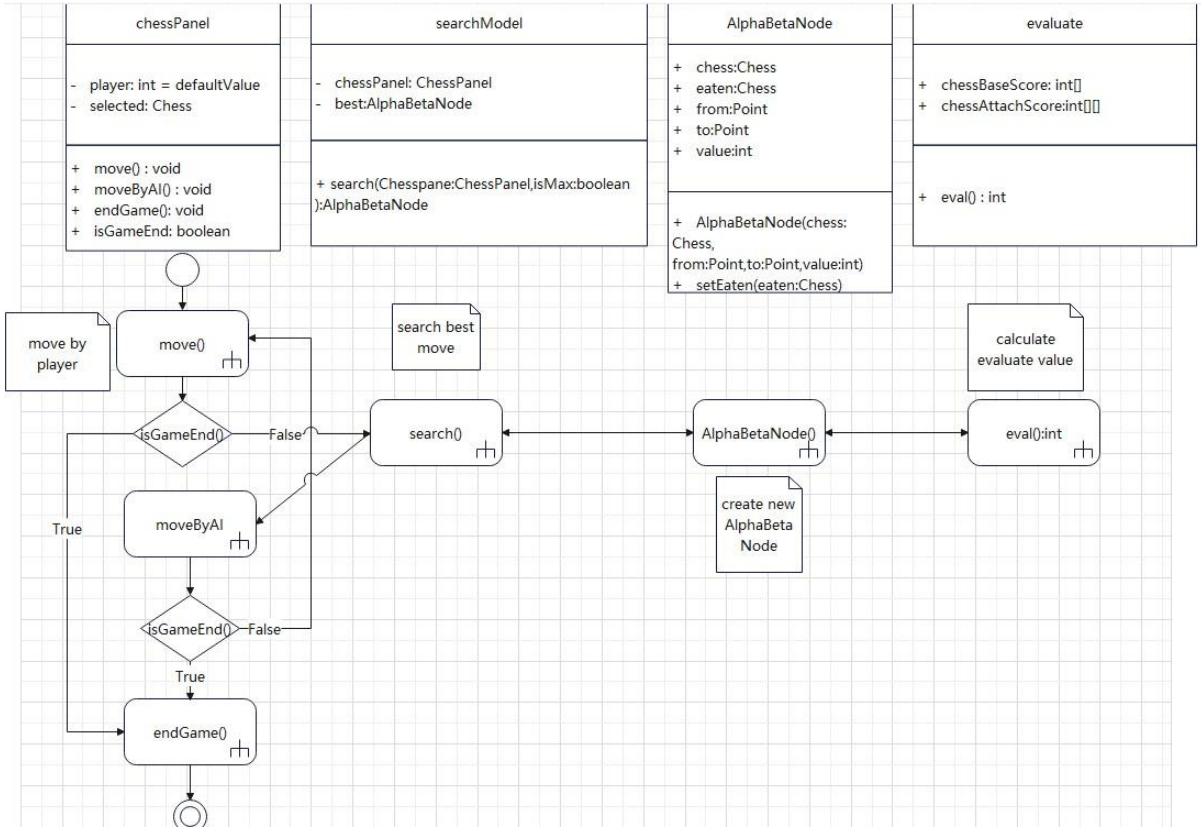


Figure 6：Object activities diagram

The object activity diagram for the player vs AI game module is shown in Figure 6, consisting of four objects: chessPanel, searchModel, AlphaBetaNode, and evaluate. Initially, the user calls the move() method in the chessPanel object to make a move, after which the computer starts an alpha-beta pruning search. Specifically, the searchModel object calls the search() method, which generates a three-layer game tree. Each AlphaBetaNode node calls the eval() method in the evaluate object to generate an evaluation function value. The searchModel object returns the optimal node, after which the MoveByAI method in the chessPanel object is called to complete the AI's move. After each move, the game end condition is checked, and if true, the game ends.

Implementation:

Development environment

Programming Language: Java
Compiler：IntelliJ IDEA 2021.3.3 (Community Edition)
IDE：openjdk-17

Key Module

(1) Artificial Intelligence Module

The artificial intelligence module is the main development module of this phase. This module consists of a package called algorithm, which contains three classes: Evaluate, SearchModel, and AlphaBetaNode. The key algorithms used are alpha-beta pruning, minimax search, and evaluation function.

Alpha-beta pruning is a commonly used optimization technique in search algorithms that reduces the number of branches in a search tree, thereby increasing the speed of the search. The algorithm uses two parameters, Alpha and Beta, during the search process. In a minimax search, Alpha represents the best-known value in the current search node, while Beta represents the best value the opponent can achieve. During the search process, if a node's value exceeds Beta, then that node and its subsequent nodes can be pruned because their values cannot be the final solution. Similarly, if a node's value is lower than Alpha, then that node and its subsequent nodes can also be pruned because their values cannot be the final solution for the current node.

In this program, a three-layer tree is used(Figure 7), and if the player is playing as black, the AI begins by performing a maximum-value search, seeking the node that can maximize the evaluation function and place the black side in the worst position. By performing a depth-first search, the value of the third-layer nodes is calculated. As the second layer is controlled by the black side, a minimum-value search is conducted, and the minimum value is updated to the Beta value. The first layer is controlled by the red side, so a maximum-value search is conducted, and the maximum value from the Beta value is selected and updated to the Alpha value. This process is repeated, and if the red side encounters a node where its Beta value is smaller than the current Alpha value, subsequent nodes do not need to be searched because the black side is guaranteed to update its value to something smaller than Beta, and the red side cannot obtain a larger Beta value. This results in the pruning of subsequent nodes, increasing efficiency.

