中山大學
SUN YAT-SEN UNIVERSITY

本 科 生 毕 业 论 文

题　　目：探讨中国象棋中的人工智能并予以实现

院　　系：软件学院

专　　业：软件工程（计算机应用软件方向）

学生姓名：林家访

学　　号：11331197

指导教师：任江涛（副教授）

二〇一五 年 四 月

摘　要

人工智能的应用范围非常广泛，计算机博弈被认为是是该领域最具挑战性的研究方向之一。棋类博弈是计算机博弈领域的代表，对它的研究显得尤为重要，在中国象棋的人机对弈中，如何使计算机更快更有杀伤力地同人类对弈以达到最终战胜人类的目的，是目前该领域研究的热点问题。

本文介绍了首先介绍了 Android 平台的基本架构和计算机博弈技术的基本知识及原理。然后，对中国象棋人机对弈进行弈棋分析和需求分析，基于需求分析规划了中国象棋游戏在 Android 系统下的设计流程及软件框架，并结合中国象棋的规则设计了博弈系统的算法；其次，对软件进行层次化的系统设计与实现，包括软件的整体架构、各个功能模块、用户界面、音效播放等，并详细介绍了程序中各个主要模块子类的关系与实现过程。在介绍模块设计的同时，对计算机博弈搜索引擎——NegaScout 算法作了详细的、由浅入深的全面讲解，同时对应用置换表、窗口原则、历史启发也都一一展开作了详细的介绍。然后，对着法生成、估值函数等计算机博弈系统必需的技术进行了介绍。对博弈系统中的设计，使用了用例图、包图、顺序图、领域模型、类图等 UML 语言作了规范、较为完整的阐述。在博弈程序开发过程中，尽量减少了个程序模块间的耦合性，改善了博弈程序的架构和可维护性，提高用户界面体验和满意度，实现了一个具有一定棋力水平的中国象棋人机对弈程序。此象棋程序实现了人机对弈、电脑难度设置、游戏音效播放等功能。最后，对博弈程序进行部署说明和应用效果的对比。

关键词： 计算机博弈；中国象棋; Android；博弈树搜索算法；软件设计

Abstract

The fields of applications of artificial intelligence are wild ranged and Computer Game is considered as one of the most challenging research directions in the field. In the field of Computer Game, Chess Game is typical representative so the research on it is particularly important. In the field of man-machine-style Chinese Chess, the issue, how to make your computer faster and more lethal to achieve the final victory over the human when playing chess with humans, is quite hot.

Firstly, this article introduces the basic knowledge and basic principles of the Android platform architecture and computer game technology. Then, the article further makes playing chess analysis and demand analysis on man-machine-style Chinese Chess, after which based on the Android system, it organizes the design processes and software framework of Chinese Chess Game, and designs algorithm of game system combined with the rules of Chinese Chess. Secondly, the essay has designed and implemented hierarchical system of the software, including the overall software architecture, all function modules, user interface and audio playback etc. Besides, the essay includes a detailed description of the relationship between the program and the implementation of each main module subclass. When introducing the module design, the essay also makes deep and comprehensive explanations to search engine of Computer Game -- NegaScout algorithm; meanwhile it describes the application of permutation table, window principle and history inspiration in detail. Thirdly, the required technology of computer game system, such as the French generation and evaluation function, are introduced. And as for the related design in Game system, the essay makes a standardized and complete explanation by using case diagram, package diagram, sequence diagram, the domain model, UML class diagrams and other UML languages. During the process of game development, this essay has minimized the coupling between two program modules, improved the game program architecture and maintainability, increased the user interface experience and satisfaction, and achieved a man-machine-style Chinese Chess program with a certain level. This chess program has achieved several main functions: man-machine module, computer difficulty

settings, the game audio playback and so on. Finally, the article explains the deployment instructions and compares application effect of the game program.

Keywords: Computer Game; Chinese Chess; Android; game tree search algorithm; software design

目　录

第一章　前言

1.1 项目的背景和意义

计算机博弈，也称机器博弈，是让计算机像人脑一样可以在遵守弈棋规则的情况下进行思考、估值、评价，从而采取对己方有利的着法与对方博弈。

在计算机诞生之前，著名的数学家和计算机学家阿伦·图灵(Alan Turing)便设计了一个能够下国际象棋的纸上程序，并经过一步步的人为推演，实现了第一个国际象棋的程序化博弈。那些世界上最著名的科学家，如计算机创始人冯·诺依曼(John von Neumann)，信息论创始人科劳德·香农(Claude E. Shannon)，人工智能的创始人麦卡锡(John McCarthy)等人都曾涉足计算机博弈领域，并做过非常重要的贡献。

从上世纪 40 年代计算机诞生，计算机博弈经过一代又一代学者的艰苦奋斗和坎坷历程，终于在 1980、1990 年代，以计算机程序战胜棋类领域的天才而享誉世界。其中最为著名的则是 1997 年 5 月 IBM“深蓝”战胜世界棋王卡斯帕罗夫，成为计算机科学史上一个不朽的丰碑。在这之后，计算机博弈一天也没有停息过拼搏。由《Science》杂志评选的 2007 年十大科技突破中，就还包括了加拿大阿尔波特大学的科研成果——解决了西洋跳棋(Checker)博弈问题，也就是说，在西洋跳棋的博弈中计算机将永远“立于不败之地”[1]。

中国象棋具有悠久的历史，深得人们的喜爱，早期的象棋是象征着当时战斗场面的游戏，后来这种观念慢慢被改变，经过千年的流传不断的完善和改进，成为人们休闲娱乐、锻炼脑力思维不可获取的好方式。中国象棋的行棋规则简单，趣味性强，载体便捷，使它本身成为雅俗共赏的竞技项目[2]。

本文的选题目的和意义在于，以理论结合实际，以工程化的方法来实现中国象棋游戏，把握整体架构和选取关键技术。通过阅读本文，读者可以了解中国象棋的设计与实现细节。

1.2 研究开发现状分析

机器棋类博弈的研究最早始于国际象棋，著名数学家，信息论创始人香农，于

1950 年提出了国际象棋程序编写的原则方法，给出了构建博弈程序的理论依据。他提出以函数评价来评价局面的优劣，把思维结果进行了量化。他还提出简化的估计方法用以剔除次要的变化。他是计算机国际象棋理论的奠基人。

和国际象棋相比，中国象棋机器博弈起步比较晚，八十年代才开始。1981 年张耀腾发表的《人造智慧在电脑象棋上的应用》，是第一篇研究中国象棋机器博弈的文章。他在他的毕业论文中以残局做实验，提出审局函数为静态子力值，棋子位置值，棋子灵活度值，威胁与保护等四项之和。1982 年廖嘉成发表的《利用计算机象棋的实验》就进了一步，包括开局、中局、残局。

同样是棋类博弈，由于文化差异、计算机软硬件发展水平的限制等其他因素，中国象棋的计算机博弈水平与国际象棋存在很大的差距。两者没有本质上的区别，区别只是形式上的不同，因此中国象棋可借鉴国际象棋的经验与技术[3]。中国象棋的计算机程序分界面部分和引擎部分，界面部分是一个可视化的游戏界面，具有显示局面和控制落子的基本功能；引擎部分必须包括数据结构定义、着法生成、局面评估和搜索机制，有些引擎还提供了开局库和残局库，供整个引擎参考走棋。引擎部分和界面部分是相互独立的，通过 UCCI 通用引擎标准协议连接。这给不同的程序之间进行比赛交流提供了方便，使研究者们可以专注于程序棋力的提高，而不用花时间编写接口程序来适应不同的界面。值得一提的是《象棋百科全书》网站对制定中国象棋引擎协议做了大量的工作[4]。

台湾大学的许舜钦教授被称为中国计算机象棋之父。他在计算机象棋博弈中取得的成果颇丰。这些研究成果为以后计算机象棋的发展做好了铺垫，至今仍在指导着人们进行计算机象棋的研究和实验工作[2]。

总的来说，中国象棋比国际象棋的研究晚，且大都理论技术都是借鉴国际象棋已有的算法，而在最流行的手机平台 Android 系统上的研究与开发也不多，有必要针对手机平台做中国象棋的研究与开发，让中国象棋更加受到普通民众的喜爱。

1.3 论文结构简介

本文将围绕基于 Android 操作系统平台的中国象棋博弈软件的设计过程展开。

第一章中，简要介绍项目的背景和意义，同时介绍计算机博弈研究的现状。

第二章中，介绍 Android 系统的在手机平台的应用现状和计算机博弈技术的原理。

第三章中，分析弈棋过程，对人机对弈进行需求建模。

第四章中，对本文中国象棋程序进行架构设计的阐述与说明。

第五章中，对博弈程序各个主要模块进行详细的算法描述与类图设计的说明。

第六章中，描述中国象棋在 Android 系统上的部署和应用效果。

第七章中，对论文工作进行总结，并对以后工作进行展望。

第二章　技术与原理

2.1　Android 及 Android SDK 技术

Android 是一个以 Linux 为基础的半开源操作系统。由于 Android 操作系统的开放性和可移植性，它可以被用在大部分电子产品上，包括：智能手机，上网本，平板电脑，个人电脑，笔记本电脑，电视，机顶盒，电子书阅读器，MP3 播放器，MP4 播放器，掌上游戏机，家用主机，电子手表，电子收音机，耳机，汽车设备，导航仪，DVD 机以及其他设备。Android 操作系统最初由 Andy Rubin 开发，被 Google 公司收购后则由 Google 公司和开放手机联盟领导及开发。2010 年末数据显示，仅正式推出两年的 Android 操作系统在市场占有率上已经超越称霸逾十年的诺基亚 Symbian 系统，成为全球第一大智能手机操作系统。在 2014 年 Google I/O 开发者大会上 Google 宣布过去 30 天里有 10 亿台 Android 设备被激活，相较于 2013 年 6 月则是 5.38 亿[5]。

Android SDK 指的是 Android 专属的软件开发工具包，它是 Google 公司提供给开发者的开发工具和函数库的集合，提供了在各种操作系统平台上的开发 Android 应用程序的开发组件，不论读者使用什么样的平台都可以开发出 Android 平台的应用软件，其中包含了 Android 平台上开发移动应用的各种工具集。通过 SDK 提供的一些工具将其打包成 Android 平台使用的 apk 文件，然后在使用 SDK 中的模拟器来模拟和测试软件在 Android 平台运行情况和效果[2]。

