

**Harbin Institute of Technology**

**软件设计报告**

**课程名称： JAVA网络程序设计**

**报告题目： 基于位棋盘的国际象棋程序**

**院 系： 电气工程及其自动化**

**班 级： 1106152**

**姓 名： 张博皓**

**学 号： 1110610236**

**第1部分 系统需求规格说明书**

1 引言

## 1.1编写目的

本说明书描述“基于位棋盘的国际象棋程序”的软件需求，目的在于明确说明本程序的需求，说明系统需要实现的具体功能，系统运行所需的环境，系统对数据的输入输出要求等。

## 1.2 背景

本程序可用于国际象棋爱好者之间在同一台计算机上的对战。规则严格按照现行的国际标准，并能生成符合规范的国际象棋记录。

国内现有的国际象棋软件由于开发人员对国际象棋规则的理解常常较为浅显，使得现有国际象棋程序常常并不能正确的执行国际象棋的全部规则。少有程序能够对国际象棋中“50步以内双方没有吃子且没有兵移动过则为和棋”的规则进行支持。早期版本的腾讯国际象棋甚至不能够判定“一方无子可动则为和棋”。除《Deep Fritz》等专业的国际象棋软件外，几乎没有程序能够按照国际标准正确的生成对局记录，使广大象棋爱好者多有不便，甚至误导初学者。

本程序限于设计时间以及未能找到有效的开源评价函数，因而并未能搭载AI模块，但是本程序的大多模块在设计中都考虑的使用AI的可能性。在下个版本中，如果能够合适的评价函数，AI部分可以很快的进行实装。

2 任务概述

## 2.1 目标

编写出规则严谨的国际象棋程序，并为联网对战及AI对战功能做出预留空间。

2.2 用户的特色

本程序的用户设想为资深国际象棋爱好者，对严谨的规则有着苛刻的追求，相对于华丽的画面更重视规则的正确严谨。

本程序采用了位棋盘的存储方式，并且有着法生成器，可以方便的在后续版本中高效的搭载AI使用

**第2部分 概要设计说明书**

1 引言

## 1.1编写目的

该阶段开发正式进入软件的实际开发阶段，本阶段完成系统的大致设计并明确系统的数据结构与软件结构。在软件设计阶段主要是把一个软件需求转化为软件表示的过程，这种表示只是描绘出软件的总的概貌。

1.2 背景

本程序使用了很多不同于通常的棋局存储方式，目的是提高生成着法和AI搭载后对评价函数对局面的评估速度，本部风将着重介绍这些数据存储结构

2 总体设计

## 2.1 运行环境

软件运行所需的设备如下表所示。

系统运行所需要求

|  |  |
| --- | --- |
| 硬件要求 | 计算机 |
| 操作系统 | 搭载最新版本Java虚拟机的任何兼容系统 |

## 2.2 着法处理流程示意



3 位棋盘介绍

## 3.1 常见棋盘存储方式

任何棋盘的数据结构都必须用一些额外的字节，来记录吃过路兵的目标格、王车易位权利等信息，这里的讨论暂时忽略它，因为这些因素可以独立处理，而且处理方法大同小异。

1.最基本的方法是：用一个64字节的数组，每个字节表示棋盘上的一个格子，用一个整数就可以表示格子的位置了。

　　2.早期的SARGON扩展了64字节的数组，在它的外围加了两圈“虚格”，并在这些格子上作了非法标记。这一技巧能加快着法产生的速度，例如象在走棋时会延斜线滑行，直到走到虚格上才中止。这样就没有必要用复杂的运算来预先判断象到达的格子是否会超出存储器上的表示了。第二圈虚格是给马用的，例如位于盘角的马试图跳出棋盘，这就必须用两圈虚格来保护。

　　2. MYCHESS用了相反的方法，它只用32字节表示一个棋盘，每个字节代表一个棋子，例如代表白方王、黑方王翼马前兵等，它存储的信息是棋盘上的位置，或者已经被吃的标记。这种技术有一个缺点，它无法描述由兵升变而来的其他棋子(同时这个棋子在棋盘上还有一个)。在后来的版本中，由于使用了64以后的数值代表由每一个兵升变成的每一种子(16x4=64)。这个缺点被修正了。

## 3.2 位棋盘介绍

用一个数组来表示棋盘，对于很多游戏来说，是一种简单易行的方法。但是对于像象棋、西洋跳棋之类在64格棋盘上的游戏来说，首先由苏联的KAISSA制作组提出的 “位棋盘”是一种更有效率的处理方法。

