

C++Builder 程序设计范例

——中国象棋

胡 达 著

清 华 大 学 出 版 社

(京)新登字 158 号

内 容 简 介

C++Builder 5.0 是 Borland/Inprise 公司继 1999 年推出 C++Builder 4.0 之后又全面升级的基于 C++ 语言的应用程序快速开发工具。C++Builder 由于其快速的开发效率、可视化的开发界面、可开发出功能强大的数据库应用程序的能力以及强大的网络支持能力等深受用户的喜爱。

本书通过对完整的中国象棋程序实例的剖析来介绍如何运用 C++Builder 进行应用程序的开发。内容包括: C++Builder 5.0 入门知识、人工智能与中国象棋的原理及应用、中国象棋实例编程技巧。该实例功能强大,可进行实战对弈,也可摆成各种残局,对弈同时由程序给出各种提示信息,并可对中国象棋进行定量分析。该实例涵盖了各种编程技巧,包括子窗体、动态菜单、自定义光标、画布的使用、动态对话框、动画处理、消息的重映射、异常的捕获、Windows API 函数和自定义函数的调用等,读者既可以感受到计算机技术的巨大魅力,也可以体会中国象棋的博大精深。

本书内容层次分明,由浅入深,实用性和指导性强。可作为 C++Builder 5.0 初学者、有一定开发经验的编程人员和广大象棋爱好者的参考用书。

版权所有,翻印必究。

本书封面贴有清华大学出版社激光防伪标签,无标签者不得销售。

书 名: C++Builder 程序设计范例——中国象棋

作 者: 胡 达 著

责任编辑: 桑任松

出 版 者: 清华大学出版社(北京清华大学学研大厦,邮编 100084)

<http://www.tup.tsinghua.edu.cn>

印 刷 者: 北京市人民文学印刷厂

发 行 者: 新华书店总店北京发行所

开 本: 787×1092 1/16 印张: 25.75 字数: 617 千字

版 次: 2002 年 1 月第 1 版 2002 年 1 月第 1 次印刷

书 号: ISBN 7-900637-77-X

印 数: 0001~4000

定 价: 45.00 元(含 1 张光盘)

前 言

C++Builder 5.0 是 Borland/Inprise 公司继 C++Builder 4.0 之后在 2000 年推出的最新版本, 是基于 C++ 语言的最快速应用程序可视化开发工具(RAD)。全面升级的 C++Builder 5.0 再次掀起了一股 C++Builder 热。

本书主要以中国象棋编程为主线, 以软件工程方法为指导, 在此基础上介绍 C++Builder 的各种开发技术和方法, 让读者在娱乐中学习、提高。作者具有多年的编程经验, 因此在对概念的理解上和示例代码的编写上都充分体现了编程人员的观点, 条理清晰, 重点明确。

本书总共分为 13 章, 内容由浅入深, 基本上涵盖了应用软件工程方法开发应用程序方面的整个流程, 安排如下:

第 1 章介绍 C++Builder 5.0 的新特性以及安装方法。

第 2 章概述了 C++ 语言的基础知识及 C++Builder 5.0 语言的新支持。

第 3 章介绍了集成开发环境, 以便读者熟悉各种开发工具的使用。

第 4 章首先建立一个简单的 CHESS 应用程序, 然后通过扩展其功能而臻于完善。

第 5 章围绕 CHESS 项目的调试介绍了各种调试工具和调试技巧。

第 6 章先介绍图形与图像的理论知识, 然后结合 CHESS 项目实例说明使用图形方法和移动棋子动画知识。

第 7 章介绍了自定义消息及其捕获与响应。

第 8 章介绍了异常的分类、发送、捕获、定制及异常处理实例。

第 9 章较全面地介绍了人工智能与中国象棋的博弈理论、程序编制。

第 10 章介绍了非可视化子窗体技术。

第 11 章介绍了可视化对话框的基本制作。

第 12 章介绍了非可视化对话框(动态对话框)技术。

第 13 章介绍了 CHESS 项目中 API 函数及自定义函数的声明及调用。

通过对本书的学习, 可以让您对 C++Builder 的使用有一个完整而详尽的了解, 并成为一个编程高手。

本书的内容浅显易懂, 操作步骤简明扼要, 由于附有象棋游戏范例作为模拟训练, 非常适合于初学者自学, 对于有一定经验的用户, 也不失为一本提高的好书。随书附带的光盘中包含有中国象棋程序实例的源代码及其相应的执行程序, 可供读者参考。

由于作者水平所限, 加之创作时间紧迫, 因此书中必定存有不少疏漏和错误之处, 恳请读者批评指正。

作者

2001 年 9 月

目 录

第 1 章 C++Builder 5.0 入门.....	1
1.1 C++Builder 5.0 简介	1
1.2 C++Builder 5.0 的特点	1
1.3 C++Builder 5.0 的安装与运行	3
1.3.1 C++Builder 5.0 对系统配置的要求.....	4
1.3.2 安装 C++Builder 5.0	4
1.3.3 卸载 C++Builder 5.0	7
第 2 章 C++与 C++Builder 5.0 语言支持.....	9
2.1 C++语言简介	9
2.1.1 程序结构	9
2.1.2 运算符	10
2.1.3 判断结构	13
2.1.4 循环结构	15
2.1.5 数据类型	17
2.1.6 数据结构	19
2.2 C++类	21
2.2.1 类的定义	21
2.2.2 构造函数与析构函数	23
2.2.3 数据成员和成员函数	24
2.2.4 多态性和虚函数	27
2.3 VCL 的 C++语言支持	30
2.4 CHES 项目中的数据类型及类的定义.....	33
2.4.1 CDefines.h 中的数据类型	33
2.4.2 TEditForm 类	36
2.4.3 TColorDlg 类.....	37
2.4.4 TInfoForm 类	38
2.4.5 TMainForm 类.....	40
2.4.6 TPieceValueDlg 类.....	44
第 3 章 集成开发环境.....	46
3.1 窗体	46
3.2 菜单栏	47
3.3 工具栏	48

3.4	组件栏	49
3.5	对象观察器	60
3.6	项目管理器	61
3.7	代码编辑器	63
3.8	图像编辑器	63
3.9	桌面工具栏	65
3.10	使用联机帮助	66
第 4 章	运行 CHESS 项目	68
4.1	管理 CHESS 项目	68
4.1.1	开始 CHESS 项目	68
4.1.2	设计对弈主窗体	69
4.1.3	交互菜单设计	107
4.1.4	运行时修改菜单	112
4.1.5	时钟组件	113
4.2	扩展 CHESS 项目	114
4.2.1	CHES 项目文件	115
4.2.2	功能菜单	117
4.3	编译、运行 CHESS 项目	120
第 5 章	调试 CHESS 项目	121
5.1	程序错误的种类	121
5.2	C++Builder 集成环境中的调试选项	122
5.3	主要调试手段	122
5.3.1	使用 Evaluate/Modify 窗口	123
5.3.2	断点	123
5.3.3	观察变量	124
5.3.4	观察类和对象	125
5.4	辅助调试工具	126
5.4.1	CPU 窗口调试工具	126
5.4.2	CodeGuard 调试工具	127
5.5	MessageBox 的妙用	128
5.6	巧用小项目文件	129
5.7	CHESS 项目综合调试实例	131
5.7.1	综合各种小技巧调试	131
5.7.2	采用输出文件的方式调试	133
第 6 章	美化 CHESS 项目——图形与图像	137
6.1	Windows 图形编程知识	137
6.1.1	设备描述表	137
6.1.2	图形设备接口	138