2.2 人机博弈的要点

人机博弈是人与计算机之间的决策计算过程，焦点在于“人脑”与“电脑”的比拼，是人工智能的一个研究分支。

人机对弈的程序，至少应具备如下几个部分：

某种在机器中表示棋局的方法，能够让程序知道博弈的状态。

产生合法走步的规则，以使博弈公正的进行，并可判断人类对手是否乱走。

从所有合法的走法中选择最佳的走法的技术。

一种评估局面优劣的方法，用以同上面的技术配合做出智能的选择。

一个界面，有了它，这个程序才能用。[6]

2.3 中国象棋计算机博弈主要技术

2.3.1 棋子、棋盘表示（Board Representations）

棋类博弈程序需要先绘制棋盘、棋子，中国象棋棋盘，可以简单地看作 10 条水平方向的线段和 9 条竖直方向的线段交叉组成。所有这些线段相交得到的 10×9 共 90 个点为棋子落子位置。中国象棋的棋子分为红黑两方棋子。“河界”把棋盘一分为二，红黑双方棋子分别布局在河界两边的区域。红黑两方各有 7 类共 32 个棋子，对棋子的表示，既可以采用同一类棋子标识的方法，也可以采用分别标识每个棋子的方法。[3]

本文程序中，用整数标识各类棋子。表 2-和表 2-分别显示了对红方棋子以及黑方棋子做的标识。

表 2-1 红方棋子标识

红方棋子	帅	仕	相	马	车	炮	兵
用作标识的整数	8	9	10	11	12	13	14

表 2-2 黑方棋子标识

黑方棋子	帅	仕	相	马	车	炮	兵
用作标识的整数	16	17	18	19	20	21	22

红方棋子从 8 开始，黑方棋子从 16 开始，目的是优化程序运行速度。

棋盘可用长度为 10×9 的二维数组表示。由于计算机做除法的效率很低，而做位运算效率很高，借鉴国际象棋的棋盘表示方法，本文将 10×9 的棋盘扩展成 16×16 的棋盘，并采用长度为 256 的一维数组存放棋盘上棋子的布局信息。这样横坐标只用和十六进制的 F 求与，纵坐标右移四位即可，效率很高。

2.3.2 着法生成 （Move Generation）

着法生成是指在当前局面下列举所有符合规则的棋子走法。博弈的规则决定了哪些走法是合法的。对有些棋类游戏来说，这很简单，比如五子棋，任何空白的位置都是合法的落子点，而对于象棋来说，有马走日蹩脚、象走田塞眼等一系列复杂的规则。

着法生成是博弈程序中一个相当复杂而且耗费运算时间的步骤，但是通过良好的数据结构，可以显著提高生成的速度。

2.3.3 搜索技术（Search Techniques）

搜索算法是搜索引擎的核心，在很大程度上决定了系统的搜索效率。对于计算机博弈程序来说，除了输赢这样的局面可以可靠地判别外，其他的判断都只能做到大致估计。判别两种走法孰优孰劣的一个好方法就是察看棋局走下去的结果。为了避免差错，需要假定对手的思考方式与我们一样，我们能想到的内容，对手也想到了，这就是极大极小搜索算法的基本原则。事实上人类棋手就是这样进行思索的，而考虑的步数越远，也就是博弈树的深度越深，棋手的水平就越高。一个优秀的棋手往往能够看到四回合以后的结果，也就是大约七八层的深度，如果搜索算法也能达到甚至超过这个深度，那就可以和人类棋手进行高水平的对弈了。除了最基础的极大极小搜索算法，还有很多其他更加智能的算法。按照对博弈树搜索策略的不同，这些算法可以分为三类：

1) 穷尽搜索(Exhaustive search): 香农提出的 A 策略，对博弈树进行完全搜索，极大极小算法就属于这一类。博弈树的规模庞大，在可接受的反应时间内穷尽搜索只能达到六层左右的深度，因此这类算法很少单独应用在博弈系统中。

2) 选择性搜索(Selective search): 香农提出的 B 策略，对博弈树剪枝，选择一部分着法展开搜索，比如 Alpha-Beta 算法，渴望搜索算法等。这一类算法是应用最为广泛的[9]。

3) 启发式搜索(Heuristic search): 也叫目标导向搜索，制定一个目标作为搜索的知道方向，优先搜索与目标接近或可以更快到目标的分支进行搜索，如贪心搜索、分支定界法等等。这类算法在中国象棋中的应用比较少[10]。

无论什么样的搜索策略，都是对一棵不完全的博弈树进行搜索。我们想对博弈树搜索的更深，就需要设计出更加合理的搜索机制，也可以采用其他的辅助手段让搜索算法能够发挥更大的效率。

2.3.4 评估函数（Evaluation）

搜索算法依据局面评价进行搜索，得到最佳算法。局面的评价值是搜索算法进行着法取舍的依据。由于我们不能将博弈树展开到分出胜负的情况（这样的博弈树太大，无法在有限的时间内展开），那就只能在博弈树展开到某一特定深度时，得到中间局面的分值，搜索算法依据这些中间局面的分值进行取舍。因此，需要一个评估函数，对中间的局面进行评价。

在搜索算法之外，评估函数是最重要的部分。局面评价的好坏将直接影响到后面对局势发展的趋势。可以想象，评估过程对具体的棋类知识非常依赖，对于人类棋手而言，对局面评价的水平将直接决定他的对弈水平。

评估函数分为动态的和静态的，动态评估函数具有自学习的能力，一般是以神经网络的形式来表示，随着一盘盘的对弈，不断调整权重，使目标最优。由于神经网络的结构选择与权重设置等问题无法从理论上解决，应用于中国象棋中的实际效果并不理想，其是否适合于中国象棋的局面评价仍在研究之中。常用的是静态评估函数[8]，它是形式一般是一个多项式：

$$PositionValue = RedValue - BlackValue \quad (2\text{-}1)$$

其中

$$RedValue/BlackValue = X1 \times PieceValue + X2 \times PiecePosition + X3 \times Flexibility + X4 \times Cooperation\ \&\ Threaten + Others \quad (2\text{-}2)$$

这个式子包括了子力价值、棋子位置、棋子灵活度、子力配合及受威胁程度等等因素。这个式子并不是一个评估函数的标准形式，实际上评估函数的设计是和设计者的棋类知识密切相关的。[7]

2.4 本章小结

本章主要介绍了本文博弈程序所依赖的平台 Android 系统和 Android SDK 技术的特点，还介绍了中国象棋计算机博弈系统中运用的相关技术，对本文采用的数据结构、着法生成以及局面评估方式作了梗概性的描述。

第三章　需求建模

3.1　弈棋过程分析

通过分析二人对弈的演化过程，建立相应的数学模型。图 3-1 给出了博弈状态演化过程图，图中表明棋局状态是在着法算子作用下进行演化的，其对应的状态转移方程可以写成

$$S_{n+1} = S_n \cdot q_{n+1}, \quad S_0 = S(0) \tag{3-1}$$

式子中 S_0 为棋局的初始局面，q_{n+1} 为第 n+1 步的着法算子，而 S_{n+1} 为下完第 n+1 步后的棋局。

由此，可以推出

$$S_F = S_0 \cdot q_1 \cdot q_2 \cdot \cdots \cdot q_F = S_0 \cdot Q \tag{3-2}$$

式中 S_F 为终局，或红方胜，或黑方胜，或和棋。显然，着法序列 $Q = \{q_1 q_2 q_3 \cdots . q_F\}$ 便是记载博弈过程的棋谱。

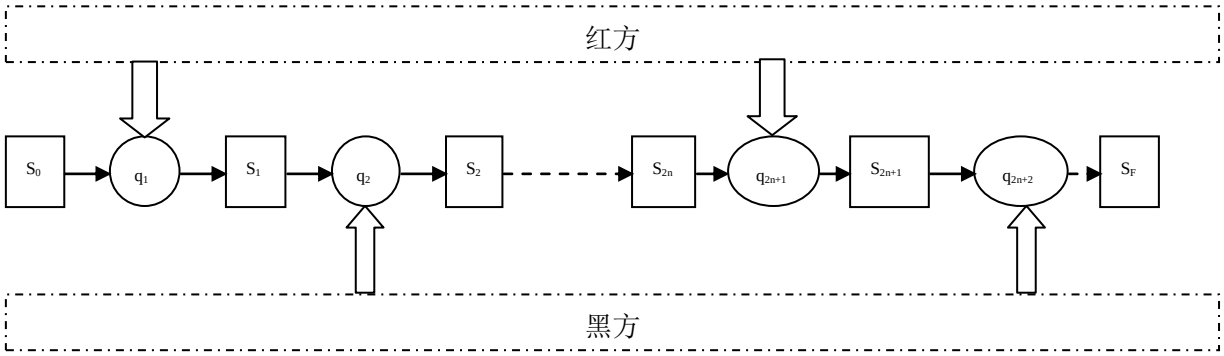

图 3-1　二人博弈状态演化过程图

下棋的过程是双方轮流给出着法，是棋局向着对本方有利的方向发展，直至最后的胜利。弈棋的核心是给出着法。这是一个复杂的思维过程：用着法推演局面，从有利的局面中选择当前的着法。[1]

3.2　让计算机下棋

首要任务是能令计算机产生一系列的着法，这种着法可以是非法的和合法的，只

要再进一步筛选出合法走法即可。为了用着法推演局面和展开博弈树，需要着法生成器，用以生成该局面下全部（或部分感兴趣）的着法。博弈者思维过程的机器实现过程如图 3-2。

双方轮流下棋，便产生了一棵决策树，也称之为博弈树的展开过程。弈棋者按照这样的思维方式，必然展开一棵规模庞大的、根在上而叶在下的博弈树，如图 3-3 所示。它与一般决策树的不同点则在于它的决策主体不是一方，而是相互对立的双方，即红方和黑方。以方节点代表红方走棋的状态，圆节点代表黑方走棋的状态，方、黑节点间的连线便对应一个个着法。博弈树上的每一个节点都代表一个棋局，棋手就是要在众多的叶子节点上挑选一个最佳的局面作为自己的选择，从而反推到根节点，产生当前的着法。[11]