如今的主流系统采用的是64位处理器，即便是32位处理器也可以很方便快捷的把一个INT64拆分成两个INT32来处理。“64”恰巧是象棋棋盘格子的数目，所以这就有可能让一个字来表示一个棋盘，以判断某个格子的状态是1或者0。例如，一个位棋盘可以回答棋盘的每个格子“是否有白子”。

因此，本程序中一个完整的局面使用14个位棋盘表示：双方的国王、王后、堡垒、主教、骑士、士兵共12个棋盘。再加上两个用来表示“所有白子”和“所有黑子”的位棋盘，以加快运算速度。

同时本程序还在启动时预处理生成了数个诸如“骑士在B2能攻击到的格子”的位棋盘，诸如此类，这些位棋可以盘灵活运用在着法产生的运算过程中。

## 3.3 位棋盘的优势

位棋盘之所以有效，是因为很多需要循环处理的运算可以转化为处理器的一个逻辑指令。国际象棋的AI部分必定要使用博弈搜索，作为指数阶O(2n)复杂度的算法，底数的减小对于缩短程序整体的时间消耗有着至关重要的意义。

例如，当需要确认黑王是否被白后将军时，如果用简单的数组来表示棋盘，需要如下步骤：

　　1. 首先找到后的位置，这需要从64个字节中一个一个地找；

　　2. 然后在后所能走的八个方向找，直到找到王或者找到后走不到的格子为止。

这些运算是相当花费时间的，尤其如果后碰巧是在数组的最后一格。更糟的是，将军只会发生在少数情况下，但是却在每一步的搜索中都要判断，造成的时间浪费难以估量。

　　如果使用位棋盘，则步骤如下：

　　1. 载入“白方后的位置”的位棋盘；

　　2. 根据这个位置，从索引数据库中找到被该位置的后攻击的格子的位棋盘；

　　3. 用这个位棋盘和“黑方王的位置”的位棋盘作按位与运算。

　　如果结果不是零，则说明黑王被白后将军。假设被后攻击的位棋盘是储藏于存储器中的，那么整个操作只需要3到4个时钟周期

值得庆幸的是，在KAISSA小组于60年代提出位棋盘时，几乎还没有真正意义上使用64位处理器的个人电脑，所以位棋盘的速度优势是要大打折扣的。然而如今，自从AMD 速龙64 3000+推出64位计算机以来，64位处理器和操作系已经成为了主流，32位处理器已经很难在桌面和膝上电脑中找到了，位棋盘的速度优势才能被真正的发挥出来。

## 3.4 特定子攻击范围位棋盘的生成

## 3.4.1 活动能力差的棋子的攻击范围位棋盘生成

对于活动能力不强的棋子计算“棋盘上哪些格子受到某一特定棋子的攻击”非常容易。原因如图：当骑士，国王或兵在特定格子上时，无论周围情况怎样变化，所攻击的格子数目不变。你不用管那些格子上是否有棋子或者周围是否有特殊情况。

|  |  |  |
| --- | --- | --- |
| http://www.xqbase.com/computer/struct_rotation1.gif | http://www.xqbase.com/computer/struct_rotation2.gif | http://www.xqbase.com/computer/struct_rotation3.gif |

## 3.4.2 堡垒的攻击范围位棋盘生成

先计算出一个位棋盘记录车所在横线上受其攻击的格子，在计算出另一个位棋盘记录堡垒所在竖线上受其攻击的格子，最后用按位或运算把这两个位棋盘合并到一起。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | http://www.xqbase.com/computer/struct_rotation4.gif | **OR** | http://www.xqbase.com/computer/struct_rotation5.gif | **=** | http://www.xqbase.com/computer/struct_rotation6.gif | |

整个操作的要点在于预先计算。对于64格中的每一格，都预先计算出当车在这一格时，在该行仅有黑子和仅有白子各256种不同情况下该横线上受到攻击的格子，将两个棋盘通过按位与运算合并得到堡垒在该点上横线能攻击到的格子的位棋盘。

随后对棋盘进行转置，求出原棋盘在数线上能攻击到的格子的转置位棋盘，将此位棋盘再次转置得到堡垒在该格子内与横行情况相同时竖行的攻击范围。

这样，当要寻找某一情况下受车攻击的格子时，分别用横线和竖线上的上的“棋子分布状态”分辨作索引从预先计算出来的2x2x64x256个位棋盘数组中取得，并做按位或运算即可得到堡垒的攻击范围位棋盘。极大的避免了重复运算。