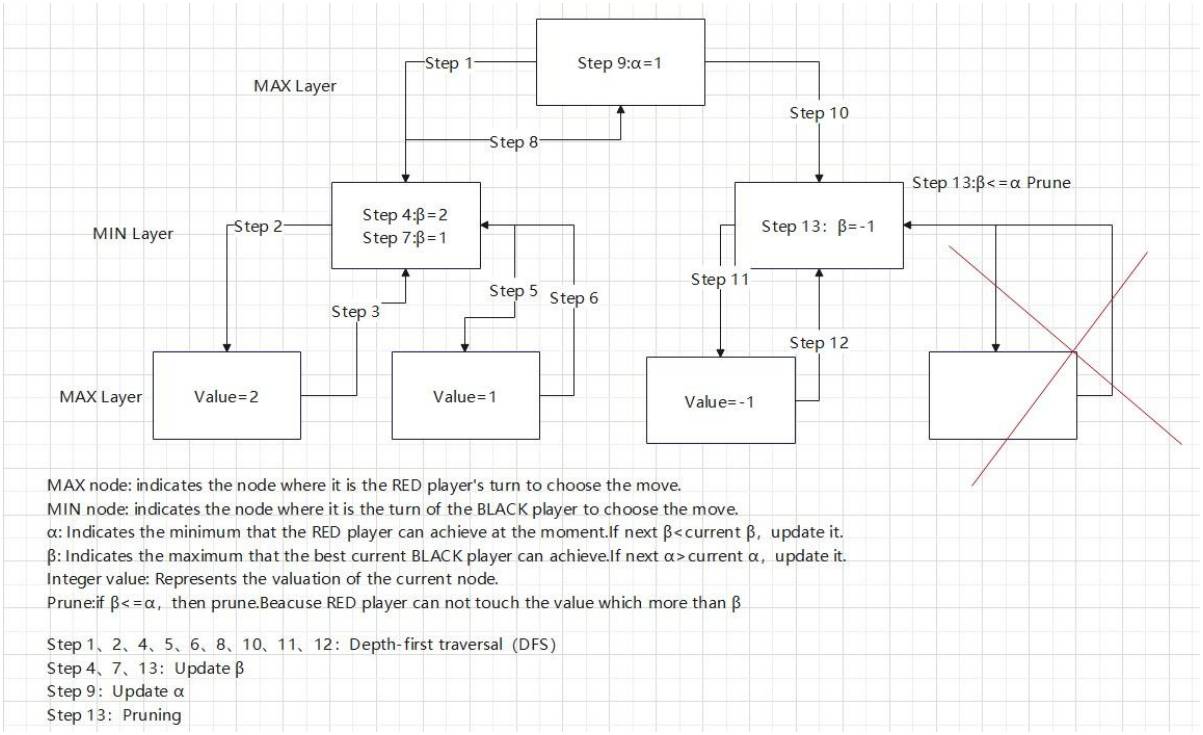


Figure 7：A Three Level Alpha-Beta Tree

```
private int alphaBeta(int depth, int alpha, int beta, boolean isMax) {//alpha-beta搜索  返回某根节点的最佳评估函数
    /* Return evaluation if reaching leaf node or any side won.*/
    if (depth == 0 ||chessPanel.isGameEnd())
        return new Evaluate().eval(chessPanel);
    ArrayList<AlphaBetaNode> moves = generateMovesForAll(isMax);

    nodeCount++;//每搜索一次即遍历一个叶子节点

    for (final AlphaBetaNode node : moves) {
        //移动
        Chess eaten=chessPanel.nextMove(node.chess, node.to);

        if (isMax) {
            alpha = Math.max(alpha, alphaBeta( depth: depth - 1, alpha, beta,  isMax: false));
        } else {
            beta = Math.min(beta, alphaBeta( depth: depth - 1, alpha, beta,  isMax: true));
        }
        //返回
        chessPanel.backMove(node,eaten);

        /* Cut-off */
        if (beta <= alpha) break;
    }
    return isMax ? alpha : beta;
}
```

Figure 8:Screenshot of Alpha-Beta Search Code

The code in Figure 8 implements the alpha-beta pruning algorithm used in the program. The algorithm is implemented using recursive depth-first search, with the termination conditions being a search depth of 0 or the end of the game. The function returns the optimal evaluation value for the current game state. The isMax variable is used to identify whether it's the maximum or minimum search layer. The calls to the nextMove and backMove methods are made to calculate the new evaluation function value after a move is made, and then move the piece back to its original position.

```
public int eval(ChessPanel chessPanel) {//计算评估函数
    int[][] values = new int[2][2];
    for (Chess chess : chessPanel.getChessArray()) {
        if (!chess.isDead()) {
            if (chess.getPlayer() == 0) {
                values[0][0] += chessBaseScore[chess.getId()];
                if (chessPanel.getOrder() == 0) {
                    values[0][1] += chessSiteScore[chess.getId()][chess.getP().x - 1 + (chess.getP().y - 1) * 9];
                } else {
                    values[0][1] += chessSiteScoreTurnDown[chess.getId()][chess.getP().x - 1 + (chess.getP().y - 1) * 9];
                }
            } else {
                values[1][0] += chessBaseScore[chess.getId()];
                if (chessPanel.getOrder() == 0) {
                    values[1][1] += chessSiteScore[chess.getId()][chess.getP().x - 1 + (chess.getP().y - 1) * 9];
                } else {
                    values[1][1] += chessSiteScoreTurnDown[chess.getId()][chess.getP().x - 1 + (chess.getP().y - 1) * 9];
                }
            }
        }
    }
    int sumRed = values[0][0] + values[0][1], sumBlack = values[1][0] + values[1][1];

    return sumRed-sumBlack;
}
```

Figure 9:Screenshot of Evaluation Value

Figure 9 demonstrates the key method of computing the evaluation function. The 2x2 two-dimensional array has four permutations. Denoted as [a][b], where a=0 represents the red player, and a=1 represents the black player. b=0 represents the basic score of the chess piece, and b=1 represents the score based on the position of the chess piece. The evaluation function is calculated by adding up the basic and positional scores of all red player's chess pieces and subtracting the sum of black player's basic and positional scores.