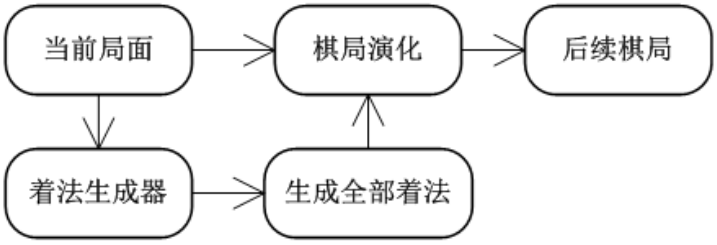

图 3-2 用着法推演棋局

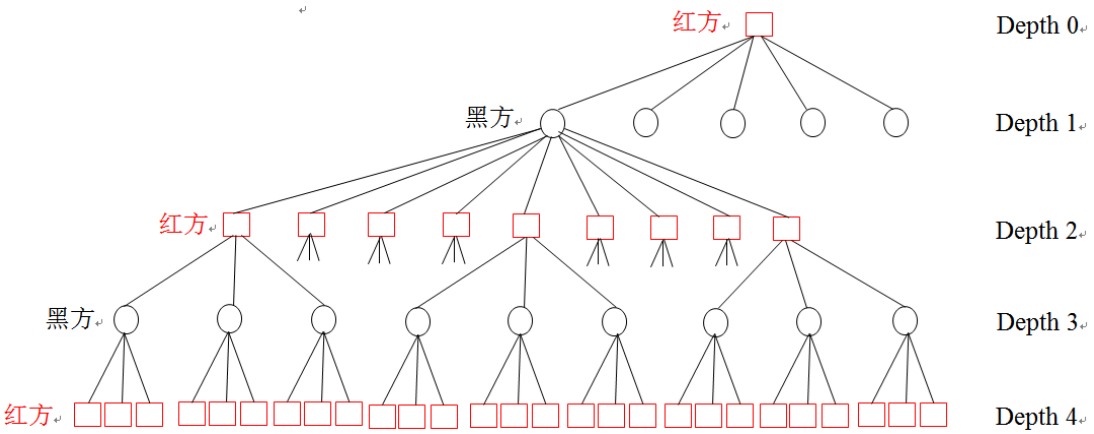

图 3-3 展开深度为 4 的博弈树

3.2 人机对弈需求建模

人机对弈的过程中，人类棋手通过观察棋盘并思考出着，计算机在产生的一系列着法中通过评估计算得到最优的着法，双方如此轮流出着，直至任何一方胜出或和棋。人机对弈的平台是棋盘，人类棋手直接操作棋盘上的棋子给出着法，此时棋盘控制权交给计算机，计算机则在产生最优着法后自动执行着法，此时棋盘控制权再次回到人类棋手。

3.2.1 人机对弈用例图

图 3-4 描述的是在中国象棋中，人类棋手与计算机棋手进行对弈的用例图。功能简述如下：（1）人类棋手可以选择计算机对手水平。（2）人类棋手下一步棋。（3）计算机搜索并产生最优着法。

3.2.2 人机对弈活动图

根据上面的用例，在这里用活动图图 3-5 详细描述人机对弈的过程。

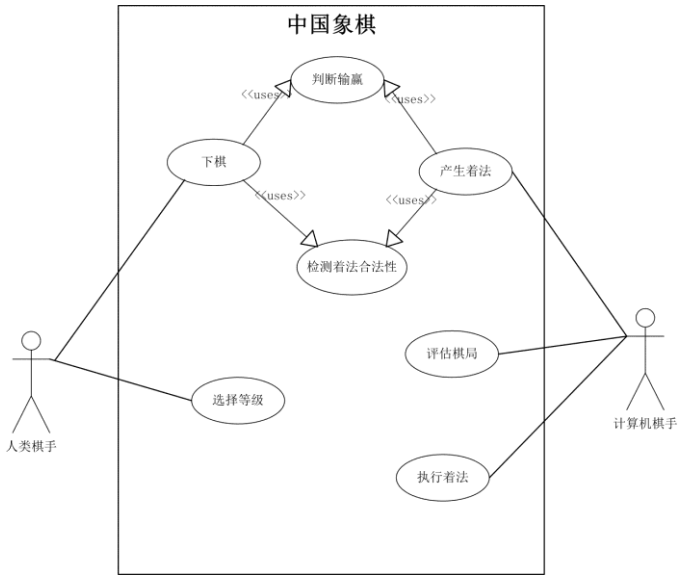

图 3-4 人机对弈用例图

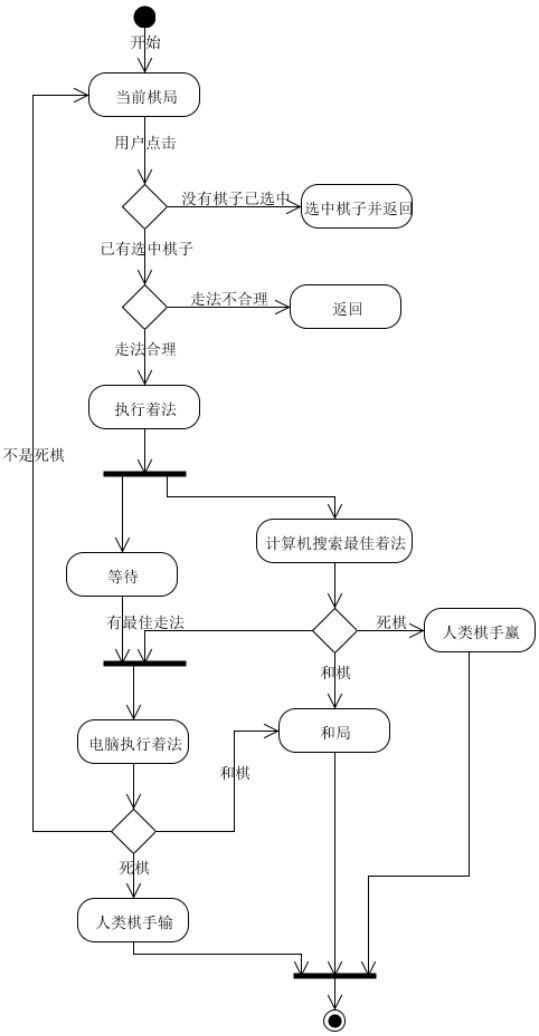

图 3-5 人机对弈活动图

3.2.3 人机对弈领域模型

从以上的用例，至少可以总结出几点，需要有一个能让人类棋手进行操作的界面，让人类棋手能够选择电脑等级和下棋，这个界面最主要的是象棋棋盘界面(GameView)，计算机棋手需要跟人类棋手对弈，它应该具有一个搜索引擎，搜索出当前局面的最佳着法，还需要一个着法生成器产生着法，一个估值器进行棋局评估。

因此，可以画出图 3-6 所示模型。

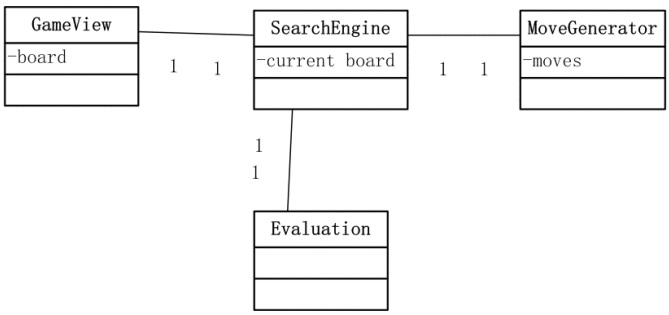
GameView
-board
1
1
SearchEngine
-current board
1
1
MoveGenerator
-moves
1
1
Evaluation

图 3-6 机对弈领域模型

第四章　架构设计

4.1　计算机博弈系统架构

中国象棋的博弈系统主要分为棋盘、搜索引擎、着法产生器、估值四大部分，但是要提高象棋的棋力，必须提高搜索速度，Alpha-Beta 剪枝、迭代深化、置换表、历史启发等手段的综合使用将搜索效率提高了几个数量级[6]。不考虑迭代深化，将 Alpha-Beta 剪枝、置换表和历史启发综合到本文的博弈系统中。一个简单但仍然可以做到高效的象棋程序，如图 4-1 所示，由 UI 包的 GameView 访问 Game 包中的搜索引擎(SearchEngine)，搜索引擎在着法产生器(MoveGenerator)、置换表(TranspositionTable)、估值函数(Evaluation)和历史启发(HistoryHeuristic)的配合下，将返回当前棋局下的最佳着法。