## 3.4.2 主教的攻击范围位棋盘生成

主教的攻击范围计算方法与堡垒类似，要点在于对棋盘不进行转置而进行旋转45度。由于本部分较为复杂，原理与堡垒的类似，而且对为实装AI模块的程序性能提升不大，因而本版本中暂时使用了传统的循环查找的方法来实现该位棋盘的生成，需要搭载AI的同时更换为旋转法。

## 3.4.3 王后的攻击范围位棋盘生成

由于王后的移动能力复合了堡垒和主教两种棋子，本程序中王后并没有专用的攻击范围位棋盘，而是在需要时分别调用该位置的堡垒和主教的位棋盘，并将两个棋盘做按位或运算得到王后的攻击范围位棋盘。

4 特殊移动规则的处理

## 4.1 王车易位

因为考虑到程序可能需要提供悔棋功能，因而未使用标志位来记录，而且查找所有之前棋盘状态来确认国王和堡垒未被位移。之后确认国王和堡垒之间未被阻挡也未被对方棋子威胁。全部判定通过则为可以易位。

## 4.2 过路兵

当兵处于可以吃过路兵的线上时（白方5线，黑房4线），判断对方是否由兵与本方兵横向相邻，如有则判断该兵上一步是否在7/2线上，全部判定通过则可以吃过路兵。

## 4.3 禁止吃王

国际象棋的规则中除快棋外禁止吃王，也就是说将己方国王暴露在对方可以攻击的格子里属于违例。因而本程序在所有的生成某个棋子着法后，均对所有可能落子位置进行一次模拟落子，如落子后造成了对方的将军，则从可以落子范围位棋盘中去除该位置。

## 4.4 兵升变

当一方兵到达对方底线是，弹出以主窗口为父窗口的模态窗口，并去除本窗口的修饰元素以避免在选择棋子前关闭。用户点击某一棋子是关闭此窗口并将所选棋子置入棋盘。

5 平局的处理

与中国象棋相比，国际象棋有着更加繁杂的平均规则，遗憾的是市面上鲜有能够完全体现出所有平局规则的程序。

## 5.1 无子可动

一方移动后开始生成对方所有棋子的可能着法，如果所有棋子的落字范围位棋盘依次做按位或运算依然为0，切当前未处于将军状态，则视为平局。

## 5.2 50步原则

50步以内双方均未移动兵或是吃子则视为平局。具体方法为检索50以内双方兵的位棋盘，看数值是否有变化，以及检查其他每个棋子，看数量是否不变，如果都满足的话则平局。

## 5.3 三次重复局面

在检查50步原则是同时查看是否出现过三组完全一样的位棋盘，如果存在则视为平局。

## 5.4 子力不足

检查双方所剩全部棋子数量，如果符合预设的子力不足够将死的情况则视为平局。值得注意的是，在残局中在同一种颜色格子中的主教无论多少应被视为一个。

## 5.5 未能判断的一种情况

本程序尚有一种平局无法判断，当双方剩余子力以兵为主，且双方所有兵均前后相邻无法移动，并封住其他棋子去路造成无法将死的情况，游戏平局，范例如图所示（出自Mednis, Edmar (1990), Practical Bishop Endings, Chess Enterprises, ISBN 0-945470-04-5），但是本情况无法通过一般规则判断，所以本程序尚存在这方面缺陷。

