摘　　要

随着科技的进步，计算机已经普及每个家庭，而计算机游戏也随之越来越多，越来越丰富。网络游戏、单机游戏等休闲游戏让众多人喜欢，而斗地主也受广大群众的追捧。

本设计以一副牌规则为准，将整个游戏按照发牌,叫牌，拆牌，以及出牌的流程划分为4个部分，开始时由系统调用发牌函数每人随机分到17张牌，留存3张底牌，然后电脑AI根据自己手上牌值与牌型结合给定权值决定是否叫三张底牌当地主。一旦叫牌成功，剩下两玩家就是农民，联合“斗地主”。双方以是否出完牌来决定胜负。如果地主的20张牌都出完了，则地主一方胜；如果两个农民中任何一人手中的17张牌出完了，则农民一方胜出。

本论文主要讲述了单机版斗地主的游戏算法的设计，以Visual C++语言编译的一个应用程序。使用者可以简单方便的实现斗地主游戏。本程序界面友好，操作简单，比较实用。

本程序在详细设计时不断地修正和完善，经过测试阶段反复调试和验证，最终形成达到设计要求的可行系统。

**关键词:** 斗地主;纸牌; Visual C++;游戏核心算法

**Abstract**

With the progress of science and technology, computers have universal every family, and with more and more computer games, increasingly rich.The network game, LAN game and other casual games for many people like it, but Landlord also affected by the broad masses in both hands.

This design will be subject to rules of a deck of cards, the entire game in accordance with the licensing, call, cards, as well as the process is divided into four parts.Licensing function is called by the system at the beginning, then 17 cards are randomly assigned to each person and three cards are also retained. After that, the computer AI decides whether to call cards based on it's own cards value and the cards type which all have a particular weight.Once the call is successful, the remaining two players is farmers, jointing "Landlords". Whether the two sides finish cards determine the winner. If the landlord's 20 cards are finished, the landlord side win; If any one of the two farmers finishes 17 cards , the farmers win.

This paper tells the stand-alone Landlord game algorithm is designed to Visual c + + language compiled an application. Users can simple to realize Fight the Landlord game. This procedure friendly interface, the operation is simple, is more practical.

This procedure in the detailed design constantly correction and perfect when, after testing phase repeated commissioning and validation, and finally form the feasible system to meet the design requirements.

**Key Words:** Fight the landlord; Cards; Visual C++; Game core algorithm

**目 录**

1 绪论

[1.1 课题背景 1](#_Toc10130)

[1.2 研究现状 1](#_Toc29645)

[1.3 研究项目及意义 1](#_Toc26451)

[1.4 本文的结构安排 2](#_Toc24913)

[1.5 系统开发工具及其介绍 2](#_Toc7705)

[1.5.1 开发工具的选择 2](#_Toc11820)

[1.6 Visual C++编程特点 3](#_Toc5796)

[1.6.1 C++语言的发展 3](#_Toc3927)

[1.6.2 Visual C++ 6.0简介 3](#_Toc23129)

[1.6.3 Visual C++ 6.0的开发环境 4](#_Toc14257)

[2 斗地主基本知识 5](#_Toc2946)

[2.1 斗地主规则 5](#_Toc6069)

[2.2游戏流程框架 6](#_Toc15005)

[3 主要模块 6](#_Toc8209)

[3.1发牌 7](#_Toc22304)

[3.1.1 第一发牌方案 7](#_Toc17884)

[3.1.2第二发牌方案 7](#_Toc8028)

[3.1.3 发牌算法的代码实现 8](#_Toc15730)

[3.2 叫牌 10](#_Toc20574)

[3.2.1 叫牌判断 10](#_Toc18755)

[3.2.2 牌型估值 10](#_Toc7684)

[3.3 拆牌 14](#_Toc9247)

[3.3.1拆牌原因 14](#_Toc31910)

[3.3.2寻找拆牌方案 15](#_Toc2746)

[3.3.3 拆牌实例 22](#_Toc20727)

[3.4 出牌 22](#_Toc18049)

[3.4.1 出牌方法产生 22](#_Toc19)

[3.4.2 产生方法 23](#_Toc27644)

[3.4.3效率问题 24](#_Toc26835)

[3.4.4逐个产生和全部产生 24](#_Toc20811)

[3.4.5内存使用 25](#_Toc22745)

[4 结论及展望 25](#_Toc9603)

[致谢 27](#_Toc25335)

[参考文献 28](#_Toc6132)

1 绪论

## 1.1 课题背景

“斗地主”起源于湖北，据传是一位叫吴修全的年轻人，根据当地流行的扑克玩法“跑的快”改编的。由于其游戏规则简单、娱乐性强、老少皆宜，如今已风靡整个中国，并流行于互联网上。具体规则为：由三人玩一副牌，其中一方作为地主，另外两方联合作为农民向地主对抗。开始时每人随机分到17张牌，然后各玩家根据自己手上牌的好坏决定是否叫三张底牌当地主。一旦叫牌成功，剩下两玩家就是农民，联合“斗地主”。双方以是否出完牌来决定胜负。如果地主的20张牌都出完了，则地主一方胜；如果两个农民中任何一人手中的17张牌出完了，则农民一方胜出。

## 1.2 研究现状

随着电脑软硬件技术和互联网的飞速发展，电脑游戏也突飞猛进，成功游戏不断涌现。比较成熟的大型游戏主要有：反恐精英（CS）、星际争霸、魔兽争霸2、FIFA2003、帝国时代、FIFA2004等等。这时也出现了越来越多的网络小游戏，休闲类项目包括网络围棋、中国象棋、四国军棋、桥牌、麻将、拱猪、拖拉机、斗地主等。