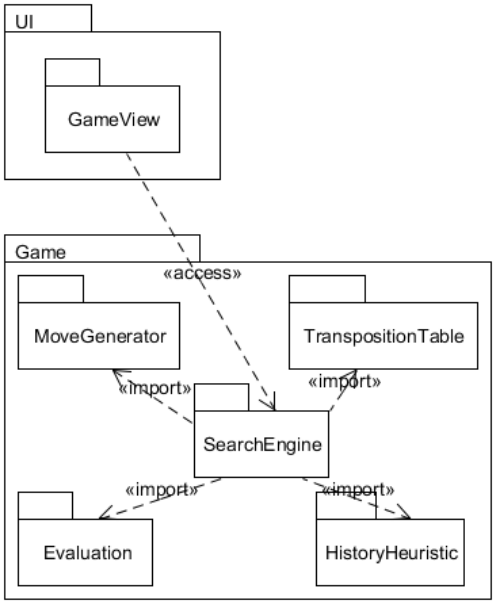

图 4-1 人机对弈系统包图

4.2　人机对弈顺序图

一个不是很完整的人机对弈顺序图如图 4-2 所示，因为象棋游戏是按照双方轮流下棋的方式进行的，所以后续棋局的发展都是像第一个回合一样的流程。

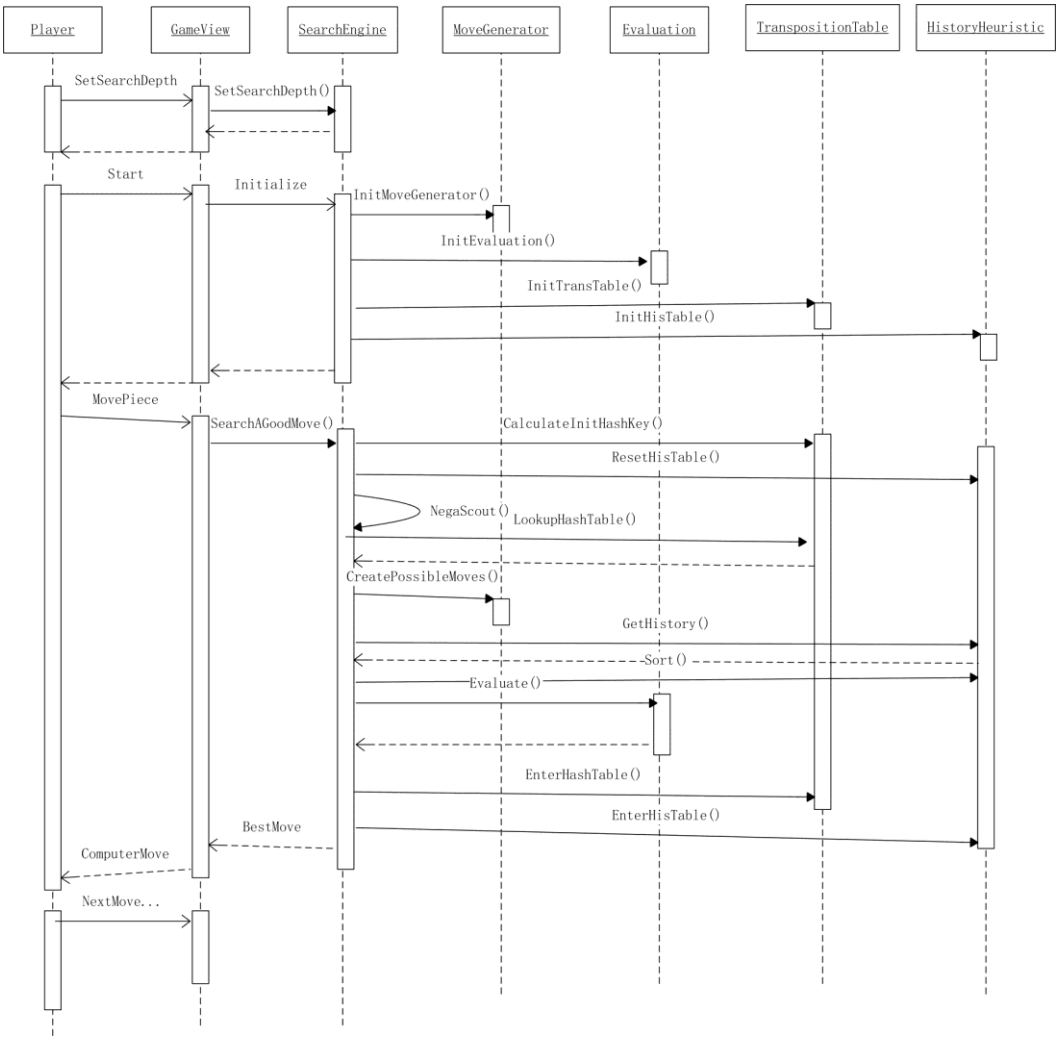
Player
GameView
SearchEngine
MoveGenerator
Evaluation
TranspositionTable
HistoryHeuristic
SetSearchDepth
SetSearchDepth()
Start
Initialize
InitMoveGenerator()
InitEvaluation()
InitTransTable()
InitHisTable()
MovePiece
SearchAGoodMove()
CalculateInitHashKey()
ResetHisTable()
NegaScout()
LookupHashTable()
CreatePossibleMoves()
GetHistory()
Sort()
Evaluate()
EnterHashTable()
EnterHisTable()
BestMove
ComputerMove
NextMove...

图 4-2 人机对弈顺序图

第五章　模块设计

5.1　搜索引擎

5.1.1 博弈树（Game Tree）

博弈事件都有着一些普遍的规律，在围棋和象棋等棋类活动中，参加者一般是两人，双方的子力在开局时完全相等，局面也是公开的，博弈双方都清楚地知道对方所采取的行动。这样的博弈事件称为二人零和、全信息和非偶然博弈。中国象棋也属于这样的博弈事件，博弈双方在对弈过程中都完全清楚每一个棋子的状态，是否存在、在棋盘中的位置、是否受到威胁和保护，进而猜测对方下一步可能采取的行动。

如果参加博弈的不是一个主体，而是对抗性的敌我双方，则搜索的过程不仅仅取决于其中一方的意愿，还取决于对方应付的策略，由此产生的搜索树，称之为博弈树[13]。博弈树搜索是与或树(AND/OR tree)。图 5-1 是典型的博弈树结构。

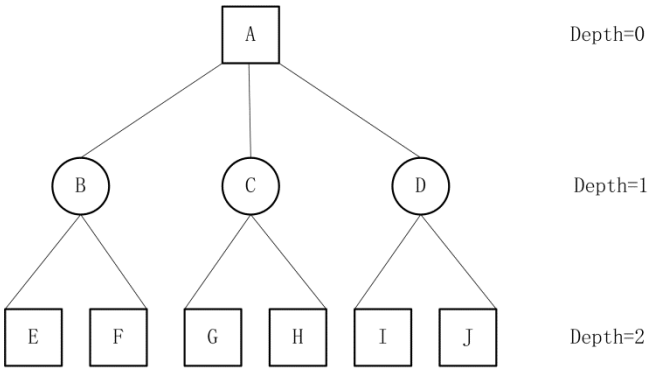

图 5-1 深度为 2 的博弈树

如图 5-1 所示，博弈双方交替地按合理走法把博弈树展开，树的每一个节点都表示某一个特定的局面。根节点 A 表示当前需要计算的局面，中间节点 B, C, D 表示对弈过程中可能出现的局面，叶子节点 E, F…等是树的最低端。叶子节点和根节点之间的距离称为搜索深度。整个博弈树描述的是从当前局面出发，包含所有可能对弈过程

的搜索树。博弈程序的任务就是对博弈树进行搜索，找出当前最优的方案。如果计算机能从初始局面开始，建立一棵博弈树，树的所有叶子节点都是分出胜负的局面，那么这棵树称为完全博弈树。如果可以建立完全博弈树，中国象棋的问题就可以得到解决：象棋的不败算法。但是中国象棋每一局面可有 20~60 种走法，以平均 40 种走法计算，双方各走 50 步，建立一棵搜索树就需要生成 40^{100} 约 10^{160} 个节点。这远远超出了当今计算机的处理能力，必须寻找其他更切合实际的方法。

解决方式是在人类棋手可以接受的时间内，让博弈树扩展到尽可能的深度，然后对没有分出胜负的叶子节点进行评估，以找到对自己尽量有利的走法。要找到对自己尽量有利的走法，就需要搜索的深度尽可能深，而搜索深度越深所需时间越长，这就爱要求博弈程序在满足性能的同时要使搜索效率尽量高[14]。

5.1.2 极小极大值算法（Minimax Algorithm）

博弈树搜索属于对抗性搜索，博弈过程当中，双方都希望自己取得胜利，所以某一方在多个行动方案可供选择时，总是挑选对自己最有利而对对方最不利的行动方案。某一方在走棋时遵守如下规则：考虑到对自己最不利的情况，从最坏的可能中选择最好的；并假定对手不会犯错误，即对手总是选择对他最有利的着法走，所以不能采取任何冒险行动。这就是极小极大原则，用此原则指导搜索的算法就叫极小极大算法[16][17][18]，其基本要点如下[15]：

设博弈的双方中一方为 A，另一方为 B。极大极小值算法是为其中一方（例如 A）寻找一个最优行动方案的方法。

为了找到当前的最优行动方案，需要对每个方案可能产生的后果进行比较。即要考虑每一方案实施后对方可能采取的所有行动，并计算可能的得分。

为计算得分，需要根据问题的特性信息定义一个估值函数，用来估算当前博弈树端节点的得分。

当端节点的估值计算出来后，再推算父节点的得分。推算的方法是：对“或”节点，选其子节点中一个最大的得分作为父节点的得分，这是为了使自己在可供选择的方案中选一个对自己最有利的方案；对“与”节点，选其子节点中一个最小的得分作为父节点的得分，这是为了立足于最坏的情况。这样计算出的父节点的得分为倒推值。

如果一个行动方案能获得最大的倒推值，则假定它就是当前最好的行动方案。

冯·诺依曼于 1928 年提出“二人零和博弈的极小极大解决方法”；而后的大约 50 年里极小极大算法没有什么改进，直到 1975 年 Knuth 和 Moore 提出负极大值方法。此方法兼顾双方的立场，消除与节点和或节点的差别，以统一的方式处理。

负极大值（Negamax）方法原理是：对于没有后代的节点和搜索层次达到极限的节点，仍采用静态估值函数来计算它们的估计值，但是要站在该节点走棋方的立场来估值；对于其他节点，均令父节点的估计值为诸子节点的估计值的负数的极大值。它是极小极大算法的一种更好的表示方式。伪代码描述如图 5-3：

```
int Negamax(position p, int depth)
{
    int value, bestvalue = -INFINITY;
    if (game over or depth <= 0)
        return winning score or eval();
    GenMove();
    for (each possible move m)
    {
        MakeMove(m);
        value = -Negamax(p, depth-1); //递归调用 Negamax 向下搜索节点
        UnMakeMove(m);
        if (value > bestvalue)
            bestvalue = value; //取最大值
    }
    return bestvalue;
}
```

图 5-3 负极大值算法

负极大值算法比极大极小算法短小并且简单。关键的不同在于：

value = -Negamax(p, depth-1)

注意其中的负号。负极大值算法的核心在于：父节点的值是各子节点值的负数的极大值。在算法的原理上，负极大值算法和极大极小值算法完全等效，负极大值算法仅仅是一种更好的表达方式。今天的博弈程序大多采用的也都是基于负极大值形式的搜索算法。

5.1.3 alpha-beta 剪枝算法(Alpha-Beta Pruning Algorithm)

Bruno 在 1963 年首先提出 alpha-beta 算法，1975 年 Knuth 和 Moore 给出了

alpha-beta 算法的数学正确性证明[19]。可以说 alpha-beta 剪枝算法是博弈树搜索的基础技术，是博弈搜索算法到目前为止使用最广泛的算法。

在极大极小搜索的过程中，存在着一定程度的数据冗余。比如：在象棋博弈的过程中，如果在某一个节点轮到甲方走棋，而甲方向下搜索节点时发现第一个子节点就可以将死乙方，则剩下的节点就无需在进行搜索了，甲方的值就是第一个子节点的值。利用类似这样的方法，可以将大量冗余的对结果不造成影响的节点抛弃。