(2) StartWin and EndWin

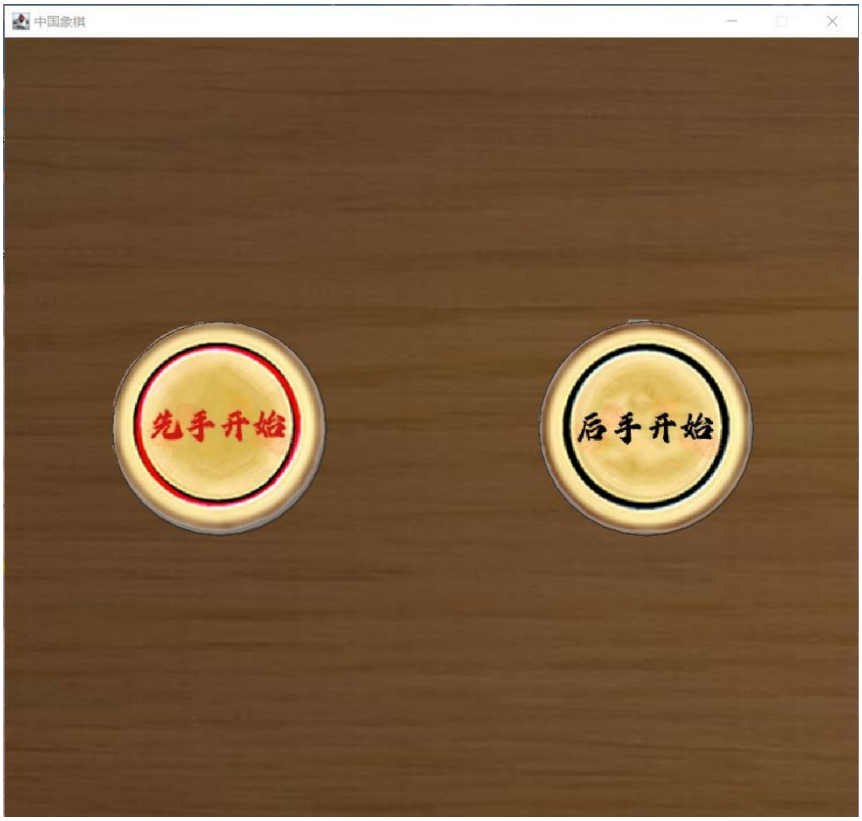


Figure 10:StartWin

Figure 11：EndWin

Testing:

In this context, the term "testing" pertains to the evaluation of the AI chess program.

Tencent Chess is a third-party chess gaming platform that provides human-machine gameplay modes and nine difficulty levels of AI. In my evaluation, the lowest level of difficulty, known as "Xiaobai," was utilized. The process involved inputting the moves generated by my chess program into Tencent Chess and vice versa. In total, 16 tests were conducted, with my program winning 15 and losing 1. According to Tencent Chess's data, the success rate of players playing against the "Xiaobai" level AI is 54.67%. Nonetheless, my program was capable of defeating the "Xiaobai" AI, fulfilling my expected goals for this phase of development.

On another aspect, I conducted statistical analysis on the search time of the three-layer game tree. The average move time during the opening and midgame was 2-3 seconds, while endgame moves were shorter with a duration of less than 1 second. The longest thinking time was 6194ms during the opening, which involved searching a total of 37509 leaf nodes. The shortest thinking time was 569ms on the fourth-to-last move before the end of the game, which involved searching a total of 4841 leaf nodes.

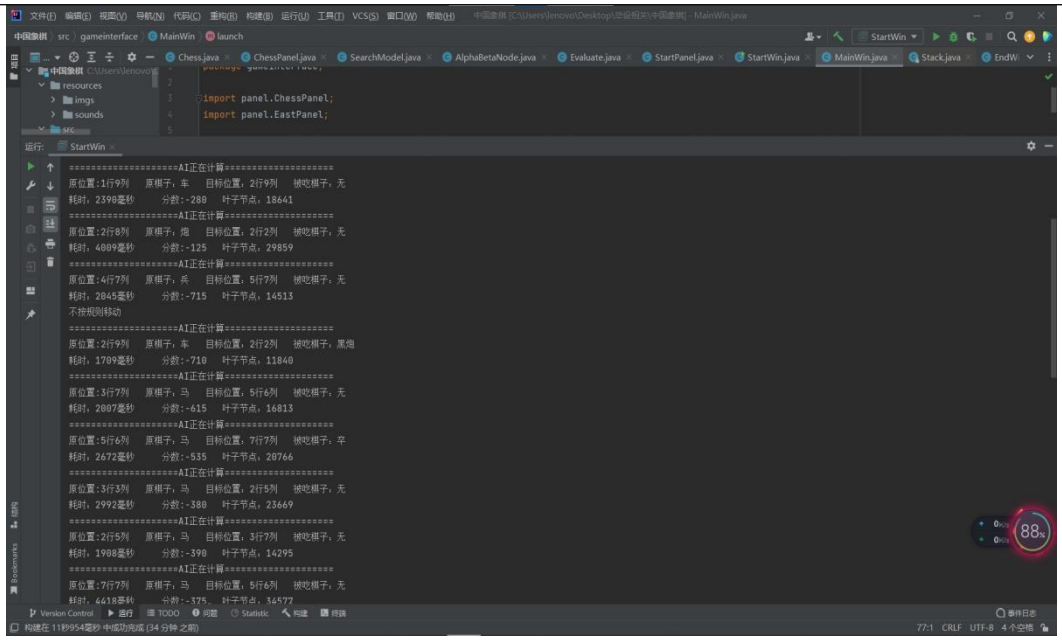


Figure 12:Test Data

Figure 13: Test AI Strength with Tencent Chinese Chess

尚需完成的任务 Work to do:

Based on the task arrangement in the Gantt chart, the next phase task is to optimize the code.