6.2 图形功能	139
6.2.1 画布(TCanvas)的属性	142
6.2.2 画布对象的常用方法	146
6.2.3 实现 CHESS 项目画布	148
6.3 图像功能	148
6.3.1 TGraphic 类	148
6.3.2 TPicture 类	149
6.3.3 TImage 组件	150
6.3.4 TShape 组件	150
6.3.5 TPaintBox 组件	151
6.3.6 创建和使用自定义光标	151
6.3.7 实现 CHESS 项目图像	152
6.4 生成动画	153
6.4.1 动画简史	154
6.4.2 生成棋子动画	154
6.4.3 棋子动画函数实现	155
第 7 章 跟踪 CHESS 项目——消息处理	162
7.1 Windows 消息与象棋窗体事件	162
7.2 自定义象棋消息	163
7.2.1 声明一个消息标识符	163
7.2.2 声明一个消息结构	163
7.2.3 定义消息处理句柄	164
7.2.4 发送和传递消息	165
7.3 CHESS 项目消息处理实例	165
第 8 章 拦截 CHESS 项目错误——异常处理	168
8.1 异常的分类	168
8.1.1 结构化异常处理	168
8.1.2 C++异常处理	169
8.1.3 VCL 异常处理	170
8.2 VCL 异常类	171
8.3 发送异常	173
8.4 捕获异常	174
8.5 定制异常	174
8.6 运行象棋异常处理	175
第 9 章 人工智能与中国象棋	179
9.1 人工智能简介	179
9.1.1 人工智能的研究方法	179
9.1.2 人工智能简史	180

9.1.3 人工智能问题的通用解答	182
9.2 敌对搜索	184
9.2.1 双方博弈的一般性理论	184
9.2.2 最小最大化过程	184
9.2.3 象棋 α - β 过程	185
9.3 中国象棋简论	187
9.3.1 中国象棋的难点——势	187
9.3.2 势是多项式	188
9.4 程序编制	192
9.4.1 中国象棋的表达	193
9.4.2 中国象棋规则的实现	194
9.4.3 局面(势)评价函数的实现	218
9.4.4 搜索过程的实现	235
第 10 章 CHESS 项目非可视化子窗体	256
10.1 棋盘编辑子窗体 EditForm	256
10.1.1 EditForm 窗体的属性	256
10.1.2 EditForm 窗体的方法	258
10.2 对弈信息子窗体 InfoForm	264
10.2.1 InfoForm 窗体的属性	265
10.2.2 InfoForm 窗体的方法	270
10.3 象棋子窗体的显示	277
第 11 章 CHESS 项目可视化对话框	278
11.1 棋盘打开保存公用对话框	278
11.1.1 棋盘打开对话框 OpenDialog1 的主要属性	278
11.1.2 棋盘打开对话框 OpenDialog1 的方法	279
11.1.3 棋盘保存对话框 SaveDialog1 的主要属性	280
11.1.4 棋盘保存对话框 SaveDialog1 的方法	281
11.2 自定义象棋对话框	282
11.2.1 象棋说明对话框 AboutDlg 的主要属性	282
11.2.2 改变棋盘颜色对话框 ColorDlg 的主要属性	286
11.2.3 改变棋盘颜色对话框 ColorDlg 的方法	292
11.2.4 改变棋子估值对话框 PieceValueDlg 的主要属性	297
11.2.5 改变棋子估值对话框 PieceValueDlg 的方法	303
11.2.6 定义对局时间对话框 InputMinPerGameDlg 的主要属性	311
11.2.7 定义对局时间对话框 InputMinPerGameDlg 的方法	313
11.2.8 定义走步时间对话框 InputSecPerMoveDlg 的主要属性	315
11.2.9 定义走步时间对话框 InputSecPerMoveDlg 的方法	318

第 12 章	CHESS 项目非可视化对话框 ——动态对话框	320
12.1	报告错误对话框 ChessErrorDlg 的主要属性	320
12.2	报告错误对话框 ChessErrorDlg 的方法	323
12.3	结束对弈对话框 EndChessGameDlg 的主要属性	324
12.4	结束对弈对话框 EndChessGameDlg 的主要方法	328
第 13 章	CHESS 项目的 API 函数及自定义函数	329
13.1	CHESS 项目中 Windows API 函数详解	329
13.2	自定义函数的声明及调用	339
13.3	CHESS 程序中部分自定义函数的使用说明	342

第 1 章 C++Builder 5.0 入门

C++Builder 5.0 是一个完全面向对象的编程工具，与 C++Builder 4.0 相比，新增了许多功能，已成为业界领先的面向对象开发工具。

1.1 C++Builder 5.0 简介

C++Builder 是 Inprise 公司在 Delphi 基础上重点推出的又一可视化应用程序集成开发工具。C++Builder 5.0 是该公司 2000 年推出的最新版本，它不仅继承了以前版本使用简单、功能强大、效率高和灵活性强的特点，而且又增加了许多新的特性，并融进了当今软件领域的新技术。

C++Builder 继承了 Delphi 的可视化构件库(VCL)，功能强大，使编程变得非常简单。同时，C++Builder 具有 C++开发环境所能提供的全部功能：快速、高效、灵活的编译器，逐步连接，CPU 透视和命令行工具等。由于 C++Builder 实现了可视化与功能强大的编程语言 C++很好的结合，既稳定了原来的 Delphi 编程人员队伍，又吸引了广大的新一代 C/C++语言用户，专家们预言 C++Builder 将成为 21 世纪的主流开发工具，正受到越来越多的开发人员的青睐。

1.2 C++Builder 5.0 的特点

C++Builder 5.0 相对于其他开发工具而言，有很多突出的特点，比如 C++Builder 5.0 所使用的 VCL 类库提供了 200 个左右丰富的组件，这就使利用 C++Builder 开发可视化应用程序，特别是开发数据库和网络应用程序非常快速方便容易。另外 C++Builder 5.0 还增加了许多新的特性。

- 强大的编译功能

C++Builder 5.0 在原有的基础上提供了符合 ANSI/ISO 标准的、功能强大的 C++编译器，支持后台编译，它的 CPU 透视工具包括 5 个独立的小面板，可以对正在运行的程序从内部进行深层次的了解，还提供了专业开发环境所必需的命令行工具。C++Builder 5.0 还支持 32 位长文件名、多线程的程序设计，支持 Windows 95/98/NT 和 Windows 2000 下的各项标准，包含完整的 Win32 API、COM、COM+、ActiveX、OLE Automation、DirectX、OLE-DB、ODBC、ISAPI 和 NSAPI，是惟一能够轻松建立 Windows 2000/NT Service、Windows 控制面板组件的 C++快速开发工具。C++Builder 5.0 还提供封装 Office 97/2000 Automation 的 VCL 组件，增强 Microsoft Office 系列软件包的集成能力。C++Builder 5.0

编译器还支持对 Visual C++ 的 MFC 和 Borland C++ 的 OWL 的调用,增加了两个修饰字: `_msreturn` 和 `_msfastcall`、两个编译开关: `-VM` 和 `-pm`。

- 真正可视化的 C++ 开发环境

C++Builder 运用面向对象技术构造以组件为基础的开发结构,软件组件让程序代码可重复使用的能力大幅提高,缩短了整个软件开发的周期。C++Builder 5.0 所使用的获得市场验证的可视化组件库(VCL)和其他面向对象编程语言所使用的类库(如 Visual C++ 的 MFC 和 Borland C++ 的 OWL)一样,每一个类都包括对象的属性、事件和方法。程序员通过设置对象的属性来控制对象的外观,通过对事件的处理来实现消息处理机制,通过利用方法来管理对象。C++Builder 5.0 的 VCL 包括众多可重用组件,包括 Windows 98/2000 常用组件(如对话框组件、TCppWebBrowser 组件等);数据库组件(如 DataSource 组件、IBX 组件、ADO 组件);网络组件(如 TWebConnection 组件、InternetExpress 组件)和图形和图像组件(如 Image、Icon 组件)等。