将上述情形推广，设想有如图 5-4 左半部分所示的一棵极大极小树的片段，节点下面数字为该节点的值，节点 B 的值为 18，节点 D 的值为 16，由此可以判断节点 C 的值将小于等于 16（取极小值）；而节点 A 的值为节点 Max(B, C)，为 18，因此不再需要估节点 C 的其他子节点如 E, F 的值就可以得出父节点 A 的值了。这样将节点 D 的后继兄弟节点减去称为 Alpha 剪枝。

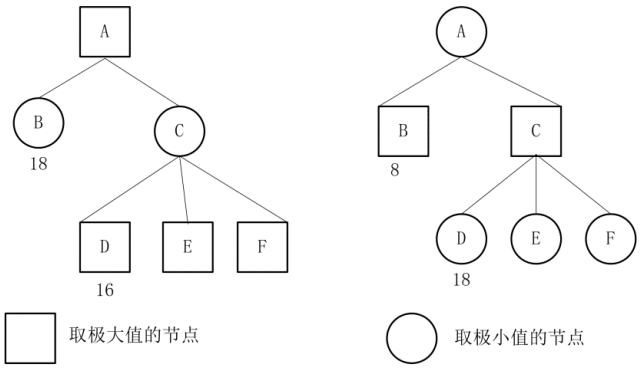

图 5-4 alpha-beta 剪枝示例

设想有如图 5-4 右半部分所示的一棵极大极小树的片段，节点 B 的估值为 8，节点 D 的估值为 18，由此可以判断节点 C 的值将大于等于 18（取极大值）；而节点 A 的值为节点 Min(B, C)，为 8，不再需要求节点 C 的其他子节点如 E, F 的值就可以得出父节点 A 的值了。这样将节点 D 的后继兄弟节点减去称为 Beta 剪枝。

将 Alpha 剪枝和 Beta 剪枝加入 Negamax 搜索，就得到 Alpha-Beta 搜索算法。将此算法用伪代码描述如图 5-5：

```
int AlphaBeta(int depth, int alpha, int beta)
{
    if (game over or depth <= 0)
        return winning score or eval();
    GenMove();
    while (MovesLeft())
    {
        MakeNextMove();
        value = -AlphaBeta(depth-1, -bata, -alpha); //递归调用
        UnMakeMove();
        if (value > alpha) //找到一步更好的棋
            alpha = value;
        if (alpha >= beta)
            break; //剪枝
    }
    return alpha;
}
```

图 5-5 Alpha-Beta 搜索算法

alpha 表示当前节点估计值的下限（已经搜索到子节点中最好的），beta 表示父节点允许当前节点估计值可以到达的上限，超过此上限，表示父节点有更好的走法，不会走此步棋，故引发剪枝。实验证明当搜索深度大于四层时，alpha-beta 剪枝算法搜索的节点数与花费的时间比单纯的极小极大值算法降低一个数量级还多。但搜索到六层已经让人不可容忍了，alpha-beta 剪枝算法还有待改进。alpha-beta 剪枝算法依赖于着法的寻找顺序，如果总是先去搜索最坏的着法，就不会引发剪枝，此时该算法就跟极小极大值算法一样，效率大打折扣，该算法最终会找遍整个博弈树。alpha-beta 剪枝有其弱点，可以通过一些策略来改进、增强 alpha-beta 剪枝算法。

5.1.4 窗口原则(Window principle)

在 alpha-beta 剪枝过程中，初始的搜索窗口往往是从-8（初始的 alpha 值）到+8（初始的 beta 值），在搜索过程中再不断缩小窗口，加大剪枝效果，这种估计是可靠的，却不是高效的。如果一开始就使用一个较小的窗口，那么就有可能提高剪枝效率，这就是窗口原则也称为有限窗口法[20]。

定理：

设博弈树中任意节点 P，P 的极小极大值为 f(P)，f(P)的近似估计值为 V，误差为 e，如果把[V-e, V+e]作为窗口执行 alpha-beta 算法后有三种情况：

若 MM <=V-e，则 f(P) <= V-e；

若 MM >= V+e，则 f(P) >= V+e；

若 V-e < MM <V+3，则 f(P) = MM；

其中 MM 是以[V-e, V+e]为窗口执行 alpha-beta 算法获得的值。

以上定理中的 1）和 2）分别称为偏高和偏低。此两种情况虽然没有准确地求出 f(P)的值，却对 f(P)的位置提供了有用的信息，可以以此为基础进行进一步的搜索。

使用窗口原则的算法有：

1) Fail-soft-alphabeta 算法

先定义一个窗口，如果极小极大值正好落在这个窗口内，算出结果即可。否则，得到有关极小极大值的位置信息，为下一步提供方便。

AlphaBeta 算法与 Fail-soft-alphabeta 算法在定理所示的第 3）种情况下运算完全一样，1）和 2）则可以提供一个参考值。以下介绍的渴望搜索将使用该参考值。

2) 渴望搜索

渴望搜索是建立在 Fail-soft-alphabeta 算法基础之上的，第一步先设定窗口[alpha, beta]，第二步用 Fail-soft-alphabeta 算法进行探路，若返回的值在窗口内表示成功，即可得出结果；不成功则修改窗口继续执行第二步。修改窗口方法：若 Fail-soft-alphabeta 算法返回的值小于 alpha，即偏高，减小 alpha 的值；若返回的值大于等于 beta，即偏低，增大 beta 的值。

3) 极小窗口搜索

极小窗口搜索算法，正式地说法是主变量导向搜索（Principal variation search），也叫 NegaScout 算法，应用相当广泛。像 alpha-beta 剪枝一样，NegaScout 算法也是双向搜索博弈树上的极大极小值。它优于 alpha-beta 之处在于它不会去搜索原本可以被 alpha-beta 剪掉的节点，但它严重依赖于一个准确的走法排序，即认为第一步是最佳的移动，其后继则次之，直到另一节点被证明是最佳的。在实际情形中，走法的排序一般可以由上一层（更浅）的搜索来决定，所以 NegaScout 算法一般需要与历史启发结合使用才能发挥出它的优势。

在一个中间节点，其第一个分支以一个完整窗口(a, b)搜索值并产生一个位于该窗中的值 v，后继的分支则以一个极小的窗口(v, v+1)搜索之。极小窗口搜索的意图在于使用极小的窗口建立极小的搜索树，以此达到高效的搜索效率。也就是说，极小窗口搜索依赖于后继节点的值相对于前驱节点值的微小变化的猜测。如果猜测被证实不正确(fail high)，随后就需要以(v+1, b)为窗口重新搜索（如果 fail low，说明这个节点不如已有的最佳节点，就不必再搜索了）。由于以(v, v+1)为窗口的搜索剪枝效率远高于更大的窗口，所以极小窗口搜索的效率很高，并且由于 fail high 以后的重新搜索相对于完整窗口也缩小了范围，其搜索效率也比以完整窗口进行的搜索效率高[6]。

图 5-6 给出伪代码形式的算法[12]：

```
function NegaScout(node, depth, alpha, beta, color)
    if node is terminal node or depth = 0
        return color × the heuristic value of node
    for each child of node
        if child is not first child
            score := -NegaScout(child, depth-1, -alpha-1, -alpha, -color) //search with a null window
            if alpha < score < beta                                        //if it failed high
                score := -NegaScout(child, depth-1, -beta, -score, -color) //do a full re-search
        else
            score := -NegaScout(child, depth-1, -beta, -alpha, -color)
        alpha := max(alpha, score)
        if alpha >= beta
            break                                                          //beta cut-off
    return alpha
```

图 5-6 NegaScout 算法

5.1.5 置换表(Transposition Table)

在极大极小搜索的过程中，改进搜索算法的目标在于将不必搜索的（冗余）分支从搜索的过程中尽量剔除，以达到搜索尽量少的分支来降低运算量的目的。实际上，还有一种方法能够降低运算量。想一下，在搜索的过程中，经常会在相同的或者不同的路径上遇到相同的局面，即着法顺序的不同可能会导致相同的局面。可否把已经搜索过的结果保存起来，下次搜索到相同局面直接取值而无需进行重复搜索？想法是可行的，这种方法就叫置换表。

置换表的实现原理是，在搜索过程中，用一张表把搜索过的节点结果记录下来，

在后继的搜索过程中，查看表中记录，如果搜索的节点已经有记录（子树的深度大于等于当前的新节点要求的搜索深度），它的信息就可以直接利用，这样可以避免重复搜索很多子树。

将这一过程结合 alpha-beta 搜索以伪代码表述如图 5-7：

```
int alphabeta(depth, &alpha, &beta)
{
    value = LookupTT(depth, index); //查询置换表
    If (value is valid)
        return value;
    Search with alphabeta…
    StoreToTT(depth, value, index); //将搜索的值记录到 TT 当中
    return value;
}
```

图 5-7 置换表结合 alpha-beta 搜索

从上述伪代码可以看出，该技术的核心是如何快速的保存与查找置换表并使得置换表的空间使用在可以接受的范围之内，二者缺一不可。因此置换表最好设置成随机存取类型并且可以在常数时间内插入，但是要在当前可以接受的存储空间内进行。哈希表就非常适合于解决这样的问题！

定义一个大小可以接受的数组，让每一个局面在数组中对应唯一的位置，但并不保证每一个位置对应唯一的局面（所有的局面加起来是一个天文数字，把所有的局面保存起来是不可能的）。不同局面对应数组中的同一存储单元的情形称为冲突，但对一次搜索而言，冲突发生的概率并不大；即使在某个搜索过的局面在改数组中没有找到，只需对它进行 alpha-beta 搜索即可。可以用图 5-8 的数组来描述哈希表：

```
enum ENTRY_TYPE{exact, lower_bound, upper_bound};
class HashItem {
    long checksum; //64 位校验码
    ENTRY_TYPE entry_type; //数据类型
    short depth; //取得此值时的层次
    short eval; //节点的值
}
HashItem hashTable[] = new HashItem[HASH_TABLE_SIZE];
```

图 5-8 哈希表数据结构

这样，对每一局面计算出一个哈希值（通常是一个 32 位的数）hashIndex，以确定它在哈希表中的位置；计算另一个哈希值（通常是一个 64 位的数）Checksum，以