存在问题 Problems:

1. The AI chess program can only compete with amateur players with relatively weaker chess skills, and cannot defeat professional players.
2. The AI program takes slightly longer to think during the opening phase, but has faster move calculation during the endgame phase.
3. The AI program may not be able to handle some special situations, such as "empty cannon" (空头炮)and "three pieces on the same side of the board"(三子归边).

拟采取的办法 Solutions:

Based on the test results, optimization of the AI can be done in terms of chess strength and search time.

Firstly, for optimizing search time, I propose using two algorithms: opening book and transposition table. Opening book is a heuristic algorithm that analyzes common opening positions in Chinese chess based on previous high-level matches, and provides the best move. The idea is to store deeply analyzed opening sequences and corresponding moves in a database, and when the program faces a specific position, it looks for the most similar opening in the database and makes the move based on the best move stored in the database. The advantage of the opening book is that it can improve the program's chess strength, but its effect is not obvious for unknown positions. Secondly, the transposition table is an optimization algorithm aimed at reducing the repetition of searches and improving efficiency. During the search process, the transposition table records the scores of previous moves in the current position, guiding subsequent searches. Each search will sort the moves based on the scores in the transposition table to prioritize previously successful moves, thereby improving pruning efficiency. The transposition table can also be used to implement a hash table to reduce duplicate searches of the same position. Through these optimizations, the transposition table can significantly improve search speed and efficiency, especially in deep-level searches.

Secondly, to improve the AI's chess strength, I propose dividing the game into different stages, where the evaluation function changes at different stages. For example, in the endgame, the evaluation function for pawns should be increased, while the evaluation function for cannons will decrease. Another advantage of dividing the game into stages is that in the endgame, where the number of moves is reduced, the number of leaf nodes in the search tree is also reduced, and the search depth can be increased to improve the chess strength. Additionally, some special situations can be considered in the evaluation function, such as empty cannon, which refers to a situation where one player's "general" is directly exposed to the other player's "cannon". In this situation, the "general's" player cannot use the elephant and advisor for defense, giving a significant advantage to the "cannon's" player. Other situations such as linked horses, trapped cannons, and three pieces returning to the edge should also be considered in the evaluation function.

论文结构 Structure of the final report: (Chapter headings and section sub headings)

I.Introduction
1.1 Background and Motivation
1.2 Objectives and Scope of Work
II.Literature Review
2.1 History of Chinese Chess
2.2 Overview of Existing Chinese Chess Programs
2.3 Analysis of Chinese Chess AI Techniques
III.Methodology
3.1 System Architecture
3.2 Algorithm Design
3.3 Software Implementation

The Program of Chinese Chess

Supervision log

本科毕业设计（论文）教师指导记录表
Project Supervision Log

学院 School		专业 **Programme**			
姓 Family name		**名 First Name**			
BUPT 学号 BUPT number		**QM 学号 QM number**		**班级 Class**	
论文题目 Project Title	The program of Chinese Chess				

Please record supervision log using the format below:

Date: dd-mm-yyyy
Supervision type: face-to-face meeting/online meeting/email/other (please specify)
Summary:

Date: 22-10-2022
Supervision type: WeChat Video Call
Topic: Interview and acceptance
Summary:
1) The supervisor conducted a brief interview with me via video call.
2) The interview covered topics related to my academic background, interests, and project experience.
3) The supervisor ultimately decided to accept me and added my WeChat to their contact information.

Date: 8-11-2022
Supervision type: WeChat discussion
Topic: Specification discussion
Summary:
1) My supervisor and I discussed the specifics of the project's specification.
2) The main point of discussion was whether or not to design and implement an AI module for chess playing.
3) In the end, we both decided that although implementing the AI module would add some difficulty and cost, it would also bring more value and challenge to the project. Therefore, we decided to include the AI module in the project's specification.
4) After the meeting, I began writing the specification report and completed the first draft, which I submitted to my supervisor.

Date: 18-11-2022
Supervision type: online discussion
Topic:Project Progress Report
Summary:
1) My supervisor and I discussed how to fill in the ethics section of the project

specification.
2) Since we decided that there were no third-party participants, we agreed to mark this section as "No."
3) We also discussed the content related to machine learning in the specification and decided to replace it with a specific decision tree algorithm. We rephrased the language to be more concise and clear.
4) Finally, I submitted the revised report to my supervisor.

Date: 3-1-2023&7-1-2023
Supervision type: online discussion
Topic: Specification discussion
Summary:
1) My supervisor and I discussed the progress I had made on the project so far.
2) I gave a brief overview of the work I had completed, which included interface design and rule implementation.
3) I also mentioned that I had submitted an initial acceptance report for my supervisor to review.
4) My supervisor provided feedback on my progress and suggested ways to improve the project going forward.

Date: 17-2-2023
Supervision type: online discussion
Topic: Mid-term progress review
Summary:
1) My supervisor and I discussed the progress I had made on the project since the last meeting.
2) We focused on the algorithm for the human-machine gameplay feature and discussed potential solutions to improve its performance.
3) My supervisor suggested that I use images to explain the completed code and use UML diagrams to clarify the relationships between classes. He also recommended that I illustrate the user-game interaction process and the order of algorithm execution activities.