- 理想的数据库开发功能

C++Builder 5.0 提供了 32 位数据库引擎 BDE(Borland Database Engine),所有数据库连接指令,均会通过 BDE 进行处理。BDE 支持桌面型数据库 Paradox 和 Visual dBase, BDE 与 ODBC 具有良好的接口,可以直接存取 Microsoft Access 和 FoxPro 等数据库。对于单机使用数据库, C++Builder 5.0 可直接通过 BDE 来控制数据库的存取;对于远程数据库, BDE 需通过 SQL Links 或 ODBC 才能读取数据库的内容。

增加了 ADOExpress(Enterprise and add-on to Professional editions)组件集。ADO 是一种数据接口。ADOExpress 组件集通过使用 ADO(ActiveX Data Objects)以及 OLE DB 驱动程序,配合各种数据感知组件,使得应用程序可以不通过 BDE 而直接访问不同类型的数据库(包括电子信箱和普通文件系统,传统的关系型及非关系型数据库)。

C++Builder 5.0 集成了 IBX 元件,通过 Interbase API 来访问 InterBase 数据库,绕过了 BDE。

- 快速的网络开发

C++Builder 5.0 提供了 20 多个原生 Internet 通讯协议加速了 Internet 应用程序开发。用户只需将网络组件拖到窗体上,进行属性的设置和事件的处理,就能建立 Internet 应用程序。利用 ActiveX 组件,可以实现 Web 浏览器进行文件传输等 Internet 功能。C++Builder 5.0 还支持 Microsoft、Sun、Netscape 等公司开发的 Internet 标准,如 ActiveX、CGI、ISAPI、Winnet 和 NSAPI 等功能。缩短了开发时间并降低了开发 Web 应用系统的成本,使开发者能够在预算内准时将现有业务拓展至 Internet 的广阔领域。

- 轻松的多媒体、图形和图像编程

和其他开发工具一样, C++Builder 5.0 也提供了多媒体组件、图形组件和图像组件,这就使得用户开发多媒体应用程序成为一项愉快的任务。同时, C++Builder 5.0 也可以使用 DirectDraw 和 OpenGL 进行深层次的开发。

- MIDAS 功能得到进一步增强

MIDAS 是一组运用 COM、COM+、CORBA 等分布式对象规范进行多层开发的技术。C++Builder 一直鼓励开发者把数据访问与数据操纵分开,以实现 3 层(甚至更多层)的逻辑结构。从 C++Builder 5.0 开始, MIDAS 支持无状态(Stateless)远程数据模式,这样,在编

写 MTS 服务器、COM+服务器和共享的远程数据模块时，不必创建自定义的接口，同时程序可以通过使用 Internet Express 组件集中的组件来创建网络应用程序，这种程序可以使用户很方便地通过 Web 方式与 MIDAS 服务器中的数据进行交互。

- 增加了一些新的工具和开发向导

CodeGuard 是 C++Builder 5.0 中新增加的一个工具，该工具主要用于捕捉运行调试时出现的错误，这些错误并没有违反一般的语法(一般的语法错误是能够被编译器发现并提示的)。例如，当用户的程序试图访问已经被删除的内存空间或资源时，CodeGuard 将会报告出错信息。比如可以为特殊的进程设置调试的功能，可修复内存中的 Bug，易于定位、分析内存泻漏及堆栈中的错误，尤其是断点的控制，因而大大加快了程序的开发效率。CodeGuard 主要检查失败的内存使用、不正确的内存分配或释放、文件流的不正确使用以及由此造成的资源缺乏、指针的引用和函数的调用失败等产生的各种错误。

C++Builder 5.0 还增加了一些新的向导，使开发过程更加方便、简单、准确。例如，控制面板程序向导(Control Panel Application)用来创建控制面板程序，控制台应用程序向导(Console Wizard)用来创建控制台应用程序，创建动态链接库向导(DLL Wizard)使得创建动态链接库变得非常容易和迅速。

- 可平稳移植至 Linux 操作系统

Borland/Inprise 是目前业界致力于 Linux 推广的重要成员，C++Builder 5.0 特意为有意在 Linux 平台开发软件的技术人员做好了准备。Borland/Inprise 是惟一能够衔接 Windows 与 Linux 两大领域，提供最佳解决方案的厂商。

- TeamSource 增强团队开发效率

C++Builder 5.0 企业版提供全新 Team Source 版本管理系统，除了提供传统程序版本管理系统的必要功能外，Team Source 还融入了工作流程管理工具，能够针对分散于不同地点的大型开发团队提供更有源的源程序管理机制。Team Source 版本管理系统还可支持 PVCS 及其他源程序版本管理系统引擎，精确管理开发团队的程序代码版本，大大简化了团队开发。

- Borland Translation Suite 帮助开发国际化应用系统

C++Builder 5.0 企业版内置的 Borland Translation Suite 可以轻松地翻译人机界面和系统信息，帮助用户开发适合国际化需求、能够在不同语言 Windows 环境执行的应用系统，还包含多项方便工具，可管理和重用已经翻译过的文字内容，方便地进行软件的本地化和国际化。

1.3 C++Builder 5.0 的安装与运行

C++Builder 5.0 的安装与运行十分简单，只需按照提示一步步操作即可。下面将介绍安装 C++Builder 5.0 的系统要求以及常用安装方法。

1.3.1 C++Builder 5.0 对系统配置的要求

安装 C++Builder 5.0 对系统配置的要求如下:

- (1) Intel Pentium 90 或者更高。
- (2) 企业版通常安装需要 253MB 的自由磁盘空间, 完全安装需要 388MB 的自由磁盘空间(硬盘的空间越大, 对程序代码的执行越有利)。
- (3) 32MB 内存(建议使用 64MB 内存)。
- (4) VGA 或更高级的显示卡, CD-ROM, Windows 环境兼容鼠标。
- (5) 运行平台要求 Windows 2000、Windows 98、Windows 95、Windows NT 4.0(Service Pack 3 以上)。

C++Builder 5.0 发行的版本有标准版(Standard Edition)、专业版(Profession Edition)和企业版(Enterprise Edition)等, 各种版本功能上有一定的差异。本书所用的版本为企业版。

1.3.2 安装 C++Builder 5.0

使用目前最常用的安装软件 Install Shield 制作出的安装程序往往有一种基本一致的可视化界面, 用户根据安装程序的提示信息就可轻松完成安装。

- (1) 把 C++Builder 的安装光盘放在光驱中, 将自动弹出安装主画面。用户也可以在安装文件所在目录下, 找到 install.exe 文件, 双击该文件, 即可弹出安装主画面。安装主画面如图 1.1 所示, 有多个选项, 每单击一个选项就进行该项的安装。

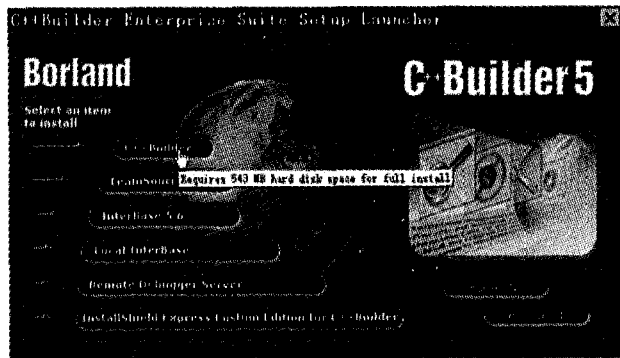


图 1.1 安装主界面

在如图 1.1 所示的安装界面上的各选项的含义如下:

- C++Builder: 将 C++Builder 5.0 安装到硬盘空间中;
- TeamSource: 开发流水线管理的工具, 以帮助应用程序的开发小组管理在共享开发环境中的日常任务;
- InterBase 5.6: 安装远程数据库开发工具;
- Local InterBase: 安装本地数据库开发工具;
- Remote Debugger Server: 安装远程调试服务工具;