确定它是否为要找的棋局，这个哈希值（64 位的数）在存入的时候已经保存在数组中。使用时，先查看哈希表 hashTable[hashIndex]，若 hashTable[hashIndex].checksum == Checksum && hashTable[hashndex].depth 大于等于当前局面需要搜索的深度，则返回 hashTable[hashIndex].eval 作为当前局面的估值。

最后一个问题是如何快速产生哈希值。对每一节点要产生 32 位的索引用来定位节点在哈希表中的位置，还要产生一个 64 位的验证值来验证表中记录的局面是否与当前局面一致。由于计算这两个值得调用十分频繁（每一局面都需要），所以要求求哈希值的过程尽量快。在博弈程序中最广泛使用的快速求哈希值的方法是 Zobrist 与 1970 年提出的[21]。

其方法如下：在程序启动时，首先建立一个多维数组 Z[pieceType][piecePosition]。其中 pieceType 为棋子种类，piecePosition 为棋子位置。本文程序中分别取 32 与 256. 然后用随机数填充此数组，要求某一局面的哈希值，则将棋盘上所有棋子在数组 Z 中对应的随机数相加，即可得到。为了避免对每一局面都要进行棋盘上所有棋子在数组 Z 中对应的随机数相加（本文程序中使用按异或操作），Zobrist 方法使用增量式计算方法：在程序的根部（棋局开始时），作一次加总操作求出根节点的哈希值，当搜索一个新节点时只要棋子在移动前将相应的随机数从哈希值中减去，再加上该棋子在移动号对应的随机数即算出了该子节点的哈希值。当然，如果移动的棋子吃了别的棋子，还有减去被吃掉的棋子被吃前所对应的随机数。当对这个局面搜索完成后，再将搜索前减掉的值加上，搜索前加上的值减掉，就恢复了当前节点的哈希值。

5.1.6 历史启发(History Heuristic)

alpha-beta 搜索的剪枝效率，几乎完全取决于节点的排列顺序。在节点排列顺序处于理想状态的情况下，alpha-beta 搜索需遍历的节点数仅为极大极小值算法所需遍历的节点数的平方根的两倍左右。也就是说对一棵极大极小树来说，如果极大极小搜索需遍历 106 个节点求得结果，那么处于理想状态的 alpha-beta 剪枝算法仅需遍历约 2000 个节点就可求得结果。因此如何调整待展开的走法序列，是提供搜索效率的关键。

历史启发就是为了迎合 alpha-beta 剪枝算法对节点排列顺序敏感的特点来提高剪枝效率的，即根据历史走法对当前搜索的节点集进行排序，从而优先搜索好的走法。

历史启发的思想是：搜索树中某个节点上的一个好的着法，对于其他节点可能是也是好的。J.Schaeffer 提出所谓好的走法即为可以引发剪枝的走法或者是其兄弟节点中最好的走法[22]。

有两个问题有待解决，1.如何给定历史得分：对于这个问题有两种考虑，一是认为某分支的搜索层数越深，搜索得到的值也就越可靠；二是认为离根节点越近，某一走法所对应的诸局面相似程度越高，越往下分支越多，相似程度越小。综合以上考虑，Schaeffer 指出每发现一个好的走法就给其历史得分增加 2^{depth} 是适合的。2.如何将一个走法映射到历史得分的数组中：用一个二维的数组，数组的前一个下标表示初始位置，后一个下标表示目标位置。本文程序中使用 256×256 的二维数组。

历史启发对 alpha-beta 搜索的增加，对于具体的棋类知识几乎无任何要求。任何棋类的类似搜索过程，只要能定义出合适的走法映射和增量因子就可以轻易的加入历史启发的增强。实验证明，加入历史启发的 alpha-beta 搜索算法，在遍历的节点数目上和时间花费上均远远少于单纯的 alpha-beta 搜索算法，而且这一差距随搜索深度的增加而增加。仅仅在最大搜索深度为 4 层的时候，二者在节点数目和搜索时间上的差距都接近 10 倍了。这一调整节点排列顺序手段所带来的效率提高充分说明了对于 alpha-beta 搜索，节点的排列顺序是极为关键的问题。

5.1.7 搜索引擎类接口

下面提供的搜索引擎类接口，图 5-9 左边，适用性非常强，实际上经过多年的研究，搜索算法有多种演化，其中大部分的负极大值搜索引擎都可从这个接口类派生，在使用上十分方便，可灵活更换搜索引擎，对于测试各个算法的优劣性非常有利。类接口说明：

1) 方法 SearchAGoodMove()供 GameView 调用，用于搜索最佳走法，并将其返回
2) 方法 SetSearchDepth()设置搜索深度，搜索深度越深博弈程序越智能，但所需要的计算时间也会大大增加。
3) 方法 SetEvaluator()添加一个估值器，只需调用 evaluator.evaluate()即能返回当前局面的估值

4) 方法 MakeMove()走一步棋，UnMakeMove()后退（撤销）一步棋
5) 方法 IsGameOver()判断当前棋局是否有一方胜出或和棋
6) 成员变量 curPosition 是当前棋局，bestMove 是最佳走法，moveGenerator 是着法产生器，evaluator 是估值器。

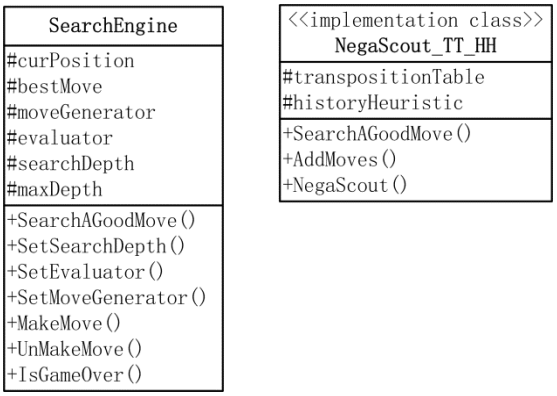

图 5-9 搜索引擎类

5.1.8 NegaScout 搜索引擎

实现该接口的 NegaScout_TT_HH 类，图 5-9 右边。类接口说明：

1) 方法 NegaScout()使用窗口原则的负极大值搜索引擎
2) 方法 AddMoves()增加已生成的着法
3) 成员变量 transpositionTable 和 historyHeuristic 是此搜索引擎使用的置换表和历史启发

它综合使用了上面介绍的窗口原则、置换表、历史启发三种策略（其中置换表采用始终覆盖策略、包括叶节点估值；历史启发表使用单层结构、每走一步直接情况历史启发信息），用伪代码描述如图 5-10：

```
int NegaScout_TT_HH(int depth, int alpha, int beta)
{
    int score, count, value = -INFINITY;
    int a = alpha, b = beta, bestvalue = -1;
    if (game over)
        return evaluate(pos);
    if (LookupTT(depth, a, b, &score, side)) //察看置换表中有无当前节点数据
        return score; //有，直接返回
    if (depth <= 0)
    {
        score = eval(pos);
        Store(depth, a, alpha, beta, side); //将搜索过的结果存入置换表
        return score;
    }
    count = GenMoves(moves);
    GetHistoryScore(moves, count); //获得历史走法的得分
    Sort(moves, count); //按历史得分排列走法顺序
    for (each possibly move m)
    {
        Make move m;
        value = -NegaScout_TT_HH(depth-1, -b, -a);
        //递归搜索子节点，对第一个节点是全窗口，其后是空窗探测
        if (value > a && value < b && m != 0)
        {
            a = -NegaScout_TT_HH(depth-1, -beta, -value); //探测结果偏高
            bestmove = m; //记录最佳走法的位置
        }
        Unmake move m;
        if (value > a) //搜索命中
            a = value; //保留最佳值
        if (a >= beta)
        {
            bestmove = m; //记录最佳走法的位置
            break; //剪枝
        }
        b = a+1; //设定新的空窗
    }
    Store(depth, a, alpha, beta, side); //将搜索过的结果存入置换表
    StoreHistoryScore(bestmove, depth); //将当前最佳走法存储到历史记录表
    return a;
```

图 5-10 NegaScout 算法结合置换表与历史启发

5.2 着法生成器

着法，就是棋子从一个位置移动到另一个位置。着法生成就是生成一个局面所有可能的有效走法，让程序在这些走法中选择最好的，并判断人的走法是否符合要求。目前的着法生成除了穷举法外没有更好的办法，同时，着法生成是一个先穷举后排除的一个过程。

5.2.1 着法生成器类图

着法生成器的类图如图 5-11。类图说明：

方法 CreatePossibleMove()生成当前棋局的所有着法

1) 方法 GenKingMoves(), GenBodyguardMoves(), GenBishopMoves(), GenHorseMoves(), GenRookMoves(), GenCannonMoves(), GenPawnMoves()分别用于生成帅（将）、仕（士）、相（象）、马、车、炮、兵（卒）的着法
2) 方法 GenKingFace()用于生成照将的着法
3) 方法 LegalMove()用于判断特定着法是否合法，方法 LegalKingFace()用于判断照将着法是否合法

图 5-11 着法生成器

5.2.2 着法生成方法

着法生成方法一般有棋盘扫描法、模板匹配法和预置表法，时常还结合使用[11]。

1) 棋盘扫描法

根据象棋规则，定义可行区域，如棋盘有效区域 A={(I, j)| 1<= i <= 10, 1 <= j <= 9}，红方半区 A = {(i, j) | 6 <= i <= 10, 1 <= j <= 9}，黑方九宫 A = {(i, j) | 1 <= i <= 3, 4 <= j <= 6}等。在已知提址和动子的情况下，根据各兵种的行棋规则，计算合法落址。有时还要考虑制约条件。表 5-1 给出了马的着法规则。其他动子以此类推。

表 5-1 提址为(i, j)动子为马的着法计算规则

棋子 moved	落子 to	有效 区域	制约条件 Subject to	吃子 Killed
马	(i+1, j+2) (i-1, j+2)	A	(i, j+1) 无子	本方子 则止 对方子 则吃
	(i+1, j-2) (i-1, j-2)	A	(i, j-1) 无子	
	(i+2, j+1) (i+2, j-1)	A	(i+1, j) 无子	
	(i-2, j+1) (i-2, j-1)	A	(i-1, j) 无子	