Date: 18-2-2023&19-2-2023
Supervision type: online discussion
Topic: Mid-term progress review
Summary:
1) I created images to better illustrate the functionality of the program, as recommended by my supervisor in the previous meeting.
2) I submitted my revised mid-term report to my supervisor for re-evaluation.
3) My supervisor reviewed the report and provided feedback on my progress.
4) After reviewing my work, my supervisor approved my mid-term report.

Date: 12-4-2023&13-4-2023
Supervision type: Online Discussion
Topic: Draft report and mock viva related issues

Summary:

Day 1 (12-4-2023):

1) Online meeting to discuss the draft report and mock viva.
2) Attendees included myself and my supervisor.
3) I briefly introduced the contents of the draft report to my supervisor.
4) I explained the current flaws in the chess software and presented solutions. My supervisor suggested including the pruning algorithm solution in the report.
5) I incorporated my supervisor's suggestion into the report.

Day 2 (13-4-2023):
1) Online meeting to continue discussing mock viva related issues.
2) Attendees included myself and my supervisor.
3) My supervisor pointed out that my presentation lacked core rules and strategy related to chess moves.
4) I accepted my supervisor's suggestion and made the necessary changes to my presentation.

Meeting 2:

Date: 16-4-2023&17-4-2023
Supervision type: Online Discussion
Topic: Project title and draft report feedback

Summary:

Day 1 (16-4-2023):
1) Online meeting to discuss the project title and the draft report.
2) Attendees included myself and my supervisor.
3) We decided to change the project title to "Chinese Chess Program" after discussing it.

Day 2 (17-4-2023):
1) Online meeting to continue discussing draft report feedback.
2) Attendees included myself and my supervisor.
3) My supervisor provided feedback on the report, including suggestions for better word usage, grammar, and semantics.
4) My supervisor suggested using tables to display chess piece storage and processing,
5) Compressing the acknowledgements section, and clarifying the use of traditional AI instead of deep learning to prevent reader confusion.
6) I accepted my supervisor's suggestions and made the necessary changes to the report.