- **InstallShield Express Custom Edition For C++Builder:** InstallShield 是一个专门进行制作安装系统的工具，用于将开发出来的应用程序打包发布。

另外还有两个选项：**Register Now!**和**Browse CD**。前者将进行联机注册，后者将浏览安装光盘中的内容。

- (2) 开始安装。安装前程序首先进行安装信息的搜集，如图 1.2 所示，进度条显示搜集信息的进度。随后进入一个欢迎画面，附带显示一些版权信息，单击 **Next** 按钮，继续安装的下一步。

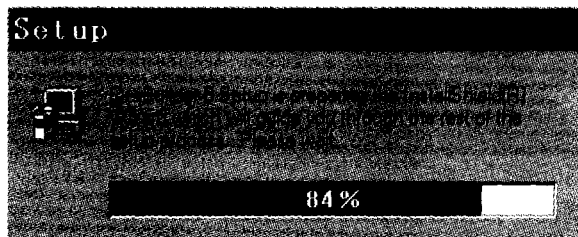


图 1.2 搜集安装信息

- (3) 输入正确的产品序列号，否则，安装将无法完成。产品序列号在光盘的封面或光盘中的文本文件 **sn.txt** 中。输入序列号后，继续下一步。图 1.3 显示了通常的典型安装方式。

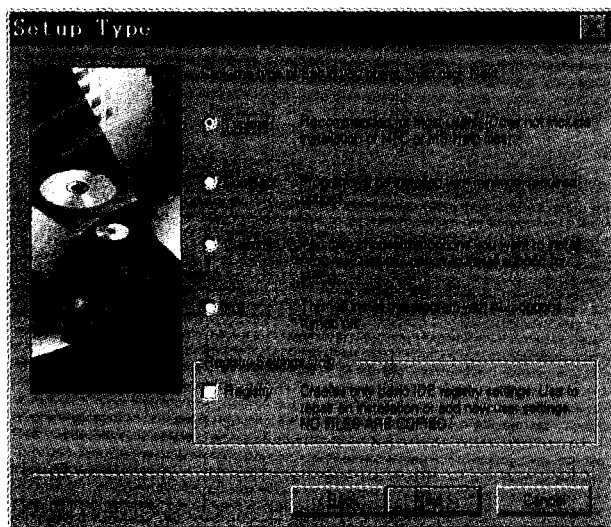


图 1.3 典型安装

要说明的是，如果已经安装了 C++Builder 4.0 或者更早的版本，最好在【控制面板】中的【添加/删除程序】对话框中予以卸载后再进行安装。

如果在原来 C++Builder 4.0 的基础上进行升级安装，将出现如图 1.4 和图 1.5 所示的对话框。图 1.4 询问用户是否把已存在的扩展名关联到 C++Builder 5.0。图 1.5 中出现的安装选项让用户选择某些项目，以便让系统自动安装。

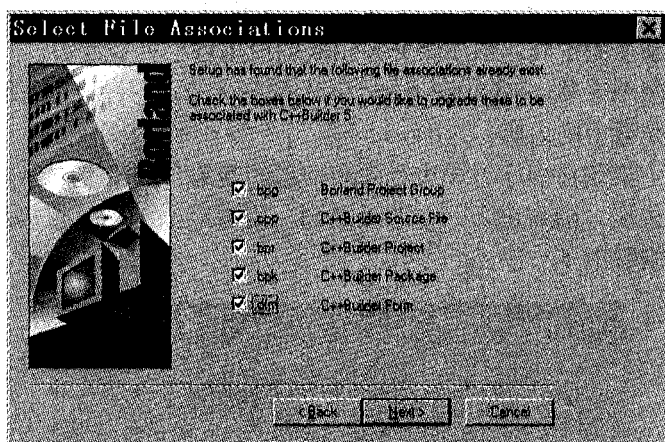


图 1.4 相关扩展名

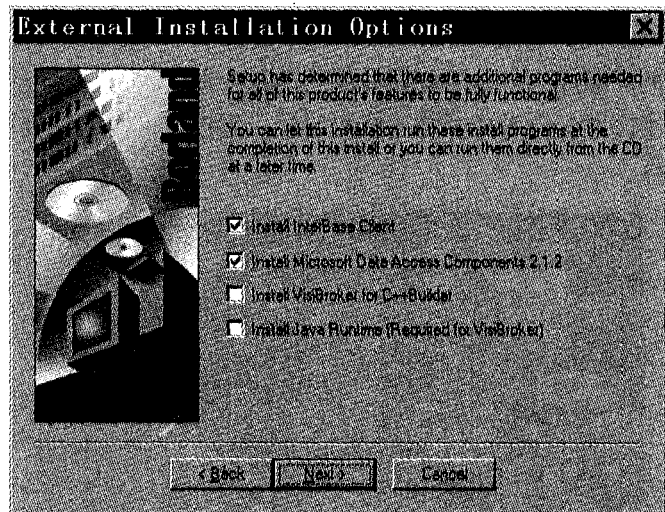


图 1.5 安装选项

- (4) 选择安装路径。系统默认的安装路径是在 C 盘，一共有 4 个安装路径，但它们只有最低一级目录不同，而父目录则都相同。用户可根据自己的工作习惯及硬盘空间的大小来设置，这可以通过单击 Browse 按钮进行选择，在弹出的对话框中设置好路径后继续安装。如图 1.6 所示。
- (5) 单击 Next 按钮后，将弹出如图 1.7 所示的安装界面，开始复制安装程序，有一个进度条显示复制的进展情况。
- (6) 让安装生效。在程序复制完后，用户只需要重新启动计算机即可。
- (7) 运行程序。当安装完成后，系统就自动地在桌面上建立了快捷方式，双击快捷图标即可运行 C++Builder 5.0；也可通过【开始】菜单中的可执行程序项来启动运行程序。