各式各样的游戏给我们的生活带来了无限的乐趣，但同时也出现了一些问题。虽然计算机电脑已经普及绝大多数家庭，但有些偏远的地方却无法正常的接受到网络信号，这就给一部分人带来了有电脑不能上网休闲的烦恼。因此，一些单机游戏更受到了人们的追捧。本设计的目的是用简单的C语言设计出一个斗地主的小游戏，使用者只要运行该程序，就可以轻松的玩游戏了。

## 1.3 研究项目及意义

“斗地主”属于不完全知识博弈，意思是指参与的三方一开始都不清楚对手手中有什么牌。作为全国广泛流传且具有中国特色的一个牌类游戏，被当做计算机技术的研究对象却是不常见的。本设计旨在分析“斗地主”AI程序的各个必要部分，以“斗地主”的游戏规则为基础，给出各部分的初级探讨,给出一个较有效的“斗地主”AI程序。重点研究拆牌算法，并做出相应的分析、设计和实现。

## 1.4 本文的结构安排

结合引言部分，参看各节的标题可以知道，本论文是由引言、本论、结论三部分组成。而本轮部分是沿着实际游戏流程一步步论述的。这样的布局，更容易被接受。

# 1.5 系统开发工具及其介绍

# 1.5.1 开发工具的选择

古语云：工欲善其事，必先利其器。利用一系列高效的、具有良好可视化的编程工具去开发各种数据库软件，从而达到事半功倍的效果，所以开发工具的选择显得非常重要。 随着计算机技术不断发展，各种编程工具也随着发展，但是现在市面上的数据库编程工具门类众多，优良不齐，比如VB、PB、DELPHI、C++ Builder、VC、JAVA等。我们先来了解一下各类开发工具。

VB，全称Visual Basic，它是以Basic语言作为其基本语言的一种可视化编程工具。VB开发效率高，代码执行效率一般，但是入门和学习速度快，有较好的学习氛围和帮助书籍和帮助文档。但是随着微软的.NET技术的推广，VB将会逐渐退出历史舞台。

PB，全称PowerBuilder，是开发MIS系统和各类数据库跨平台的首选，使用简单，容易学习，容易掌握，在代码执行效率上也有相当出色的表现。但是PB的高级编程书籍的缺乏和基础语言PowerScript的普及程度对PB的明天有一定的限制。

DELPHI/ C++Builder它们都是基于VCL库的可视化开发工具，它们在组件技术的支持、数据库支持、系统底层开发支持、网络开发支持、面向对象特性等各方面都有相当不错的表现，并且学习使用较为容易，充分提现了所见即所得的可视化开发方法，开发效率高。，它们所作的最大不足之处就是他们的帮助系统在众多的编程工具中是属于比较差的。

JAVA工具，目前比较出名的是Borland出的JBuilder和IBM出的Visual Age for Java,两种工具。适用于开发除了系统软件、驱动程序、高性能实时系统、大规模图像处理以外所有的应用。在一般的管理信息系统中和一般的数据库开发中，很少有人会选择JAVA。但是JAVA的强大网络功能和真正的面向对象特性决定了JAVA的未来，JAVA将在一个较为长期的时段里可以与微软家族抗衡。

Visual C++，是基于MFC库的可视化的开发工具，从总体上说它是一个功能强大但是不便使用的一种工具。它在网络开发和多媒体开发都具有不俗的表现，帮助系统也做得非常不错（Microsoft 在细节方面的处理往往都让人觉得亲切），从功能上讲除了跨平台应用外什么都可开发；从开发效率角度讲只局限于开发windows系统应用、可重用组件及驱动程序。又因为有微软的支持，其自身的基础语言的普及程度高，以及其代码的执行效率高等特性、VC自带强大帮助文档和大量优质教材，VC在开发工具中始终可以稳住阵角。

## 1.6 Visual C++编程特点

### 1.6.1 C++语言的发展

C++语言发展大概可以分为三个阶段：

第一阶段从80年代到1995年，C++语言基本上是传统类型上的面向对象语言，并且凭借着接近C语言的效率，在工业界使用的开发语言中占据了相当大份额；

　　第二阶段从1995年到2000年，这一阶段由于标准模板库(STL)和后来的Boost等程序库的出现，泛型程序设计在C++中占据了越来越多的比重性。当然，同时由于Java、C#等语言的出现和硬件价格的大规模下降，C++受到了一定的冲击；

　　第三阶段从2000年至今，由于以Loki、MPL等程序库为代表的产生式编程和模板元编程的出现，C++出现了发展历史上又一个新的高峰，这些新技术的出现以及和原有技术的融合，使C++已经成为当今主流程序设计语言中最复杂的一员。

C++是由C语言发展而来，是面向对象程序设计语言之一。C++语言是一种应用较广的面向对象的程序设计语言，使用它可以实现面向对象的程序设计。面向对象的设计与面向过程的设计是有很大区别的，面向对象的程序设计是在面向过程的程序设计的基础上一个质的飞跃。

### 1.6.2 Visual C++ 6.0简介

Visual C++是Microsoft公司的Visual Studio开发工具箱中的一个C++程序开发包。VisualStudio提供了一整套开发Internet和Windows应用程序的工具，包括VisualC++， Visual Basic，Visual Foxpro, Visual InterDev， Visual J++以及其他辅助工具，如代码管理工具Visual SourceSafe和联机帮助系统MSDN。Visual C++包中除包括C++编译器外，还包括所有的库、例子和为创建Windows应用程序所需要的文档。

Visual C++作为一个主流的开发平台一直深受编程爱好者的喜爱，但是很多人却对它的入门感到难于上青天，究其原因主要是大家对他错误的认识造成的，严格的来说Visual C++不是门语言，虽然它和Visual C++之间有密切的关系，如果形象点比喻的话，可以Visual C++看作为一种“工业标准”，而V Visual C++则是某种操作系统平台下的“厂商标准”，而“厂商标准”是在遵循”工业标准”的前提下扩展而来的。  
 从最早期的1.0版本，发展到最新的6.0版本，Visual C++已经有了很大的变化，在界面、功能、库支持方面都有许多的增强。最新的6.0版本在编译器、MFC类库、编辑器以及联机帮助系统等方面都比以前的版本做了较大改进。