Additional Appendices

Project implementation planning map

Development chart of product

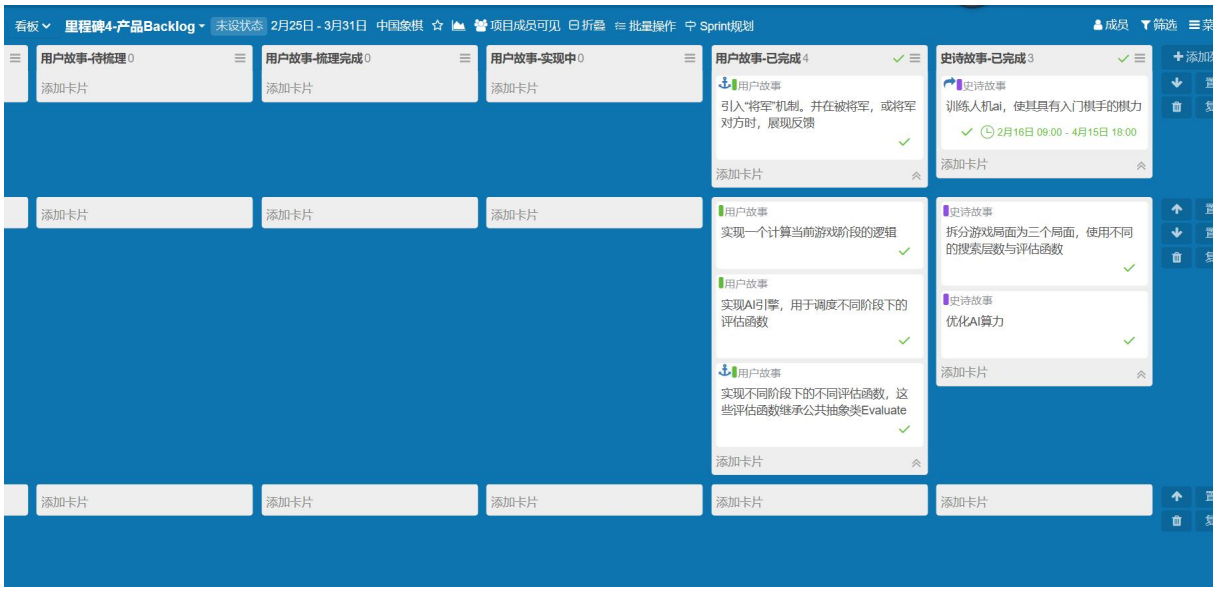
Milestone1

MileStone2

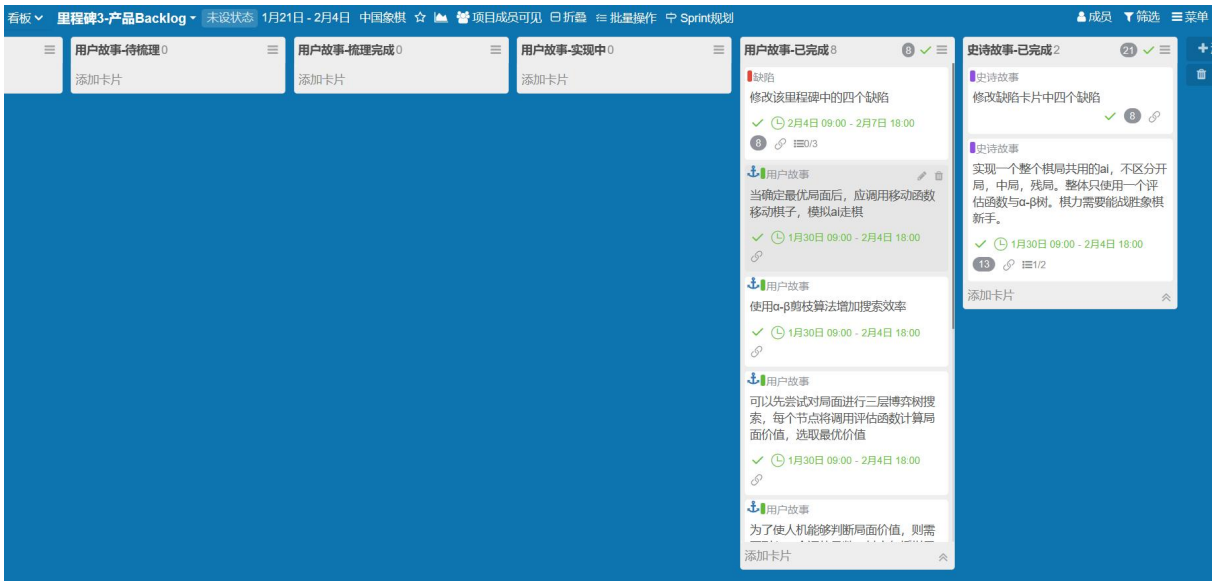

用户故事-已完成 18
史诗故事-已完成 4
判断点击的地方是否有棋子
区分棋子，重新选择棋子，移动棋子，和吃子
点击棋盘得到事件反馈
基本事件监听
3
将的移动和吃子
车的移动和吃子
马的移动和吃子
炮的移动和吃子
士的移动和吃子
兵的移动和吃子
相的移动和吃子
棋子按照规则移动
3
棋子按照规则吃子
2


红方不可操作黑方棋子
一方走完另一方才可以走
简化为将/帅被吃，结束游戏
当选择棋子时，需要有视觉反馈
显示轮次数
显示该哪一方走
当一方走完棋子，需要留下痕迹以作提示
当吃子时，需要有听觉反馈
增强交互体验，美化交互过程
5

Milestone3



Milestone4



Test case documentation

Black-box testing:

Equivalent classification:

Test case for the car:

Use case number: TB 001
Equivalent category: vertical movement, upward
Test condition:car location (1,1) and target location (1,8)
Input data: the coordinates of the vehicle location and the target location
Operation steps: Move the car to the target position
Expected result: the car was successfully moved to the target position
Actual result: The car was successfully moved to the target position

Use case number: TB 002
Equivalent category: illegal movement
Test condition: vehicle (1,1) and target (8,8)
Input data: the coordinates of the vehicle location and the target location
Operation steps: Move the car to the target position
Expected result: The car cannot be moved to the target position
Actual result: The car cannot be moved to the target position

Test case for horses:

Use case number: TB 003
Equivalent category: Japanese-shaped movement
Test condition: horse location (3,3) and target location (2,5)
Input data: the coordinates of the horse location and the target location
Operation step: Move the horse to the target location
Expected result: The horse was successfully moved to the target position
Actual results: The horse was successfully moved to the target position

Use case number: TB 004
Equivalent category: not mobile
Test condition: horse location (3,3) and target location (1,2)
Input data: the coordinates of the horse location and the target location
Operation step: Move the horse to the target location
Expected outcome: The horse cannot move to the target location
Actual results: The horse does not move to the target location

Boundary value method:

Use case number: TB 005
Test condition: the game state is black square chess
Input data: the black side player operation
Operation steps: Test whether each operation can be performed normally (such as moving black chess, moving red square chess, etc.)
Expected result: mobile black chess is executed normally, while mobile red chess cannot be executed
Actual results: mobile black chess is executed normally, while mobile red chess cannot be executed

Use case number: TB 006
Test condition: the game state is red square chess
Input data: red side player operation
Operation steps: Test whether each operation can be performed normally (such as moving red, moving black, etc.)
Expected result: Mobile red is executed normally, mobile black cannot be executed
Actual results: mobile red is executed normally, while mobile black cannot be executed

Use case number: TB 007
Test condition: the game state is red square chess
Enter the data: The Black Square General is being eaten
Operation steps: Test whether each operation can be performed normally (such as pieces movement, AI movement pieces, etc.)
Expected results: all operations can not be executed normally, prompt the end of the game, red victory
Actual results: all operations can not be executed normally, prompt the end of the game, red victory

Use case number: TB 008
Test condition: the game state is black square chess
Enter data: Red Fang is eaten
Operation steps: Test whether each operation can be performed normally (such as pieces movement, AI movement pieces, etc.)
Expected results: all operations can not be executed normally, prompt the end of the game, black victory

White box testing:

Use case number: TB 009
Test condition: Move the horse two steps left from the [3,3] and then one step down
Input data: preP = [3,3], p= [1,2], chessPanel= a board, with pieces at [2,3]
Operation steps: Perform stumble (preP, p, chessPanel)
Expected result: return false

Use case number: TB 010
Test condition: Move the horse two steps left from the [3,3] and then one step down
Input data: preP = [3,3], p= [1,2], chessPanel= a checkerboard, no chess piece at [2,3]
Operation steps: Perform stumble (preP, p, chessPanel)
Expected result: return to true

Use case number: TB 011
Test condition: Move the horse two steps to the left from the [3,3] and then one step up
Input data: preP = [3,3], p= [1,4], chessPanel= a checkerboard, with chess pieces at [2,3]
Operation steps: Perform stumble (preP, p, chessPanel)
Expected result: return false

……

Use case number: TB 013
Test condition: Move the horse two steps up from the [3,3] and then one step up to the left
Input data: preP = [3,3], p= [2,5] chessPanel= a checkerboard with pieces at [3,4]
Operation steps: Perform stumble (preP, p, chessPanel)
Expected result: return false

.....(Complete the judgment of the remaining points in turn)

Use Case Number: TB017
Test condition: Move the horse from [3,3] to the point "day"
Input data: preP = [3,3], p= [2,7] chessPanel= a checkerboard with or without pieces at [3,4]
Operation steps: Perform stumble (preP, p, chessPanel)
Expected result: return false

Basic path coverage method:

Specific examples such as logical overlay method, so no longer redundant.