第4部分 详细设计说明书

1 引言

本部分介绍了程序的具体实现细节。

2 程序整体示意流程图

3 类、方法、变量功能概述

## 3.1 class Initialize

仅在程序启动时实例化一次，用以初始化程序程序所需的各种位棋盘和公共变量。

## 3.2 class Global

仅在程序启动时实例化一次，存放程序所需的公共变量。

## 3.2.1 static long mask[64]

索引为棋盘格子的坐标，数据内容为仅在该坐标有子时的位棋盘。

## 3.2.2 static long bitBoard[400][14]

存储记录棋局状态所需的14个位棋盘。为了保证悔棋与部分需要向前检索的操作正常进行，每步采用独立的棋盘记录。

## 3.2.3 static long tempBoard[14]

存放临时的棋局状态，用以在落子前判断着法的合法性。

## 3.2.4 static long kingAttack[64]

索引为棋盘格子的坐标，存放国王在该坐标能攻击到的格子的位棋盘。

## 3.2.5 static long rookAttack\_rank\_ally[64][256]

索引为棋盘格子的坐标，当前行棋子分布，存放堡垒在该坐标时，行内仅有友方棋子且按照索引的位置摆放是堡垒横向能攻击到的格子的位棋盘。

## 3.2.6 static long rookAttack\_rank\_enemy[64][256]

索引为棋盘格子的坐标，当前行棋子分布，存放堡垒在该坐标时，行内仅有敌方棋子且按照索引的位置摆放是堡垒横向能攻击到的格子的位棋盘。

## 3.2.7 static long rookAttack\_file\_ally[64][256]

索引为棋盘格子的坐标，当前列棋子分布，存放堡垒在该坐标时，行内仅有友方棋子且按照索引的位置摆放是堡垒纵向能攻击到的格子的位棋盘。

## 3.2.8 static long rookAttack\_file\_enemy[64][256]

索引为棋盘格子的坐标，当前列棋子分布，存放堡垒在该坐标时，行内仅有敌方棋子且按照索引的位置摆放是堡垒纵向能攻击到的格子的位棋盘。

## 3.2.9 static long knightAttack[64]

索引为棋盘格子的坐标，存放骑士在该坐标能攻击到的格子的位棋盘。

## 3.2.10 static boolean goodToGo

玩家是否已选择了要移动的棋子。

## 3.2.11 static boolean onGoing

棋局是否还在进行。

## 3.2.12 static long nextMove

临时存放当前所选棋子本轮所有合法移动位置位棋盘。

## 3.2.13 static currentX,currentY,currentKind

当前所选棋子坐标及种类。

## 3.2.14 static int promotion=4

仅在兵升变时使用，玩家所选升变的棋子类型。

## 3.2.15 static String record[400]

存放本局游戏的对局记录。

## 3.3 class Frame extends JFrame

## 3.3.1 public Frame()

本程序的窗口框架，同时包括了悔棋和新游戏按钮，以及带滚动条的对局记录显示文本框。

## 3.3.2 public void fresh()

刷新文本框内的对局记录。

## 3.4 class ChessBoard extends JPanel

## 3.4.1 public ChessBoard()

本程序棋盘部分的容器，同时包括了在每轮落子后的局面判断方法的引用。

## 3.4.2 public void paintComponent(Graphics g)

棋盘的绘制函数。

## 3.5 PromotionDialog extends JDialog

兵升变选择对话窗框架。

## 3.6 class PromotionPanel extends JPanel

兵升变选择对话框的具体内容。

## 3.7 class Range

包含每种不同棋子的不考虑被将军的合法落子位置的判定用方法。

## 3.8 class Move

包含全部普通或特殊棋子移动的数据修改方法。

## 3.9 class Situation

包含全部局面的判定方法，包括将军，无子可动，将死，以及各种条件下的平局。

第5部分 测试结果报告

1 引言

## 1.1 编写目的

　　本报告记录了本程序的测试过程和结果。

## 1.2 背景

A．测试需对程序的全部功能进行实验。

B．测试前需要完成相应模块的编写工作。

2 测试方法

黑盒测试：黑盒测试也称[功能测试](http://baike.baidu.com/view/651435.htm)，它是通过测试来检测每个功能是否都能正常使用。

3 测试过程

程序员在本程序的编写过程中对每个模块进行了逐一独立的黑盒测试，确保每个模块在极端情况下能够正常工作，逻辑正确。

程序编写完成后由数名用户进行了整体性黑盒测试。在数局模拟对战中均表现良好，部分局面的判断能力使用户惊讶。

参考文献

[1] 布克兰德.吴祖增,沙鹰 译.游戏编程中的人工智能技术[M]. 北京：清华大学出版社，2006年.第七章

[2] François Dominic Laramée. Chess Programming Series[OL]; http://www.cnblogs.com/wentingtu/archive/2012/05/03/2479921.html，2000年5月17日

[3] James F. Swafford. Rotated Bitboards [OL]. http://www.xqbase.com/computer/struct\_rotation.htm (译文)

[4] 孙卫琴. Java逍遥游记[M].北京：电子工业出版社，2010年9月1日

[5] Cay S. Horstmann, Gary Cornell. Java核心技术-基础知识[M].北京：机械工业出版社,2008

[6] 梁建全. 你必须知道的261个Java语言问题[M].北京：人民邮电出版社, 2009年11月1日