Visual C++的优点是界面简洁，占用资源少，操作方便。

### 1.6.3 Visual C++ 6.0的开发环境

Visual C++ 6.0是一个Windows应用程序，启动后显示一个标准窗口，包括窗口框架、标题栏、菜单栏、工具栏、项目工作区、状态栏等，在这个窗口中可完成程序创建工程、编辑编译文件、资源编辑、程序调试、链接生成可执行文件等工作，是一个集成化开发环境。

应用程序的大部分命令是通过菜单发出的，在使用一个开发工具之前，了解其菜单功能是非常重要的。Visual C++ 6.0的主菜单包含File、Edit、View、Insert、Projec、Build、Tools、Windows、Help等项。



图1.1 Visual C++ 6.0的标准窗口

File菜单：New选项的新建文件中可以选择创建新的 C++源文件，头文件等

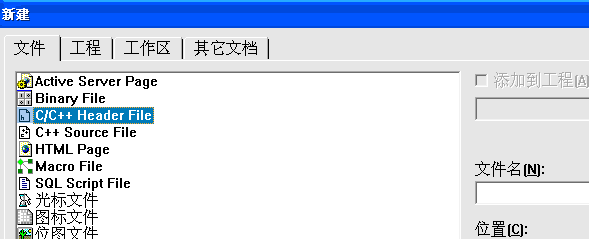


图1.2 创建新的头文件和源文件界面

Edit菜单：包含对源文件进行编辑的命令，有许多与Word编辑命令相同，另外还包含一些针对方便源文件录入的辅助工具。

Build菜单：对程序代码进行调试和生成可执行程序。

# 

# 2 斗地主基本知识

## 2.1 斗地主规则

1. 发牌

一副牌 54 张，一人 17 张，留 3 张做底牌，在确定地主之前玩家不能看底牌。

1. 叫牌

叫牌按出牌的顺序轮流进行，叫牌时可以选择 “叫地主 ” 、“ 不叫 ” 。如果有玩家选择 “叫地主 ” 则立即结束叫牌，该玩家为地主；如果都“不叫”，则重新发牌，重新叫牌，直到有人“叫地主”为止 。

1. 牌型

“斗地主”的牌型丰富多彩，根据QQ游戏提供的说明，牌型定义如下：

火箭：即双王（大王和小王），最大的牌；

炸弹：四张同数值牌（如四个7）；

单牌：单个牌（如红桃5）；

对牌：数值相同的两张牌（如梅花4+方块4）；

三张牌：数值相同的三张牌（如三个J）；

三带一：数值相同的三张牌+一张单牌或一对牌，如：3 3 3+6或4 4 4+9 9；

单顺：五张或更多的连续单牌（如：4 5 6 7 8或7 8 9 10 J Q K），不包括2点和双王；

双顺：三对或更多的连续对牌（如：3 3 4 4 5 5、7 7 8 8 9 9 10 10 J J），不包括2点和双王；

三顺：二个或更多的连续三张牌（如：3 3 3 4 4 4、5 5 5 6 6 6 7 7 7 8 8 8），不包括2点和双王；

飞机带翅膀：三顺＋同数量的单牌（或同数量的对牌），如：4 4 4 5 5 5+7 9或33 3 4 4 4 5 5 5+7 7 9 9 J J；

四带（一）二：四张牌＋（一）两手牌（四带（一）二不是炸弹），如：5 5 5 5＋3＋8或4 4 4 4+5 5＋7 7。

1. 牌型大小

各牌型的大小定义如下：

火箭最大，可以打任意其它的牌；

炸弹比火箭小，比其它牌大，都是炸弹时按牌的分值比大小；

除火箭和炸弹外，其它牌必须牌型相同且总张数相同才能比大小；

单牌按分值比大小，依次是大王>小王>2>A>K>Q>J>10>9>8>7>6>5>4>3，不分花色；

对牌、三张牌都按分值比大小；

顺牌按最大的一张牌的分值来比大小；

飞机带翅膀和四带二按其中的三顺和四张部分来比，带的牌不影响大小。

## 2.2游戏流程框架



图2.1 游戏流程框架图

# 3 主要模块

## 3.1发牌

### 3.1.1 第一发牌方案

由于调试和实际应用需要，对于既定牌局的出牌方案，需要手动输入。然而手动输入手中持牌是远远不够的。如果是开局，除了输入手中持牌，在程序处理方面，应该将剩下的牌随机分配且均匀分配给另外两家，最终剩余三张底牌。如果是已经叫了地主，剩余的三张底牌也是公开的，仅有34张需要随机分配给剩余两家。如果是进行到一半，需要将牌池中的牌型以及出牌玩家和出牌顺序记录下来，供程序分析，调用算牌程序，从而推导出优化出牌方案。

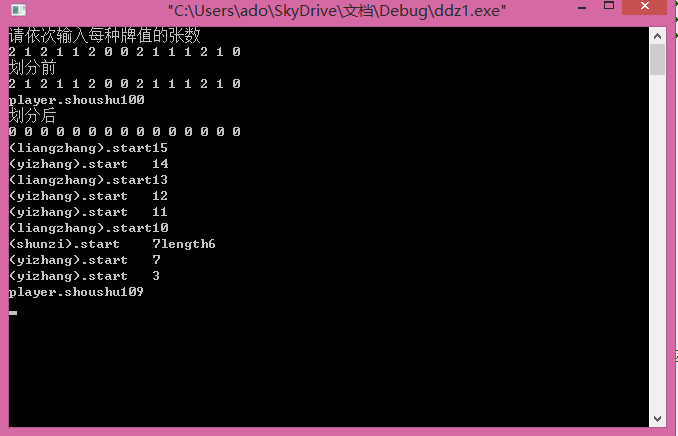


图3.1 手动输入牌值运行界面

### 3.1.2第二发牌方案

由于调试和权值确定的需要，对于不确定的牌局，需要机器随机分配。对于长篇代码的修改，为了省时并且测试意想不到的数据可能效果，通常是一个循环，连续多次不断分配，直至强行退出DOS程序运行界面。