虽然在着法的表达上，棋盘扫描法最为直观易懂，但在着法生成的过程中需要在棋盘上反复扫描有效区域、制约条件和落址状况，时间开销巨大，实战意义不强。

2) 模板匹配法

当动子确定之后，其落址与提址的相对关系便被固定下来。于是可以为某些动子设计“模板”，只要匹配到提址，便可以迅速找到落址。图 5-12 给出了走马的匹配模板。当将马字对准提址，×表示蹩马脚的制约条件，○表示符合“走日”规则的落址，根据×的具体分布，很容易判断可能的落址。再通过单项比特矩阵比对，实现“本方子则止，对方子则吃”，完成“提-动-落-吃”内容的确定。

比较适合使用模板的动子为马和相（象）。

	○		○	
○		×		○
	×	马	×	
○		×		○
	○		○	

图 5-12 走马匹配模板

3) 预置表法

预置表法的思想是对某一类棋子考虑其在棋盘上所有可能处的位置，提前给出该类棋子在不同位置对应的可能的吃子着法和非吃子着法，并根据棋子种类和棋子位置将得到的结果存储在一个表中，这个表被称作预置表。预置表法是以空间换取时间，尽管着法生成的时间的确可以大大缩短，但是采用此类方法需要消耗大量空间资源。考虑到手机配置普遍比较低的实际情况和对时间、空间的利用，本文采用的是模板匹配法。

5.2.3 具体棋子类型着法生成

1) 马的着法生成

如图 5-12 所示，根据马的位置 N，可以迅速知道下棋位置〇和马脚位置×的下标，这些下标相对于 N 的偏移量存放在常量数组里面，可以用图 5-13 的数组表示

```
//马的步长，以帅（将）的步长作为马腿
public final static short HORSE_DELTA[][] = {{-33, -31}, {-18, 14}, {-14, 18}, {31, 33}};
//帅（将）的步长
public final static short KING_DELTA[] = {-16, -1, 1, 16};
```

图 5-13 马的着法生成所需数组

HORSE_DELTA[4][2]数组表示马可以到达的位置，KING_DELTA[4]表示对应到达位置的马腿，这样依次获取 8 个位置的信息，如果马腿出没有棋子，跳转的位置在棋盘内并且目标位置没有本方棋子，就把该走法加到可行走法列表中。马的走法可以用图 5-14 的代码表示。

```
public static int[] GenHorseMoves(int squares[], final int sqSrc, final int pSide, final boolean illeagal){
        int sqDst, pieceDst, genCount = 0;
        final int pieceSelfSide = SideTag(pSide);

        for (int i = 0; i < 4; ++i){
            sqDst = sqSrc + KING_DELTA[i];
            if (squares[sqDst] != 0){
                continue;
            }
            for (int j = 0; j < 2; ++j){
                sqDst = sqSrc + HORSE_DELTA[i][j];
                if (!InBoard(sqDst)){
                    continue;
                }
                pieceDst = squares[sqDst];
                if ((pieceDst & pieceSelfSide) == 0 || illeagal){
                    mMoves[genCount++] = Move(sqSrc, sqDst);
                }
            }
        }

        mMoves[genCount] = -1;
        return mMoves;
}
```

图 5-14 马的着法生成

2) 其他棋子着法生成

类似于马的着法生成，程序都会生成各类棋子对应的模板数组，然后根据提址匹配对应的模板。只是对不同棋子可能还需要考虑它们不同的行为，如将（帅），士（仕）只能在九宫内活动，象不能过河等规则限制。它们的着法生成原理都差不多，在此不一一列举。

5.3 评估函数

在博弈树叶子节点往往不能给出胜、负、和的结果，实际上，在弈棋过程中，大多数的博弈树叶子节点是给不出胜、负、和的结果（在残局中可做到），即使博弈树展开的深度非常深，但博弈树每增加一层，节点就会指数式增长。所以需要评价棋局

对我“有利”还是“不利”，以及“有利”或“不利”的程度，这只能依靠评估函数了。设计局面的评估函数，通常需要考虑如下不同类型的知识，并通过量化后加权组合而成。评估方法也在第 2 章介绍过，公式 2-2 给出了评估的标准，即

RedValue/BlackValue = X1 ×PieceValue + X2 ×PiecePosition + X3 ×Flexibility + X4 ×Cooperation & Threaten + Others

这个式子包括了子力价值、棋子位置、棋子灵活度、子力配合及受威胁程度等等因素。

5.3.1 子力

在中国象棋和国际象棋中，它是所有子力价值的和。简单地说就是评估双方都有哪些棋子在棋盘上。如设定兵-100，士-250，象-250，车-500，马-350，炮-350，将-10000. 这些值反应了一个棋局最基本的情况，往往为估值的过程不可或缺。

5.3.2 位置

棋子落于不同的位置其作用可能差别很大。如象棋中的车占中路、兵过河、马卧槽都是具有威胁性的位置。相反如果马窝心、兵下底又不怎么理想。于是不同的位置给予不同的分值，以表示不同的价值。

5.3.3 空间

棋盘可以分为本方控制的区域和对方控制的区域，以及有争议的区域。本方的区域包括一些棋位，它被本方的棋子攻击或保护，而不被对方棋子攻击或保护。一方控制的位置越多，可以认为一方越有利，给其越高的分值。空间的评价就是简单地把这些区域加起来，如果所含棋位的重要程度存在区别，那就在区域的计算上增加棋位重要性的因素。

5.3.4 机动性

一个棋子的活动能力越强，其价值越高。各个棋子的机动性如何，关系到棋子可

行着法的多少。显然机动性越好，可行着法越多，选择有利局势的机会也越多。

5.3.5 威胁

受到敌方棋子威胁的棋子，其价值就要相应降低。威胁的思想是步步紧逼，或取胜，或吃子，是对方仅有招架之功，而无还手之力。

5.3.6 棋局性能评估

从表面上看，棋局评估的越全面、越准确，棋力性能就会越高。但实际情况是，棋力性能 = 知识×速度，时间作为约束条件。评估中考虑的问题越多、越细致，耗费的时间则越多，必然影响到单位时间内搜索节点的数目，影响到搜索的速度和深度。过于简单的估值函数和过于复杂的估值函数同样性能不佳，在同等的知识含量下，速度越快，性能越高；在同等速度之下，知识量越大性能越高。在速度和知识量二者的相互作用下，开发者要寻找的是一种平衡，是能够使性能最大化的速度和知识量。

5.3.7 评估函数类图

本文评估函数的类图如图 5-15 所示。

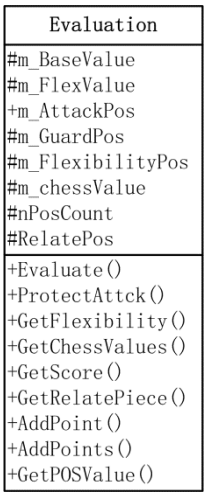

图 5-15 评估函数类图

类图说明

1) 方法 Evaluate()评估当前棋局的得分，它会调用其他方法来进行评估

2) 方法 GetRelatePiece()获取某一棋子的关联位置，即该棋子能到达的位置
3) 方法 AddPoints()配合 GetRelatePiece()添加关联位置，将关联位置保存到成员变量 RelatePos 数组中
4) 方法 ProtectAttck()计算棋子受保护和被威胁之后的得分
5) 方法 GetFlexibility()计算棋子在灵活性方面的得分
6) 方法 GetChessValue()计算棋子在子力价值方面的得分
7) 方法 GetPOSValue()获取某棋子的位置附加值
8) 方法 GetScore()根据当前走棋方计算出棋局的最终得分
9) 成员变量 m_BaseValue 存放各棋子子力价值，m_FlexValue 存放各棋子灵活性分值，m_AttackPos 是各个位置被威胁的得分，m_GuardPos 是各个位置受保护的得分，m_FlexibilityPos 是各个位置的灵活性得分，m_chessvalue 是各个位置的棋子得分

第六章　部署与应用

6.1　中国象棋博弈程序部署图

图 6-1 是本文博弈程序的部署图，它只依赖于手机的 Android 操作系统，不需要与其他设备通信

图 6-1 部署图

6.2　系统重要输入/输出界面

中国象棋重要的界面如 6-2 至 6-6 所示

6.3　应用效果

本文程序在与“象棋巫师”软件进行对弈时，当搜索深度设为 6 层时，能够击败对方的“菜鸟级”、“入门级”与“新手级”水平的计算机棋手，但此时本文的博弈程序运行速度下降的比较厉害，每一步的搜索时间需要 10-20 秒。搜索深度在 4 层以内时，搜索所需时间在 2 秒以内，基本上能够做到立即响应，而搜索深度在 3 层以内时，计算机的棋力不高，这与评估函数的知识量有很大的关系，本文的评估函数设计比较简单，所含知识量有限，未能达到比较理想的状态和效果。

图 6-2 选择电脑水平

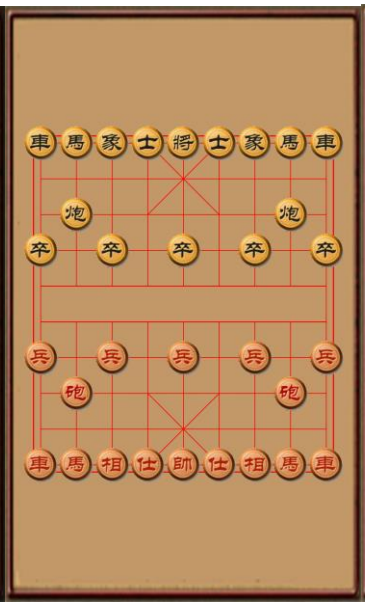

图 6-3 初始棋局

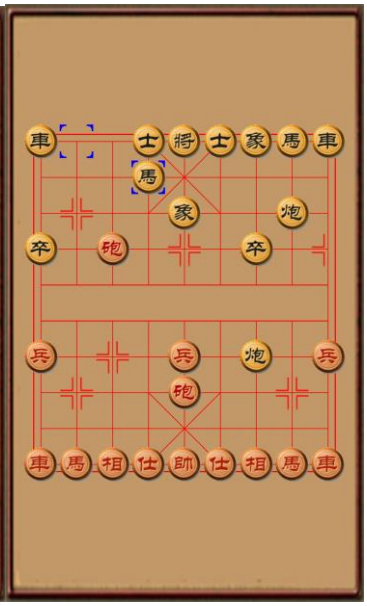

图 6-4 中局棋盘

图 6-5 玩家胜利

图 6-6 玩家输

第七章　结论

7.1 论文工作总结

本文首先介绍了计算机博弈研究的历史以及中国象棋博弈技术研究的现状，然后介绍并简要分析了当前 Android 系统的发展现状，Android 是基于 Linux 内核的操作系统，开放源码，是目前使用最广泛的智能手机操作系统。其次，还简要分析了计算机博弈所需的技术与原理。然后，对本课题进行了弈棋过程分析、人机对弈需求分析、人机对弈需求建模，通过对人机对弈的用例分析，进而分析和设计人机对弈的顺序图和系统包图，完成本文博弈程序的需求建模。之后对博弈程序的架构进行设计，对系统进行了层次化结构设计，针对主要模块进行详细的解说和设计，从类图的角度对各模块的实现进行了描述。最后，说明了系统的部署和应用效果，与现有的象棋程序进行对弈，比较中观察本文博弈程序实现的效果。