Unit test:

Use case number: Stumble 001
Test condition: Move the horse two steps left from the [3,3] and then one step down
Input data: preP = [3,3], p= [1,2], chessPanel= a board, with pieces at [2,3]
Operation steps: Perform stumble (preP, p, chessPanel)
Expected result: return false

Use case number: Stumble 002
Test condition: Move the horse two steps left from the [3,3] and then one step down
Input data: preP = [3,3], p= [1,2], chessPanel= a checkerboard, no chess piece at [2,3]
Operation steps: Perform stumble (preP, p, chessPanel)
Expected result: return to true

Use case number: Stumble 003
Test condition: Move the horse two steps to the left from the [3,3] and then one step up
Input data: preP = [3,3], p= [1,4], chessPanel= a checkerboard, with chess pieces at [2,3]
Operation steps: Perform stumble (preP, p, chessPanel)
Expected result: return false

……

Use case number: Stumble 005
Test condition: Move the horse two steps up from the [3,3] and then one step up to the left
Input data: preP = [3,3], p= [2,5] chessPanel= a checkerboard with pieces at [3,4]
Operation steps: Perform stumble (preP, p, chessPanel)
Expected result: return false

.....(Complete the judgment of the remaining points in turn)

Use case number: Stumble 009
Test condition: Move the horse from [3,3] to the point "day"
Input data: preP = [3,3], p= [2,7] chessPanel= a checkerboard with or without pieces at [3,4]
Operation steps: Perform stumble (preP, p, chessPanel)
Expected result: return false

Integration testing:

top-down:

Test Case Number: SYS001
Test objective: To test the normal operation of the entire system when the player interacts with the AI
Test condition: Click the "Start game" button in the interface module, and the AI module and the rule moving module are already correct Start
Test steps:
1. The interface module displays the board and prompts the player to advance
2. The player moves on the board, and the interface module transmits the position of the drop to the rule moving module for legitimacy verification
3. After the rule movement module verification, return the legal drop position to the interface module and display the piece on the board
The interface module notifies the AI module to drop positions
4. The AI module calls the rule moving module to calculate the position of the drop
5. The rule move module returns to the drop position to the AI module
6. The AI module displays the piece on the board and notifies the interface module of the player's turn

Repeat steps 2 - 6 until one side wins or under the board is full
Expected results: The game can play normally, the player and the AI take turns, and the interface correctly displays the position of the drop and the game knot fruit

bottom-up:

Test objective: rule mobile module-> interface module-> AI module
Test content: Test rules: interaction between mobile module and interface module and interaction between interface module and AI module
test case:
Test Case Number: SYS002
Input data: the user selects the piece on the interface
Procedure: The user selects the chess pieces and moves them on the interface
Expected result: The interface module responds normally and updates the interface; the AI module normally receives the interface information and calculates the next step

Test Case Number: SYS002
Input data: The AI module calculates the next move
Operation steps: The AI module calculates the next move
Expected result: The AI module transmits the information of the next move to the interface module; the interface module responds normally and updates the interface

Test Case Number: SYS 003
Input data: The interface module sends a move request to the rule moving module
Procedure: The interface module sends a move request to the rule move module
Expected result: The rule module receives the request and calculates the movement; the rule module returns the movement result to the interface module
Test result: Through the above test cases, the interaction between the rule mobile module, the interface module and the AI module is normal, and the whole system can run normally.
Test conclusion: The bottom-up integration test of the program is passed, and the different modules of the system can work together to realize the overall function.

SystemTest:

function testing:

Number in the same integration test: SYS001

Correctness / Justification test:

Test Case Number: ChessBoard001
Test case name: Whether the status of the pieces on the board is correct after the game starts
Test objective: Make whether the status of the pieces on the board is correct after the program starts
Test steps:
Start the program and start a new game
Observe the status of the pieces on the board
expected result:
Chess positions follow the rules of chess
Red pieces are red, black pieces are black
The type and number of chess pieces conform to the rules of chess
The initial position of the piece follows the chess rules
Actual results: Pass the test

Robustness test:

Test Case Number: JXT001
Test objective: Test whether the procedure can properly handle illegal movement and can avoid crashes and bad behavior.
test condition:
Player 1 is black, and Player 2 is red.

The program is currently running.
Some pieces have been placed on the board.
Player 1 tries to make an illegal move, moving the car from position [1,1] to position [3,3] (a pawn block in the middle).
operating steps:
Player 1 selects the car [1,1] on the interface and tries to move it to the [3,3] position.
The program attempts to perform this move.
The program should display an error message to player 1 stating that the move is illegal.
expected result:
The program successfully detects the illegal move and displays an error message to player 1.
The program should not crash or produce any bad behavior.
Actual results: Pass the test

performance testing:

Test Case Number: PT 001
Test conditions: the game to the middle set, both sides have multiple pieces
Test objective: Test the response time and fluency of the system in complex games
Enter the data: The player moves the chess pieces
Operation steps: multiple operations, record the time of each operation
Expected result: The game response time is not more than 1 second on average, and the game runs smoothly.
Actual results: The average game response time is no more than 1 second, and the game runs smoothly.