对于权值确定方面，主要功能是统计不通牌型和牌值出现的概率。将这些硬数据作为权值确定的参数基础。

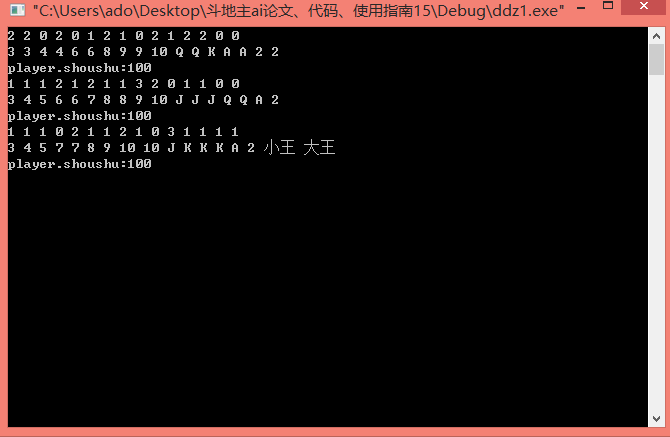


图3.2 系统自动发牌界面

### 3.1.3 发牌算法的代码实现

int GetCards(PLAYER player[], int hide\_cards[])//发牌算法

{

int numCards[16]={0},i,j,k=0;

bool fenpei=false;

for(i=1;i<=15;i++)

{

if(i==15||i==14)

numCards[i]=1;

else numCards[i]=4;//到此时numCard数组里面是0 4 4 4...4 1 1

for(j=0;j<3;j++)

player[j].numCards[i]=0;//到此每个player的numcard里面都是0 0 0 ...0 0 0

}

srand((unsigned int)time(NULL));

//发牌过程

for(i=0;i<17;i++)//发17轮牌 3个do-while为一轮发牌

{

for(k=0;k<3;k++)

{

do{

j=rand()%15+1;//随机发牌

if(numCards[j]>0)

{

player[k].numCards[j]++;

numCards[j]--;

fenpei=true;

}

}while(!fenpei);//如果fenpei为false说明此次分配未成功，继续循环

fenpei=false;

}

}

for(i=1,j=0;i<=15;i++){//剩余牌提出做底牌

while(numCards[i]){

hide\_cards[j]=i;

j++;

numCards[i]--;}

}

return 1;

}

### 3.2 叫牌

### 3.2.1 是否叫牌

初始模型是直接根据每种牌型的权值给出的，然而仅仅用权值是无法刻画出真实的是否叫牌的决断的，有时甚至还会产生笑话。由此，给出除了权值的另外一种刻画方式：手数。手数是指将手中的牌按规则出完所需要轮过的次数。

然而，不同的拆牌方式将导致手数的不同。由于牌型的变化极其复杂，所以从众多的拆牌方式中选出手数最少的一种是比较困难的，这是本文源码实例的重点，也是本章的主要内容。

### 3.2.2 牌型估值

1，牌型的价值评估

牌型的价值评估，简单地说就是评估牌型的可以算一手味的权值。一手味是指可以在一轮牌中压倒另外两家。初步的方法是直接给出各种牌型的值，牌型的权值KindValue=（CardNumber\*CardValue），其中，KindValue是指牌型的权值，CardNumber指该牌型的牌所具有的牌的张数，CardValue指单张时候牌的权值。

仅仅这样简单的评估，显然是不符合实际需要的。原因是对于牌型的权值评估，往往还跟其它因素有关。例如，当三方手上的牌较少时，长顺子的权值会更高。这就需要动态地设计权值了。

2．牌型关系的评估

牌型间的关系也是估值的重要内容之一，可以将某个牌型很有可能被敌方压住看成是一个不利因素。例如手中持有34567的顺子，但是没有顺子回手，这时可以根据该顺子的权值与恶劣的牌型关系导致的负值之和来刻画。

牌型关系的评估应该考虑到该谁出牌和地主农民的座次问题。如果自己是农民，且是地主的下家，恶劣的牌型关系比较轻；如果是地主的上家或者是地主，牌型关系更加恶劣。

3．编写统计程序，实时更改相应参数

针对每局的拆牌分析和对剩余两家的算牌分析，分别存入拆牌牌型数目统计表和预测牌型统计表。前者用于确定权值的硬指标参数。例如：在没有复杂牌型，且不考虑王和2的情况下，三无且不考虑王、2和三张底牌，有每把各牌型出现次数如下：一张的：4.145，两张的：3.245，三张的：1.135，四张的：0.133。而4.145×1 +3.245×2 +1.135×3 +0.133×4 =4.145 +6.590 +3.405 +0.532=14.672≈15(期望值)，绝对误差0.338，相对误差0.338/15≈2.253%。这四个参数可以用来描述这四种牌型的相互关系。而牌型内部关系可以根据牌值的不同和方差来确定。

由于根据参考文献[7]的随机方法来预测是不现实的，因为变数太多，情况极其复杂，在现有的水平内无法实现加法和乘法、条件概率以及先验概率的多重严格的数学分析。故可以通过预测牌型统计表的后验法来统计正确率，并根据正确率的不同调整预测参数。例如：在A出了777+5的时候，参看自己手上没有7，这样的情形下，原本猜测C方有一个7的可信度为1。可是在经过几局的后验之后，可能出现A有含7的顺子，而C没有7的情形。从而可根据这种情形在总局数中的比例，适当调整这个可信度。

bool Jiaopai(PLAYER &player)