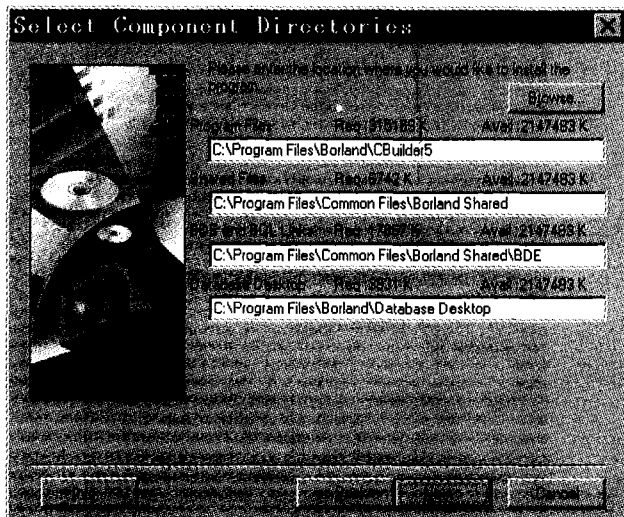


图 1.6 设置路径

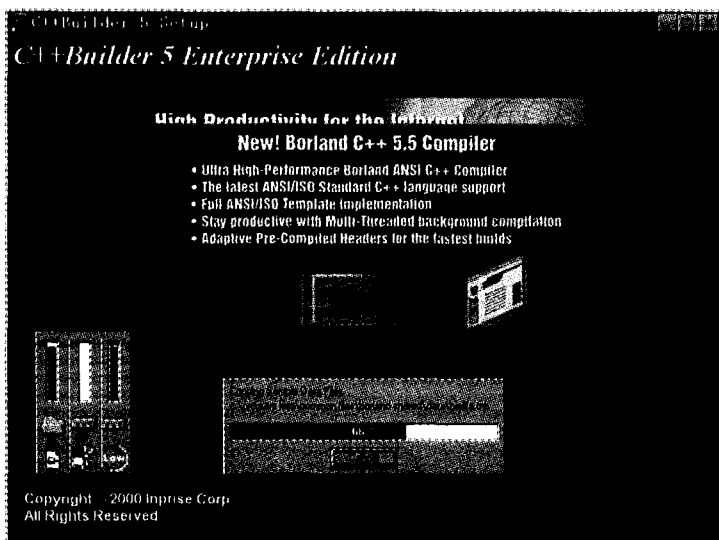


图 1.7 复制安装程序

1.3.3 卸载 C++Builder 5.0

如果要卸载 C++Builder 5.0, 最好通过 Windows 系统【控制面板】上的【添加/删除程序】方式来进行(如图 1.8 所示)。

- (1) 单击【Borland C++Builder 5】选中该组件。
- (2) 单击【确定】按钮。

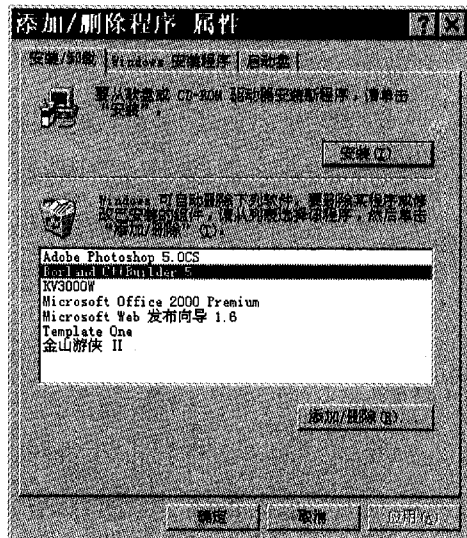


图 1.8 卸载安装

采用这种方式删除最彻底，但如果用户在 C++Builder 5.0 的目录下建有文件夹或文件的话，还需要手工删除这些文件夹或文件。

第2章 C++与C++Builder 5.0 语言支持

C++Builder 5.0 支持最新版本 ANSI/ISO C++ 标准，完整实现 ANSI/ISO C++ Template 相关规范，能够让用户开发出可移植至 Windows 以外操作平台的 C++ 程序。对 ANSI/ISO C++ 标准在某些方面未做明确定义的，C++Builder 5.0 自由地定义了这些方面。

2.1 C++语言简介

C++语言是美国贝尔实验室的 Bjarne Stroustrup 博士在 C 语言的基础上，于 1980 年开发出来的一种过程性与对象性结合的程序设计语言，最初被称作“含类的 C”，到 1983 年才取名为 C++。C++语言继承了 C 的原有精髓，如效率高、灵活性好的特点，扩充增加了对开发大型软件颇为有效的面向对象机制，弥补了 C 语言不支持代码重用、不适宜开发大型软件的不足，是优秀的程序设计语言之一。

2.1.1 程序结构

C++程序是结构化程序，由 3 种基本结构组成：顺序结构、选择结构和循环结构。从形式上分为声明区、主程序区和函数定义区。声明区主要包括包含文件、宏定义等。在 C++Builder 5.0 中，WinMain 作为主程序的入口点。在没有跳转条件和循环条件的情况下，程序将按书写顺序一行接一行地执行。

1. 编译预处理

- `#include` 包含指令：把一个头文件插入到当前源文件中。如：`#include <stdio.h>`表示引用 C 语言中标准输入输出库函数。
- `#define` 宏定义指令：定义常量、带参数的宏。如：`#define Pi 3.14159265`表示圆周率 Pi 取值为常数 3.14159265。

2. 注释

C++中有两种类型的注释：

- `/* */`：忽略“`/*`”与“`*/`”之间的所有内容(即注释)，注释内容允许占多行。如：

```
/* This is  
a example. */
```

● **//**: 忽略从“//”到本行结束处的所有内容。如: `x=y+z; //This is a comment`
在程序中使用任一种注释,但不要将两类注释符混合使用。`/*...*/`注释不能具有嵌套结构,但它可以嵌套**//**方式的注释。如:

```
/* This is the begin of the multiline comment.  
Inside of which // is nested a single line comment  
Here is the end of the multiline comment. */
```

在 C++Builder 5.0 中注释部分以蓝色的斜体字表示,很容易分辨。

3. 语句

语句就是可执行代码。程序和函数都由语句构成。语句分为简单语句与复合语句。比如赋值语句、过程及函数的调用都属于简单语句。

```
int M;  
AnsiString String[20];  
M = 5; //变量 M 赋值为 5  
String = "Hello"; //把 "Hello" 指定给字符型变量 String  
Function1(); //执行函数 Function1
```

任何可以使用简单语句的地方都可以使用复合语句。复合语句是由一组语句构成的程序块,“{”为复合语句的开始分隔符,“}”为终止分隔符。例如:

```
if (k == 2)  
{  
    showMessage("Error k!");  
}
```

4. 分号

在 C++ 中,可以随意使用多行代码构成一个语句。只有简单语句必须用分号隔开。因为复合语句使用“{”和“}”作为分隔符,而不需要用分号隔开。

```
if (k == 2)  
{  
    showMessage("Error k!");  
    k = 3; //这里需要分号  
} //这里不需要分号
```

2.1.2 运算符

运算符是对数据进行操作、变换的符号。C++ 语言中定义了许多运算符,基本上可分为 3 类:算术运算符、位运算符和专用运算符。

1. 算术运算符

算术运算符类似于数学中的常用运算符，如表 2.1 所示。

表 2.1 算术运算符

运算符	运 算	运算变量类型	运算结果类型
+	加法	整数或实数	整数或浮点数
-	减法	整数或实数	整数或浮点数
*	乘法	整数或实数	整数或浮点数
/	除法	整数或实数	浮点数
div	整除	整数	整数
mod(%)	模(求余数)	整数	整数

例如：

```

5 + 3 = 8;
100 - 20 = 80;
2 * 4 = 8;
10 / 2 = 5;
20 div 7 = 2;
20 % 7 = 6;

```

2. 关系运算符

比较两个操作数大小的符号，是双目运算符，如表 2.2 所示。

表 2.2 关系运算符

关系运算符	意 义	相当于
<	小于	<
>	大于	>
<=	小于等于	≤
>=	大于等于	≥
==	相等于	=
!=	不等于	≠

- 小于运算符< 在表达式 E1<E2 中，要求两操作数都是算术类型，或为指向限定或未限定对象的指针类型。如果 E1 的值小于 E2 的值，结果为 1，否则为 0。
- 大于运算符> 在表达式 E1>E2 中，要求两操作数都是算术类型，或为指向限定或未限定对象的指针类型。如果 E1 的值大于 E2 的值，结果为 1，否则为 0。
- 小于等于运算符<= 在表达式 E1<=E2 中，要求两操作数都是算术类型，或为指向限定或未限定对象的指针类型。如果 E1 的值小于等于 E2 的值，结果为 1，否则为 0。

- 等于运算符== 在表达式 $E1=E2$ 中, 要求两操作数都是算术类型, 或为指向限定或未限定对象的指针类型。如果 $E1$ 的值等于 $E2$ 的值, 结果为 1, 否则为 0。
- 不等于运算符!= 在表达式 $E1!=E2$ 中, 要求两操作数都是算术类型, 或为指向限定或未限定对象的指针类型。如果 $E1$ 的值不等于 $E2$ 的值, 结果为 1, 否则为 0。