Test Case Number: XN 002
Test conditions: The AI needs to calculate within 3 seconds to ensure a smooth game
Test objective: To test the speed and accuracy of the AI algorithm
Input data: The AI needs to calculate less than 3 seconds, and under the best walking method
Operation steps: record the AI calculation time, and record the pieces and winners under the AI
Expected result: the AI calculation time is not more than 3 seconds, and the chess pieces and the outcome under the AI are correct.
Actual results: AI calculation time is not more than 3 seconds, the pieces and outcome under AI are correct

User interface test:

Test Case Number: UI001
Test conditions: The player clicks on the "New Game" button on the game interface
Test steps:
Open the game program, enter the main game interface, and click the "New game" button
expected result:
The game program can start normally and enter the main interface. After clicking the "new game" button, the program can enter the game interface, The player can see the game board and the initial piece layout
Actual result: Show normal

Information Security Testing:

The procedure does not involve

Stress test:

PC games have no high concurrency

reliability test:

Test Case Number: RT001
Test name: Power-off recovery test
Test objective: Test whether the program can operate normally after power failure recovery
Test environment: when using the program, the test machine suddenly power off
Test steps:

Start the program and play the game;
The simulator is suddenly power off;
Return the power supply to the simulator;
restart routine;
Check whether the program is restored to the state of the last game, including the position and time of the chess pieces and other information;
Perform multiple tests to check whether the procedure is stable and reliable.
Expected result: The program can run normally after the power failure recovery, restore to the state of the last game, and the test results are stable and reliable.
Actual result: The program does not return to the last game state

Easy-to-use test:

Test Case Number: UX 001
Test objective: To test the accessibility of the user interface
Test condition: program startup
Test steps:
Open the program
Confirm that the interface layout is clear and easy to understand
Click on the new game button
Play the game and complete one operation
Confirm that the game operation is simple and easy to understand, without too many complex operations
Actual results: The game operation is simple and easy to understand, without too many complex operations

compatibility test:

Test Case Number: COMP001
Test Objective: Test the compatibility of games on different operating systems.
test condition:
Operating system: Windows 10, macOS Big Sur.
Software environment: the latest version of the Java environment.
Hardware environment: a desktop computer or a laptop computer.
Test steps:
Install and run the game on the Windows 10.
Install and run the game on the macOS Big Sur.
In the two operating systems respectively on the game test, man-machine battle.
test result:
Running the game on the Windows 10 is normal and all features are available.
Running game on macOS Big Sur is failed to start the game.
Test conclusion: The game runs normally on Windows 10, but has compatibility problems on macOS Big Sur, unable cannot start the game. Developers are recommended to make relevant adjustments to improve game compatibility.

Installation / reverse installation test:

Test Case Number: INS 001
Test conditions: Download the compression package from github and extract it, and then open it with the compiler
operating steps:
Download the program compression package from the website
Unzip program compression package
Open the program by using the compiler
expected result:
The program compression package was successfully downloaded and decompressed
The program opened successfully
You can run the program and play chess games correctly
Results: You can run the program correctly and play the game
There is no reverse installation involve

Risk and environmental impact assessment

Prevents the successful completion of the Project:

1. Inadequate understanding of Chinese Chess rules and gameplay mechanics

Likelihood (L): 2 (Unlikely)

Consequence (C): 3 (Very Serious)

Risk Level (R): 6 (Moderate Risk)

Action: Thorough research and consultation with domain experts to ensure accurate implementation of game rules and mechanics.

2. Insufficient expertise in Java language and artificial intelligence algorithms

Likelihood (L): 2 (Unlikely)

Consequence (C): 4 (Major)

Risk Level (R): 8 (Significant Risk)

Action: Conduct in-depth training and study of Java programming and AI algorithms or consult with experienced developers to ensure proper implementation.

3. Hardware or software malfunction during development or testing

Likelihood (L): 3 (Moderate)

Consequence (C): 2 (Serious)

Risk Level (R): 6 (Moderate Risk)

Action: Regularly backup and version control the project files, and test the game on multiple hardware configurations to minimize potential issues.

4. Project delays or budget overruns

Likelihood (L): 3 (Moderate)

Consequence (C): 3 (Very Serious)

Risk Level (R): 9 (Significant Risk)

Action: Implement effective project management practices, including setting milestones, tracking progress, and managing resources efficiently.

Causes potential harm to the environment:

1. Energy consumption and waste generation during development and testing

Likelihood (L): 1 (Rare)

Consequence (C): 1 (Minor)

Risk Level (R): 1 (Low Risk)

Action: Implement energy-saving measures, such as turning off equipment when not in use,

and recycle or properly dispose of electronic waste.

2. Energy consumption and waste generation during the game's lifecycle (e.g., user's devices)

Likelihood (L): 1 (Unlikely)

Consequence (C): 1 (Minor)

Risk Level (R): 1 (Low Risk)

Action: Optimize the game's performance to minimize energy consumption on users' devices and provide guidelines for responsible disposal or recycling of electronic waste.

Causes potential harm to people and /or animals

Negative impact on user health due to prolonged gaming sessions

Likelihood (L): 3(Moderate)

Consequence (C): 2 (Serious)

Risk Level (R): 6 (Moderate Risk)

Action: Incorporate features to encourage responsible gaming, such as reminders for taking breaks, limiting daily playtime, and providing information on potential health risks associated with excessive gaming.

Causes potential financial loss to the project or to other individuals or organizations:

Legal issues related to intellectual property or copyright infringement

Likelihood (L): 1 (Rare)

Consequence (C): 5 (Catastrophic)

Risk Level (R): 5 (Moderate Risk)

Action: Ensure that all game assets, including artwork, music, and software, are either original or properly licensed. Consult with legal professionals to avoid any potential copyright or intellectual property issues.