{

//拆分之后，复原player.numCards数组的值，以便后续分析

vector<PAIXING>::iterator it;

//七种牌型的顺序：一张、两张、三张、四张、顺子、连子、飞机

double fangcha[7]={0},middle[7]={0};//方差：用来描述照应关系，越大越好；中值

double quan\_sum=0,quan\_dandu=0,quan[7]={0};//总权，七种牌型的分权{不包括王、2}

int i,num[7]={0};//num用来记录各种对应牌型的数目

for(it=player.paixing.begin();it!=player.paixing.end();it++)

{

if((\*it).kind==1)//单张

{

player.numCards[(\*it).start]++;

middle[0]+=(\*it).start;

num[0]++;

}

if((\*it).kind==2)//对子

{

player.numCards[(\*it).start]+=2;

middle[1]+=(\*it).start;

num[1]++;

}

if((\*it).kind==3)//3张

{

player.numCards[(\*it).start]+=3;

middle[2]+=(\*it).start;

num[2]++;

}

if((\*it).kind==4)//炸弹

{

player.numCards[(\*it).start]+=4;

middle[3]+=(\*it).start;

num[3]++;

}

//顺子连子飞机的权值确定方法：由各个相应牌型的长度和数目决定

if((\*it).kind==5)//单顺

{

for(i=(\*it).start;i>=(\*it).start-(\*it).length+1;i--)

player.numCards[i]++;

num[4]++;

}

if((\*it).kind==6)//双顺

{

for(i=(\*it).start;i>=(\*it).start-(\*it).length+1;i--)

player.numCards[i]+=2;

num[5]++;

}

if((\*it).kind==7)//三顺

{

for(i=(\*it).start;i>=(\*it).start-(\*it).length+1;i--)

player.numCards[i]+=3;

num[6]++;

}

}

ShowPlayer(player);

for(i=0;i<7;i++)

{

cout<<"quan[";

switch(i)

{

case 0:cout<<"一张";break;

case 1:cout<<"两张";break;

case 2:cout<<"三张";break;

case 3:cout<<"四张";break;

case 4:cout<<"顺子";break;

case 5:cout<<"连子";break;

case 6:cout<<"飞机";break;

default:break;

}

cout<<"]========"<<setprecision(3)<<quan[i]/quan\_sum<<endl;

}

cout<<"quan\_dandu========"<<quan\_dandu/quan\_sum<<endl;

cout<<"quan\_sum=========="<<quan\_sum<<endl;

getch();

if(quan\_sum>QUAN\_GATE)

return true;

return false;

}

4.叫牌实例

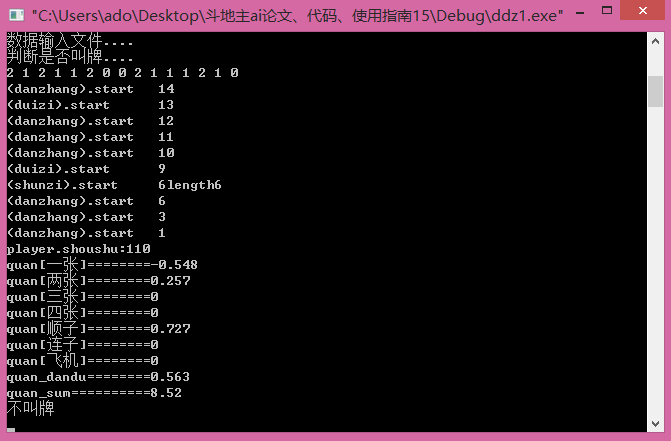


图3.3 叫牌实例图

# 3.3 拆牌

## 3.3.1什么叫拆牌，为什么要拆牌

拆牌是指依据游戏规则对手中持牌的合理牌值的组合。例如：

实例一：1 2 3 4 5

3 4 5 6 7 8 9

3 4 5

的最优拆分不是：（5 6 7 8 9）+（1+3×3+4×3+2）+ 5×2，共3手

也不是（1+2+6+3×3+4×3+5×3）+7+8+9，共4手

更不是（1 2 3 4 5）+（3 4 5 6 7 8 9）+（3+4+5），共5手

而是（3×2+4×2+5×2）+（1 2 3 4 5 6 7 8 9），共2手

实例二：1 2 3 4 5

3 4 5 6 7 8 9

3 4

的最优拆分不是（1 2 3 4 5）+（3 4 5 6 7 8 9）+（3+4），共4手

而是（1+2+3×3+4×3）+（5 6 7 8 9）+5，共3手

以上两个实例直观地给出了拆牌的原因。不仅是是否叫底牌的牌面分析中主要就是拆牌分析，而且出牌方案的产生也很大程度上依赖于拆牌分析。拆牌方案是否合理往往决定了胜败。

## 3.3.2寻找拆牌方案

为了解决拆牌这个难题，可以将牌按照牌型划分为不交叉的部分和交叉的部分。首先将顺子、连子、飞机定义为复杂牌型。如果某种牌值既可以出现在复杂牌型A中，又可出现在复杂牌型B中，则属于交叉部分；否则属于非交叉部分。

非交叉部分可以通过直接根据手数评价。

1.简单牌型（包括单张、两张、三张、四张中不与复杂牌型相交的部分）

直接压入相应牌型向量中。

2.下面给出最小元顺子算法：

（1）首先从该段牌值范围内选出最小元顺子。所谓最小元顺子指的是从该段牌值范围选出的五张连续牌且组成顺子之后对这部分的手数负面影响最小的一个。如果这样的顺子有几个，选择最大的一个。