3. 位运算符

类似于汇编语言的位操作。

- 左移<< 左移一位相当于乘以 2, 被挤出的最高位将丢失。如:
 $(11000000)_2 \ll 1 = (10000000)_2$
- 右移>> 右移一位相当于除以 2, 损失的位将不能补回, 如:
 $(11000001)_2 \gg 1 = (01100000)_2$
- 位与(AND)& 要求两操作数都为整数类型。如: $(0101)_2 \& (0011)_2 = (0001)_2$
- 位异或(XOR)^ 要求两操作数都为整数类型。如: $(0101)_2 \wedge (0011)_2 = (0110)_2$
- 位或(OR)| 要求两操作数都为整数类型。如: $(0101)_2 | (0011)_2 = (0111)_2$

4. 逻辑运算符

- 逻辑与(AND)&& 要求两操作数都为布尔类型, 如果两操作数都为非零, 则结果为 1, 否则结果为零。如: $1 \&\& 1 = 1, 1 \&\& 0 = 0$ 。
- 逻辑或(OR)|| 要求两操作数都为布尔类型, 如果两操作数中有一个非零, 则结果为 1, 否则结果为零。如: $1 || 1 = 1, 1 || 0 = 1$ 。

5. 赋值运算符

分为简单赋值运算符和复合赋值运算符。只有等号为简单赋值运算符, 形如 $op=(op$ 指操作符)称为复合赋值运算符。

- 简单赋值运算符= 在表达式 $E1=E2$ 中, $E1$ 为可修改的左值。 $E2$ 的值在转换为 $E1$ 类型后, 存放到由 $E1$ 指明的对象中, 取代 $E1$ 原先的值。
- 复合赋值运算符有 10 个: 积赋值*=、商赋值/=、模赋值%=、和赋值+=、差赋值-=、左移位赋值<<=、右移位赋值>>=、按位与赋值&=、按位异或赋值^=、按位或赋值|=。它们中 r 任何一个解释如下:

$E1 \text{ op} = E2$

与

$E1 = E1 \text{ op} E2$

作用相同, 只是前者左值 $E1$ 仅计算一次。具体说 $E1 += E2$ 与 $E1 = E1 + E2$ 相同。

6. 结构选择运算符(.和->)

结构与联合的成员可用选择操作符 . 和 -> 进行存取。假定 s 为结构类型 S 的一个对象, $sptr$ 为指向 s 的指针, 若 i 为在 S 中说明的类型为 I 的成员标识符, 则表达式 $s.i$ 和 $sptr->i$ 都为 I 类型, 且都表示 s 中的成员对象 i 。表达式 $sptr->i$ 和 $(*sptr).i$ 同义。例如:


```
struct MyStruct
{
    int i;
    double d;
} s, *sptr = &s;
...
s.i = 5;           //赋值给 MyStruct 的成员 i
sptr->d = 2.45;     //赋值给 MyStruct 的成员 d
```

7. 三元条件运算符?:

运算符?: 可以用来代替如下形式的 if else 语句:

```
if(条件)
    表达式
else
    表达式
```

关键是 if 和 else 后必须是单个表达式, 而不能是其他 C 语言语句。

它要求 3 个操作数, 形式如下:

EXP1?EXP2:EXP3

表达式的值是这样的: 先求表达式 1 的值, 若为真则求表达式 2 的值并把这作为整个表达式的值。若为假则计算表达式 3 的值并作为整个表达式的值。例如:

例如 a?T:F 若 a 为真, 则结果为 T, 否则结果为 F。

8. 逗号运算符(,)

逗号运算符用于将多个表达式串在一起, 逗号运算符左边的值总是不返回的, 也就是说, 逗号右边表达式的值才是整个表达式的值。例如:

```
x=(y=5,y+1);
```

首先把 5 赋给 y, 然后把 6 赋给 x。整个等式右边的表达式要用括号括起来, 因为逗号运算符的优先级低于赋值号。例如:

```
y=30;
x=(y=y-5,25/y);
```

执行后 x 的值为 1, y 的值为 25。因为 y 的初值是 30, 减 5 后为 25, 再将 25 除以 25 所得结果是 1。

2.1.3 判断结构

判断结构是一种基本的程序结构, 完成一定功能的程序一般都会有一个或多个判断结构。构成判断结构的关键是条件转移命令。一般分为二分支结构和多分支结构。C++支持多

种判断结构，它们分别适用于一些控制流程。

1. if...else 语句

条件语句 if...else 的基本形式为：

```
if (逻辑表达式)
<语句>;
```

可在以上基础加上 else 子句，其基本语法为：

```
if (逻辑表达式) <语句 1>
else
```

```
    <语句 2>;
```

该语句的另一变形是在 else 后再跟一个 if 子句，如：

```
if (逻辑表达式) <语句 1>;
else if (逻辑表达式) <语句 2>;
```

当多于一条语句时，要用“{”和“}”括起来。条件语句可以嵌套。例如：

```
# include <stdio.h>
void main()
{
    int i;
    printf("请输入一个数：");
    scanf("%d",&i);
    if (i<0)
        printf("数值为负。");
    else
        printf("数值为零或正。");
}
```

该程序判别输入每个整数是否为负数。

2. switch <x> case 语句

当需判断的条件较少时，if...else 结构行之有效，但遇到多路分支的情况，该结构变得越来越繁琐。这种情况下，使用 switch 结构更为便捷。在 16 位 Windows 编程的窗口消息循环过程中最为常见。switch 语句的基本语法如下：

```
switch (switch 整型变量)
{
    case <常量表达式>: <语句>; [break]

    default:<语句>
}
```

在 switch 语句中使用了一个变量，用它与 case 后的常量表达式的值匹配。同一个 case 下可以有多个值。例如：

```
switch (CHAR)
{
    case 'A':
        printf("\n 选择了选项 A. \n");
        break;
    case 'B':
        printf("\n 选择了选项 B. \n");
        break;
    default:
        printf("\n 选项没有激活. \n");
        return(-1);
}
```

这个例子中，通过测试字符型变量 CHAR 的值，确定选择项是选项 A 还是选项 B，如果 CHAR 的值不属于这两者之一，则显示“选项没有激活”。

switch 结构中的语句可以是简单语句，也可以是复合语句。对于复合语句必须用“{}”将这些代码括起来。

2.1.4 循环结构

在程序设计语言中，循环是 3 种基本结构之一。其特征是反复地执行一些指令，直到满足某种条件为止。C++提供了几种循环控制结构，用户可根据不同需要进行选择。有些情况下可以相互替代。

1. while 语句

当循环条件为真时，执行语句，当循环条件为假时，结束循环。while 循环的基本语法为：

```
while (<条件>) <语句>;
```

如其他成分的语法结构一样，<条件>后面的语句既可以是简单语句，也可以是复合语句。下面是简单语句：

```
while (*ptr == ' ') ptr++;
```

这个语句的作用是在*ptr 值为空格之前，始终增加 ptr；

2. for 语句

for 循环语句指定了一个控制循环次数的变量。for 语句的基本语法为：

```
for ([起始条件]; [终止条件]; [<增量>])  
(<语句>;)
```

控制变量的类型通常是整型(或某种整数类型), 起始条件确定循环的初值, 终止条件决定何时退出循环, 增量控制循环的方式, 举例如下:

```
for (int i =1;i<100;i++)  
{  
    if (i ==10)  
        cout<<"\n i="<<i;  
}
```

在执行第 1 次循环时, i 的值为 1。每执行循环一次, i 的值递增一次(递增的步长为 1)。在执行最后一次循环时, i 的值等于 99。i 的值继续递增到 100, 此时由于 i 值不小于循环终值 100, 因而循环结束。

3. do...while 语句

第 3 种类型的循环结构是 do...while 循环。它的基本语法为:

```
do{<语句>;}while(<条件>;)
```

do...while 循环将执行循环中的所有语句, 直到条件值为假时才退出循环。例如:

```
do{  
printf(输入口令: );  
scanf("%s",password);  
}while(strcmp(password,checkword));
```