7.2 主要成果

本文取得的主要成果有：

1) 人机对弈的中国象棋游戏采用层次化结构设计，极大地减少了各程序模块间的耦合性，改善了程序的架构和可维护性。
2) 良好的代码设计，本文程序充分考虑后续维护和升级需求，对系统进行了清晰的划分。如果需要改进程序的某一模块，只需对该模块进行实现，包含必需的接口，即可应用到系统中，把之前的模块替换就可使用。
3) 对象棋程序加入了游戏音乐，提高了用户界面体验和满意度。
4) 玩家可以选择计算机程序的搜索深度，从而可以与不同棋力水平的计算机进行对弈。

7.3 问题与展望

本文博弈程序简单易用，然后，由于能力和时间有限，本文博弈程序仍有一定局限性：

1) 估值函数的优化。本文估值函数所含的棋类知识量比较少，只对最基本的子力，空间，威胁，灵活性等要素考虑在内，在相同局面下只有一种输出，没有在得分相当的着法之间进行随机选择。
2) 没有悔棋、阵营选择等功能，一旦下错棋不能回退，默认玩家选择红方阵营；对于长将局面没有判断，对于将军局面没有提醒，没有解决水平效应问题。
3) 粗陋的界面设计，由于本文的主题是探究计算机博弈所需的博弈技术与原理，在界面设计上没有花更多的时间，只是实现了最简单的功能。
4) 本文程序曾经尝试使用位棋盘方式实现，这种方法会使搜索速度更快，只是相对于扩展棋盘难度更大，最后考虑时间有限，还是选择了简单而且通用的扩展棋盘。后续的改进可以尝试用位棋盘提高搜索速度。
5) 没有加入开局库、残局库。加入开局库对于开局的棋力提升有很大的帮助，可大幅计算机的棋力。

致谢

至此，本文临近尾声，我的本科 4 年的学习生活也画上了句号。此时此刻，我首先要衷心感谢的是我的指导老师——任江涛副教授。从选题到开题，到毕业设计，到论文的撰写，任老师一直循循善诱，耐心地给我进行指导与分析。任老师精益求精的治学作风、海纳百川的学术底蕴、兢兢业业的奋斗精神和豁达开朗的人生态度无时无刻不在潜移默化地影响着我。从任老师身上，我领悟到了研究学问所必须具备的三个要素：厚积薄发的知识、严肃认真的态度和开拓创新的思维，这些都是鼓励我不断去追求真理的动力。在此，我再次向我的导师任江涛老师表示最诚挚和衷心的感谢！

同时，也要感谢余阳老师在工程论文模板方面给我们提供了指导，这个模板对我的论文章节安排给予了很大的帮助，在此我表示衷心的感谢！

参考文献

[1] 徐心和,徐长明. 计算机博弈原理与方法学概述. 中国人工智能进展（2009）-计算机博弈专辑, 1-2, 2009.

[2] 张红兵. 中国象棋人机对弈程序在 Android 系统上的设计与实现[D].西安电子科技大学,2014.

[3] 刘婉. 基于 Android 系统的中国象棋人机博弈的应用研究[D].武汉科技大学,2013.

[4] 郭秀丽. 中国象棋计算机博弈中搜索算法的研究与改进[D].河北大学,2010.

[5] 维基百科. 中文“Android”词条. [2015].http://zh.wikipedia.org/wiki/Android

[6] 王小春. PC 游戏编程（人机博弈）. 重庆大学出版社, 2002

[7] 裴祥豪. 基于剪枝策略的中国象棋搜索引擎研究[D].河北大学,2009.

[8] 肖齐英,王正志. 博弈树搜索与静态估值函数[J]. 计算机应用研究,1997,04:76-78.

[9] Bouke van der Spod. Algorithms for Selective Search. [2007].http://people.cs.uu.nl, March 8th.

[10] Jaime. Carbonell. Artificial Intelligence 15-381 Heuristic Search Methods.[2003]. www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15381-f01/www/handouts/083001.pp

[11]徐心和,王骄. 中国象棋计算机博弈关键技术分析[J]. 小型微型计算机系统,2006,06:961-969.

[12]维基百科,“Principal variation search”词条, http://en.wikipedia.org/wiki/Principal_variation_search ,April 15th, 2015

[13]Marsland TA. Computer chess and search [D].Edmonton: University of Alberta, 1991

[14]危春波. 中国象棋博弈系统的研究与实现[D].昆明理工大学,2008.

[15]王骐. 博弈树搜索算法的研究及改进[D].浙江大学,2006.

[16]TODHUN TER I.A History of the Mathematical Theory of Probability from the Time of Passcal of that of Lap lace [M] New York: Chelsea Publishing, 1965: 105-108

[17]BOREL E. The theory of play and integral equations with skew symmetric kernels[J]. Econometrica, 1953,21:91-117

[18]Rivest, R-L Game Tree Searching by MinMax Approximation. Aritficial Intelligence, 1998, Vol.34, No.1.

[19]Donald E, Knuth and Ronald W, Moore, An Analysis of Alpha-Beta Pruning Artificial Intelligence, vol.6, 1975:293-326

[20]陆汝钤. 人工智能（上册）第一版, 北京：科学出版社,1989

[21]A Zobrist A New Hashing Method with Application for Game Playing Technical Report 88, Computer Science Department, University of Wisconsin, Madison, 1970

[22]Jonathan Schaeffer, The history heuristic and alpha-beta search enhancements in practice, IEEE Transactions on Pattern Analysis and Machine Intelligence, November 1989, PAMI-(1):1203-1212.

毕业论文成绩评定记录

指导教师评语：

论文讨论了一个基于人工智能的象棋游戏软件的设计与实现，具有较强的实践意义。论文较为详细地讨论了象棋游戏中设计的人机对弈算法，分析了软件需求，进行了软件架构设计及模块设计，在设计过程中运用了软件工程方法及面向对象模型。论文结构较为清晰，文字表达较为准确，逻辑性较强，格式符合相关规范。

成绩评定：良好

指导教师签名：　　年　月　日

答辩小组或专业负责人意见：

成绩评定：

签名（章）：　　年　月　日

院系负责人意见：

成绩评定：

签名（章）：　　年　月　日

附表一、毕业论文开题报告

论文（设计）题目：探讨中国象棋中的人工智能并予以实现

选题目的：

探讨零和对弈中国象棋中的人工智能以及在受限时间与空间（Android 平台）下如何快速作出最优决策。

选题思路：

中国象棋在发明起存在了上千年，以前都是人与人之间进行对弈，自计算机诞生以来才出现了人与机器对弈。机器如何“思考”，机器为什么能比发明它的人类“聪明”？我们将深入分析和探讨其中用到的计算机科学知识与人工智能，然后在 Android 平台下编程实现中国象棋游戏。

选题方法：

从现有的游戏程序中分析前人优秀的算法并结合在课程中学习到的知识整理出自己的零和对弈算法，在 eclipse 开发环境和 Android 操作系统下，利用 Java 面向对象程序设计，Android SDK，设计模式等软件工程知识开发出一款中国象棋游戏。

相关支持条件：

Windows/Linux 操作系统，Eclipse 集成开发环境，Android 手机

进度安排：

2014 年：

11 月 8 日至 12 月 31 日，初步完成中国象棋中对弈算法设计

2015 年：

1 月 1 日至 2 月 28 日，着手 Android 平台游戏的编程实现

3 月 1 日至 3 月 30 日，撰写并提交论文初稿

4 月 1 日至 4 月 15 日，根据论文指导老师的修改意见作出修改

4 月 16 日至 4 月 30 日，视审查情况进行论文的修改完善

5 月 5 日前按学院要求完成论文最终稿提交

学生签名：　　年　月　日

指导教师意见：

1、同意开题（√）　2、修改后开题（　）　3、重新开题（　）

指导教师签名：　　年　月　日

附表二、毕业论文过程检查情况记录表

指导教师分阶段检查论文的进展情况（要求过程检查记录不少于 3 次）：

第 1 次检查

学生总结：

确定题目后，我开始找相关领域的文献并阅读，先对这一领域有一个清晰全面的认识，并理解其中的一些技术。

指导教师意见：

同意。

第 2 次检查

学生总结：

在通读各种文献的情况下，我开始设计项目的整体框架，并开始编程实现，我的初级目标是先实现一个简单的、易于理解的程序版本，在功能上做到简单，并能稳定地运行。

指导教师意见：

同意。

第 3 次检查

学生总结：

有了一个功能齐全，但效率稍差的程序，我开始尝试对程序进行优化，对部分代码进行了重构，使代码更易于管理和程序的拓展。

指导教师意见：

同意。

学生签名：　　年　月　日 指导教师签名：　　年　月　日	
总体完成情况	指导教师意见： 1、按计划完成，完成情况优（ ） 2、按计划完成，完成情况良（ √ ） 3、基本按计划完成，完成情况合格（ ） 4、完成情况不合格（ ） 指导教师签名：　　年　月　日

附表三、毕业论文答辩情况

答辩人		专 业	
论文（设计）题目			
答辩小组成员			

答辩记录：

记录人签名：　　年　　月　　日

学术诚信声明

本人所呈交的毕业论文，是在导师的指导下，独立进行研究工作所取得的成果，所有数据、图片资料均真实可靠。除文中已经注明引用的内容外，本论文不包含任何其他人或集体已经发表或撰写过的作品或成果。对本论文的研究作出重要贡献的个人和集体，均已在文中以明确的方式标明。本毕业论文的知识产权归属于培养单位。本人完全意识到本声明的法律结果由本人承担。

本人签名：　　日期：