（2）左右延伸：仍然根据手数最小原则，比较往左延伸和不往左延伸任意张牌的手数，选择最小手数的方法。往右延伸的方案同理。

3 一个只针对不含交叉牌型的顺子部分的完整算法：

（1）直接利用最小元顺子方法给出手数评价

（2）对于有二连对且可拆分成一套可评价顺子和一套待评价顺子的，前半部分可评价的顺子直接利用最小元顺子方法，而后半部分调用本算法直接递归执行， 直至没有该类二连拆分的情形。

举一个实例：3 444 5 6 6 7 7 8 9 10，根据手数最少原则，应该划分成3 4 5 6 7+6 7 8 9 10+4 4

（3）对牌型范围内剩余的零散牌再次利用。

4. 对于不含交叉部分的特殊牌型范围，即连子和飞机与顺子处理类似，简单之处在于不用考虑重叠。

5.对交叉牌型的顺子部分的算法：

（1）检测交叉牌型中包含哪几类牌型；

（2）首先分别根据最小元方法，取出相应牌型；

（3）剩下的部分再利用总拆牌函数得出手数值，比较手数，选择最优的拆牌方案。

这整体上是一个大的深搜过程。

一个“斗地主”程序的拆牌框架如下：

12345五步总结为初步划分函数；5总结为交叉划分函数。按需递归调用。

int Huafen(PLAYER &player)

{

int i,sum\_cards=0;

for(i=1;i<13;i++)//12种牌值 应该是3到1

{

if(player.numCards[i])

sum\_cards+=player.numCards[i];//手中总牌数

}

if(sum\_cards)//只要手中有牌

Huafen\_Chubu(player);//提出不相关的一张、两张、三张的牌型

sum\_cards=0;

for(i=1;i<13;i++)

{

if(player.numCards[i])

sum\_cards+=player.numCards[i];//初步划分后剩余牌数

}

if(sum\_cards)

Huafen\_Jiaocha(player);//交叉部分的划分

return 0;

}



图3.4 “斗地主”程序的拆牌框架示意图

针对非交叉部分的初步划分

PLAYER Huafen\_Chubu(PLAYER &player)//将不关系到顺子、连子、飞机的牌全部先压入牌型，并将player.numcards数组做相应改变

{

int flag[16]={0};//其实flag[0]是无用的，为的是序号一致

int i,j,k;

//找顺子

vector<PAIXING>shunzi\_temp\_vec;

PAIXING shunzi\_temp;

shunzi\_temp.length=0;shunzi\_temp.start=0;shunzi\_temp.shoushu=0;

for(i=12;i>4;i--)//顺子的开头，从12开始依次向下遍历。

{

for(j=0;j<=i;j++)//j+1记录顺子的长度

{

if(player.numCards[i-j]>=1)//如果顺子未出现终断

{

if(j<4) continue;//长度不够，说明未成顺子，继续

else shunzi\_temp.start=i;//成了顺子，记录开始位置

}

else//出现终断

{

if(shunzi\_temp.start)//如果记录了开始位置，成了顺子

{

shunzi\_temp.length=j;

i-=j;//以保证不会重复算

break;

}

else break;//未成顺子，退出内层循环

}

}

if(shunzi\_temp.start&&shunzi\_temp.length)

{

for(k=shunzi\_temp.start;k>shunzi\_temp.start-shunzi\_temp.length;k--)

flag[k]=1;//表示牌值为k的已包含在顺子中

shunzi\_temp\_vec.push\_back(shunzi\_temp);

shunzi\_temp.start=0;

shunzi\_temp.length=0;//必须重新回到0，否则可能出错

}

}

//找连子部分略去

//找飞机部分略去

}

针对交叉部分的

交叉部分的划分方案：

取炸弹，然后对剩余的牌调用原函数检查手数

取飞机，然后对剩余的牌调用原函数检查手数

取连子，然后对剩余的牌调用原函数检查手数

直接按顺子拆分，作为保底方案

四者相比较，选择最优方案，是深搜的例子

int Huafen\_Jiaocha(PLAYER &player)

{

//找出所有的交叉段

vector<PAIXING>jiaocha;

int i;

PAIXING jiaocha\_temp;

jiaocha\_temp.start=0,jiaocha\_temp.length=0;

for(i=15;i>0;i--)

{

if(jiaocha\_temp.start==0&&player.numCards[i])

jiaocha\_temp.start=i;

if(jiaocha\_temp.start&&(player.numCards[i]==0||i==1))

jiaocha\_temp.length=jiaocha\_temp.start-i;

if(jiaocha\_temp.start&&jiaocha\_temp.length)

{

jiaocha.push\_back(jiaocha\_temp);

jiaocha\_temp.start=0;jiaocha\_temp.length=0;

}

}

vector<PAIXING>::iterator it;

for(it=jiaocha.begin();it!=jiaocha.end();it++)

{

//cout<<(\*it).start<<(\*it).length;

PLAYER copy\_player[5];

for(i=4;i>=0;i--)

FuzhiPlayer(copy\_player[i],player);

bool flag[5];

for(i=0;i<5;i++)

flag[i]=false;

//取一个炸弹

for(i=(\*it).start;i>=(\*it).start-(\*it).length+1;i--)

{

if(copy\_player[0].numCards[i]==4)

{

copy\_player[0].shoushu+=zhadan\_shoushu;

copy\_player[0].numCards[i]=0;

PAIXING sizhang\_temp;

sizhang\_temp.start=i;

sizhang\_temp.kind=4;

copy\_player[0].paixing.push\_back(sizhang\_temp);

flag[0]=true;

Huafen(copy\_player[0]);

break;