以上循环会读入输入的内容, 直到 password 等于 checkword;

do...while 循环至少执行一次后, 才会测试循环条件。

4. break 与 continue 语句

break 语句使程序立即跳出包含它的循环, 而执行以后的语句。例如:

```
switch (CHAR)  
{  
    case 'A':  
        printf("\n 选择了选项 A. \n");  
        break;  
    case 'B':  
        printf("\n 选择了选项 B. \n");  
        break;  
    default:  
        printf("\n 选项没有激活. \n");
```

```
    return(-1);  
}
```

在这个例子中，如遇到 **break** 时，程序的执行点将跳到紧接循环之后的第 1 条语句。

另一个改变循环执行的命令是 **continue** 语句。**continue** 语句可使程序的执行略过本次循环而回到下一个循环。例如：

```
{  
    for (I=0;I<10;I++)  
    {  
        if (array[I] == 0)  
            continue;  
        array[I]=1/array[I];  
    }  
}
```

以上这个例子中，**continue** 语句的作用是预防除数为零的错误发生。

5. goto 语句

是一种无条件转移语句，格式如下：

```
goto <语句标号>;
```

标号必须与 **goto** 语句同处于一个函数中，但可以不在一个循环层中，一般不用，但在多重嵌套时，仍然需要用到它。例如：

```
m=1;  
SIXTY:  
m++;  
if (m<60)goto SIXTY;
```

这里 1~60 的循环使用了一个 **goto** 和一个标号。

2.1.5 数据类型

数据类型本质上定义了一组值组成的集合，以及相应运算的集合。或者说，一个数据类型定义了变量或表达式可以取值的范围，以及可以施于它们的运算。

其意义是：

- 无论常量、变量、函数或表达式，它们都有且仅有一个类型。
- 类型决定了变量、变量、函数或表达式的取值集合以及可施于的运算的集合。
- 每个值属于且仅属于一个类型。
- 任何常量、变量、函数或表达式所表示的值的类型，都可由其形式或上下文推知。

1. 数值类型

C++有两种数值类型，一种是序数类型，另一种是实数类型。

序数数值类型是整数，它没有小数部分。例如：

```
int counterN;  
int flags =5;
```

第 1 行说明中，保留字 `int` 通知程序为一个整数设立足够的空间，变量名 `counterN` 用来代表那个存储空间。

第 2 行说明建立名为 `flags` 的变量，并给它赋了初始值。

整数类型又包括若干种不同类型，如表 2.3 所示。

表 2.3 整数类型

类 型	范 围	内部格式
int	-32 768~32 767	占据 2 字节
unsigned int	0~65 535	占据 2 字节
short	-32 768~32 767	占据 2 字节
long	-2 147 483 648~2 147 483 647	占据 4 字节
unsigned long	0~4 294 967 295	占据 4 字节

实数数值类型又称为浮点数。当想要表达一个精确的数值量时，可以使用实数类型。除了精确之外，还能表示更大更小的值。

说明一个实数类型变量比较简单，用保留字 `float` 来说明变量名即可。例如：

```
float radius;  
float interst, money;  
float km=4.34;
```

第 1 行说明了一个变量，没有赋初值，第 2 行说明了两个变量，第 3 行给变量赋了初始值。

实数数值类型如表 2.4 所示。

表 2.4 实数类型

类 型	范 围	长 度
float	$3.4*(10E-38) \sim 3.4*(10E+38)$	占据 4 字节
double	$5.0*10E-324 \sim 1.7*10E308$	占据 8 字节
long double	$3.4*10E-4932 \sim 1.1*10E4932$	占据 10 字节

2. 字符类型

C++的字符类型用来存放一个单独的字符。字符类型可以存放任一 ASCII 字符。用类型定义符 `char` 定义字符数据类型。字符类型变量长度为 1 字节，一个字符可以是有符号、

无符号或未定义的(表 2.5 所示)。默认状态为有符号。被声明为字符型的对象可以足够存储基本 ASCII 字符集的任意成员。

表 2.5 字符类型

类 型	转换为	方 法
char	int	0 或扩展符号(默认类型)
unsigned char	int	高位字节填 0
signed char	int	扩展符号(永远)
short	int	同值; 扩展符号
unsigned short	unsigned int	同值; 填 0
enum	int	同值

3. 布尔类型

布尔类型变量的值只能是 true 或 false。有些编译器中写为 TRUE 或 FALSE。true = 1, false = 0。

4. 指针、自定义类型

指针分为两种主要类型: 指向对象的指针和指向函数的指针。这两种指针类型均为保存内存地址的特殊对象。

尽管 C++Builder 5.0 对这两类指针的操作相同, 但这二者具有不同的属性、目的和操作规则。一般而言, 指向函数的指针用于访问函数, 并将函数作为参数传递给其他函数, 不允许对函数执行基于指针的算术运算; 另一方面, 指向对象的指针一般会在扫描内存中的数组或更复杂的数据类型时增大或减小。

C++允许用户创建自己的数据类型。如:

```
enum number{zero,one,two,three,four,five,six,seven}anymum;
```

在以上语句中 number 是一个整数类型的枚举, anynum 是这个类型的一个变量, zero 的值为 0, one 的值为 1, 其余类推。再如:

```
enum nodes{FIRST=-5,SECOND=3;THIRD, LAST=10};
```

在以上语句中 nodes 是枚举类型, THIRD 的值为 4。

2.1.6 数据结构

数据结构是用来表示或实现抽象数据类型的。通过利用基本的数据结构可以构造出复杂的数据结构。

1. 数组

数组定义了一个包含多个数值的变量, 这些数值如同一个列表, 是有一定数目具有相同数据类型的若干变量的有序集合。在 C++中, 数组通常具有类型, 也就是说数组元素的

类型必须相同。声明一个数组类型的基本语法为：

<类型声明> <数组名> [<第一维的大小>] [<第二维的大小>]...

例如：

```
int IntArray[100]; //包含 101 个整数的数组
```

下面是一个建立二维数组的实例：

```
float **convertd;  
try {  
    convertd = new float*[m];          // 第 1 步：设置行数  
    for (int j = 0; j < m; j++)  
        convertd[j] = new float[n]; //第 2 步：设置列数  
}  
catch (std::bad_alloc)  
{  
    cout << "Could not allocate. Bye ...";  
    exit(-1);  
}
```

`float **convertd` 声明了一个二维数组，可以看出如何引用数组元素。访问数组时通过数组元素的位置即可。由于数组是基于 0 的，所以第一个数组元素的位置为 0。

2. 动态数组

动态数组是一个长度不固定的数组。仅在需要时才为动态数组分配内存(即在为动态数组元素赋值时)。

动态数组是一个实现数组可变大小的集合。C++Builder 5.0 提供 `DynamicArray` 模板作为 `DynamicArray` 的实现。`DynamicArray` 模板使用了与 Delphi 的 `DynamicArray` 实现兼容的二进制设计，这样可通过 Delphi 和 C++Builder 5.0 共享兼容的 Pascal 代码和 C++代码。

声明动态数组的语法为：

```
template <class T>class DELPHIRETURN DynamicArray;
```

例如：

```
DynamicArray<int>arrayOfInt;  
ArrayOfInt.Length = 5;  
cout<<"ArrayLength:" <<arrayOfInt.Length << endl;
```

只要给动态数组的 `Length` 赋值为 0，就可释放为其分配的内存。

3. 结构

结构数据类型可将同一类型的数据组合在一个变量下。其定义的语法为：


```
struct [<结构类型名>]{  
    [<类型><变量名[, 变量名, ...]>]  
    .  
    .  
    .  
    } [<结构变量>];
```

例如：

```
struct name_struct{  
    char name[20], telephone[40];  
    int weight, age;  
    } varname;
```