}

}

//取一个飞机略去

//取一个连子略去

/\*对于copy\_player[3]的顺子拆分，与之前顺子拆分不同，是直接从头开始，一直循环找5张顺，直到没有5张顺为止，

剩下的零牌尽量凑到原存在的顺子上，然后计算手数\*/

//取一个顺子

return 1;

}

### 3.3.3 拆牌实例

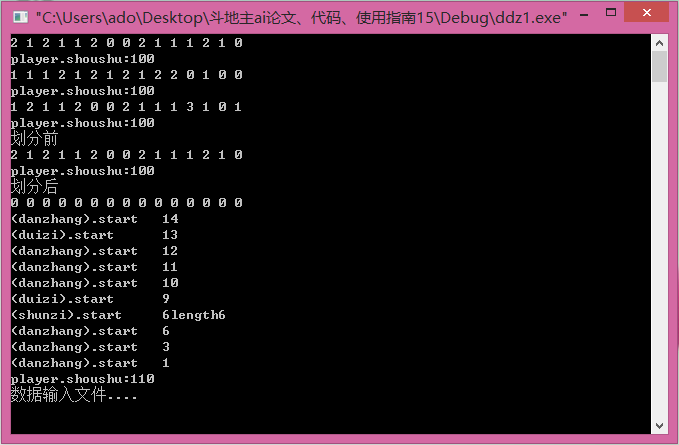


图3.5 拆牌实例

## 3.4 出牌

## 3.4.1 出牌方法产生

博弈的规则决定了哪些出牌方法是合法的。对有的游戏来说，这很简单，比如五子棋，任何空白的位置都是合法的落子点。但对于”斗地主”来说，有三带一可以打三带一而不能打三带二、炸弹可以炸除炸弹之外的所有牌型等一系列复杂的规则。

出牌方法是博弈程序中一个相当复杂而且耗费运算时间的方面。不过，通过良好的数据结构，可以显著地提高生成的速度。

出牌方法的产生，依赖于几个条件决策：座次和身份、算牌分析结果、出牌规则和自身的拆牌倾向。而出牌规则的满足属于基本条件，座次和身份直接影响着各部分的权值分配，算牌分析则是通过牌池中已有的牌和剩余两家的出牌猜测各自剩下的牌。拆牌倾向性直接由第五节中的拆牌方案，给出出牌期望。综合以上条件，从而有针对性地给出合理的出牌方案。

## 3.4.2 如何产生

由于叫牌比较简单且和出牌方案有很多重叠部分，故不单独列为一节。出法产生是指将一个牌局所有可能的出法罗列出来的那一部分程序。也就是用来告诉其它部分下一步可以往哪里走的模块。各种牌类游戏随规则的不同，出法产生的复杂度也有很大的区别。例如，升级的牌型比较单一，除去同色连对和同色AKK和AAK之外，只有单张、两张；而在“斗地主”中，需要仔细判断。比如四张可以压三张，三张可以压两张，顺子必须同数目等等。

对于“斗地主”程序，首先分析牌局，如果自己是该轮首家出牌者，任何出牌方式都是合乎规则的；如果不是，根据上一家出牌的牌型，可给出合乎规则的出法。

一个“斗地主”程序的出法产生代码如下：

void GenerateAll (PAIXING,VALUE)

{

Case Shunzi:GenShunzi();

Case Lianzi:Lianzi();

Case Feiji:GenFeiji();

Case Danzhang:GenDanzhang();

Case Liangzhang:GenLiangzhang();

Case Sanzhang:GenSanzhang();//三不带

Case Sanzhang\_Danzhang:GenSanzhang\_Danzhang();

Case Sanzhang\_Liangzhang:GenSanzhang\_Liangzhang();

Case Sizhang:GenSizhang();

Case Sizhang\_Danzhang\_Danzhang:GenSizhang\_Danzhang\_Danzhang();

Case Sizhang\_Danzhang\_Liangzhang:GenSizhang\_Danzhang\_Liangzhang();

//四张带一张和四张带两个一张

Case Sizhang\_Liangzhang\_Danzhang:GenSizhang\_Liangzhang\_Danzhang();

Case Sizhang\_Liangzhang\_Liangzhang:GenSizhang\_Liangzhang\_Liangzhang();//四张带一对和四张带两对

}

//产生顺子的出法

void GenShunzi(Length)

{

查找自家顺子可达长度Length\_max

If Length\_max<Length 不可顺子压；

Else 可用顺子压，方法有Length\_max-Length+1种

检查是否有四张或者对王，表明是否可用炸弹压

选择一种可选出法中最好的方法

}

## 3.4.3效率问题

出法通常伴随着搜索进行，并且频繁调用。但是像“斗地主”这样有复杂规则的不完全博弈游戏，出法往往导致大量繁琐的判断。这在一定程度上形成了程序的性能瓶颈。

为了提高出法产生的速度，目前有很多种优化方法。由于优化方法很多情况下与游戏具体规则相关，这里只说明具有普遍意义的常用方法。

在“斗地主”中，每一种出法各有不同，这样的情形导致了较为复杂的判断。但是我们可以将第一次搜索过的各种最大值保留在数据库中，要用的时候只需要做简单判断。

还是以顺子为例。每次产生顺子走法时，检查可产生顺子的最大长度是否大于需要的长度，可以则选择一种出法后修改这个最大长度。与前面的每一次都查找自家顺子的可达长度相比，少了一步每一次都从头检查。这个可达长度可以作为静态变量保存，从而提高效率。

## 3.4.4逐个产生和全部产生

在进行出法产生的时候，往往伴随着搜索进行。对于一个牌局的所有直接后继，可以有两种选择：一次产生一种走法然后搜索之；或者一次产生其所有走法然后搜索之。由于存在着剪枝算法，对一个牌局的某一出法搜索之后往往可能不再需要搜索其它的后继。这就是说：可能不用产生全部出法就能够完成搜索。一次产生一种出法看起来似乎更有效率。但是，由于剪枝算法的剪枝效率很大程度上依赖于结点的排列顺序，往往一次产生所有结点，然后以某种方法调整其排列顺序会使搜索效率大大提高。

所以，在实际使用中，绝大部分程序都是一次产生一个局面的全部走法，然后调整其搜索顺序。

## 3.4.5内存使用

在产生走法时，通常设计人员会将走法队列置于一段预先申请的内存当中（通常是全局变量，或者是在程序启动时申请的整块内存），以避免频繁的申请动态内存而引起大量时间的耗费。那么，这样内存要多大才够用呢？

“斗地主”中，如果自己是该轮的首家，最大出法为：顺子有：（12-5+1）+（12-6+1）+（12-7+1）+……+1=36种，单张：15种（只有15种牌值），两张：13种，连对：最多10连对，与顺子类似，有36种；三张：13种，三张带一张：13×14（包括带一张王）种，三张带两张：13×12（不包括带一对王）；四张：13种，四张带一个一张：13×14，四张带两个一张：13×（2\*12+12\*11），四张带一个两张：13×12，四张带两个两张：13×12×11。

飞机的情形就更复杂了，大约是顺子和四张的各种情况的乘积，另外加一种对王的情形。在不考虑牌数限制的情况下，总共的出牌方法不下于一万种。

由于出法的复杂多样，如果直接存储各种出法，将导致效率奇低。分析出法复杂多样的原因是由于各种牌型的组合，所以可先将各种可能的组成部分分别存储到一个小型数据库，到需要的时候进行组合运算动态组合，产生各种各样的出法，从而提高效率。

# 4 结论及展望

本文的主要完成以下工作：

1.对整个“斗地主”流程的全盘探讨

2.拆牌分析的重点讨论

3.“手数”作为主要评价标准的确定和应用

4.最小元搜索方式的应用

5.给出了一个较为可行的“斗地主”AI方案，弥补了当前市面上对于“斗地主”这一风靡全国的游戏的AI方案的空白，对后续此方面的研究有一定的指导意义。

本文还有不少地方需要进一步探索和完善

1.对人工智能的进阶搜索算法，例如α-β搜索、渴望搜索、哈希置换表、历史启发等[8]的应用和“斗地主”相结合。

2;.出牌和统计算法代码的细化。

3.水平效应及代码优化的探讨。

# 致谢

为期三个多月的毕业设计即将结束，也预示着四年的大学学习生活将要画上一个圆满的句号。刚拿到这个课题的时候，总觉得课题比较难。可是今天回顾了一下一个学期的努力，却欣然发现，在这个学期里学到的东西很多很多。无论从课题的选定还是程序的编写，每一个环节、每一个都凝聚着我的汗水，也正因为如此，通过这一段时间的实际操作,使得我对本专业所学的课程有了更加深刻的理解,把书本上死的东西比较好的掌握了。同时对于专业的特点,性质有了进一步的认识。

在此次毕业设计中，我要感谢那些在设计期间帮助过我的老师和同学，尤其要感谢的是我的指导老师王少威王老师，在设计期间，王老师帮助我分析课题要求，收集文献资料，指导操作方法，并对我所做的课题提出了有效的改进方案，他以渊博的学识和丰富的实践经验给我的毕业设计提供了很多宝贵的指导性意见和建议，让我顺利的完成了自己的毕业设计。

# 参考文献

[1] 张志刚.计算机桥牌程序的设计与实现.硕士论文，辽宁科技大学，2008

[2] 林建伟.张杰.博弈树搜索技术在牌类网络游戏中的应用.汕头大学硕士学位 论文.2009：22～23

[3] 王小春.PC游戏编程（人机博弈），12～14，180～190

[4] S.H.Fuller,J.G.Gaschng,J.J.Gllogly.An analysis of the alpha-beta p runing algorithm[D]，Pittsburg:Carnegie-Mellon University,1973.

[5] D.E.Knuth,R.W.Moore.An analysis of alpha-beta p runing[J].Artificial Intelligence,1975

[6] R.Korf.Iterative deepening:an optimal admissible tree search [J]. Artificial Intelligence,1985

[7] George F Luger.Artificial Intelligence Structures and Strategies for Complex Problem Solving Fifth Edition，120～137

[8] 王镌.博弈树搜索的算法改进.福建商业高等专科学校计算机.2004

[9] 王小春.PC游戏编程（人机博弈），12～14，180～190

[10] 严蔚敏、吴伟民.数据结构(C语言版) [M],北京：清华大学出版社.1997： P109-243

[11] 葛垚.Visual C++ MFC棋牌类游戏编程实例.人民邮电出版社，2008.7

[12] 万红.程序设计教程（C/C++版）.清华大学出版社，2005.6

[13] 曹静.C++面向对象程序设计.中国水利水电出版社，2006.8

[14] 徐孝凯.C++面向对象程序设计/VC++6.0运行环境.天津大学出版社，2009.1

[15] 徐青.VC.NET游戏开发技术.北京交通大学出版社，2008.8

[16] 谭浩强.C++程序设计.清华大学出版社，2004.6

[17] 沈大林.C++游戏设计案例教程.电子工业出版社，2009.5

[18] 范辉.Visual C++ 6.0程序设计简明教程.高等教育出版社，2001.7

[19] 吴玺玺.游戏设计入门.重庆大学出版社，2005.6

[20] 王鹏杰.董西广.游戏设计基础.机械工业出版社，2009.1

[21] Gordon Hogenson.C++/Cli The Visual C++ Language For.Net [M].Wiley India Pvt.Ltd，2007.

[22] Mark DeLoura.Best of Game Programming Gems. Mark Charles River Media，

2008.6

[23] John E.Swanke.Visual C++ MFC Programming by John E. Swanke R&D Books，1999.1