可用结构选择符(.)或(->)访问记录结构中的元素。

2.2 C++类

C++类是 C++最重要的特征，也是面向对象程序设计(OOP)的重要特征。应用 C++类能更简便地创建可重用的模块。

2.2.1 类的定义

类是 C++的核心，是面向对象编程的主要内容，同时也是 VCL 的核心。类是完成特定编程任务的变量和函数的集合。

在 C++中，类用关键字 `class` 来声明。其定义格式如下：

```
class<类名>  
{  
    public:  
        <成员函数或数据成员的说明>  
    protected:  
        <成员函数或数据成员的说明>  
    private:  
        <成员函数或数据成员的说明>  
};  
    <各个成员函数的实现>
```

其中，`class` 是声明类的关键字，<类名>是要声明的类的名字。花括号内是类的成员说明部分。类的成员包括数据成员和成员函数两部分。按访问权限又可分为：公有的(`public`)、私有的(`private`)和受保护的(`protected`)3 类。公有成员用 `public` 来说明，通常是一

些操作(成员函数)作为提供给用户的接口,这些成员可以在程序中引用。私有成员用 `private` 来说明,通常是一些数据成员,用来描述类中对象的属性,用户无法直接访问它们,而只有成员函数或经特殊声明的函数才可以引用。保护类成员的访问特性介于公有和私有成员之间。

关键字 `public`、`private` 和 `protected` 为访问权限修饰符或访问控制修饰符。它们在同一类体内出现的先后顺序无关,并可出现多次,用它们来说明类成员的访问特性。

<各个成员函数的实现>是类定义中的实现部分,包含所有在类体内说明的函数的实现,如果所有成员函数都在类体内定义,则该部分可以省略。

下面给出点类的定义,该类是二维点的抽象,其对象将是某个具体的点。

```
//-----  
class TPoint {  
public:  
    void SetX(const int val);  
    void SetY(const int val);  
    int GetX(){return x;}  
    int GetY();  
private:  
    int x,y;  
};  
//-----  
void TPoint ::SetX(const int val)  
{  
    x= val;  
}  
//-----  
    void TPoint ::SetY(const int val)  
{  
    y = val;  
}  
//-----  
int TPoint ::GetY()  
{  
    return y;  
}  
//-----
```

点类的实现部分中,对类体内说明的 4 个成员函数进行了定义,即给出了函数功能的具体实现。这里使用了作用域运算符`::`,用它来标识某个成员函数是属于哪个类的。其作用格式如下:

```

返回类型 <类名>::<函数名>(<参数表>){
// 函数体
}

```

C++通过声明类的变量来定义类的实例，即类对象。例如，对于前面定义的 TPoint 类，只需有下面的代码就可以定义类对象：

```

TPoint apoint;           //定义一个 TPoint 类对象 apoint
TPoint.SetX(200);        //初始化对象 TPoint

```

C++采用另一种称为对象引用的机制，类的对象并不真正保存类的数据，而只是个引用，指向类对象数据在内存中的地址。因此，下面的语句并没有真正创建对象，而仅是它的一个指针的位置：

```

TPoint apoint;

```

而用户自己定义的类对象，需要用户自己手工建立实例，而组件的实例可以由 C++Builder 自动建立。

2.2.2 构造函数与析构函数

构造函数是在生成类对象时自动调用的函数，其作用是在创建对象时，使用给定的值为对象分配空间，进行初始化。

构造函数是成员函数，函数体可以写在类体内，也可以写在类体外；

构造函数一般由用户自己编写，对于简单的类，若没有提供构造函数，C++Builder 5.0 编译器将提供一个默认的构造函数。构造函数名与类名完全相同，可进行重载，可以有任意类型的参数，但不具有返回类型的参数。

```

//-----
class TPoint {
public:
    TPoint(){x=y=0;}
    void SetX(const int val);
    void SetY(const int val);
    int GetX(){return x;}
    int GetY();
private:
    int x,y;
};
//-----

```

构造函数 TPoint()将把每一个点的坐标初始化为(0, 0)。

析构函数执行与构造函数相反的操作，是在删除对象时自动调用的一类特殊成员函数。析构函数也是成员函数，函数体可以写在类体内，也可以写在类体外；析构函数是用来释放一个对象所占的资源。它与构造函数相对应，没有返回值，也不带参数，不能重载，一个类中只能有一个析构函数，其命名规则是在类名前加字符“~”作为其名字。如下所示。

```
//-----  
class TPoint {  
public:  
    TPoint(){x=y=0;}; //构造函数  
    TPoint(const int x0, const int y0){x=x0;y=y0;};  
    ~TPoint(){ }; //析构函数  
  
    void SetX(const int val);  
    void SetY(const int val);  
    int GetX(){return x;}  
    int GetY();  
private:  
    int x,y;  
};  
//-----
```

2.2.3 数据成员和成员函数

C++类对结构类型进行了扩充，它不仅含有不同类型的数据，而且还可以含有函数。类结构中的数据称为数据成员，类结构中的函数称为成员函数。

1. 数据成员

类的数据成员就是在类体内定义的变量。与其他构造数据类型如结构中的数据成员相似，唯一不同是类中数据成员有访问控制权限。对于私有和保护类数据成员，类体外一般不得直接访问，而公有数据成员则可为类外代码访问。

对私有数据成员值的访问可通过类体内赋值函数和取值函数进行。它们都声明为公有的成员函数，赋值函数对私有数据成员赋值，而取值函数则获得私有数据成员的当前值。

2. 成员函数

成员函数是在类体内声明的函数，对其所属的类是局部的，只存在于类体内。成员函数可以声明为公有的、私有的和受保护的，并可通过类本身和类对象调用。

每个类都有一个或多个公有成员函数作为外部访问该类的接口，类外元素正是通过该接口来使用这些函数所实现的功能。私有的成员函数仅能在类体内调用，但其派生类可以访问。

成员函数可分为外联函数和内联函数。

外联函数是在类声明中只给出原型，在类的外部才进行定义。如：

```
class TPoint {
    private:
        int x,y;
    public:
        void SetPoint(int,int);
        int GetX();
int GetY();
};
//-----
void TPoint::SetPoint(int a, int b) //外部定义成员函数 SetPoint()
{x=a;y=b;}
int TPoint::GetX()    //外部定义成员函数 GetX()
{return x;}
int TPoint::GetY()    //外部定义成员函数 GetY()
{return y;}
```

内联函数是指在类体内定义的成员函数，即其函数体放在类体内。如果在外联函数定义部分的函数头前加上关键字 **inline** 也可将其变为内联函数。前一个例子可以改为如下形式：

```
class TPoint {
    private:
        int x,y;
    public:
        void SetPoint(int,int);
        int GetX();
int GetY();
};
//-----
inline void TPoint::SetPoint(int a, int b) //定义内联函数 SetPoint()
{x=a;y=b;}
inline int TPoint::GetX()    //定义内联函数 GetX()
{return x;}
inline int TPoint::GetY()    //定义内联函数 GetY()
{return y;}
```

内联函数在被调用之前必须进行完整的定义, 否则编译器无法知道应该插入什么代码, 在调用内联函数的地方用内联函数体的代码来替换调用语句, 这样加快了代码的执行, 减少了调用时的系统开销。

3. 静态类成员

静态成员是为了解决数据共享问题而提出的, 包括静态数据成员和静态成员函数。静态数据成员在一个类中被说明为 `static`, 可以实现多个对象之间的数据共享, 并且不会破坏类隐藏的原则, 保证了安全性。静态数据成员是类的所有对象共享的成员, 而不是为某个对象专门定义的成员。

使用静态数据成员可以节省内存, 由于它是所有对象公有的, 对多个对象而言, 静态数据成员只存储一次, 供所有对象共用。此外静态数据成员还可以提高时间效率, 因为只要对其值更新一次, 就能保证所有对象存取更新后的值。

静态数据成员属于类, 而不像普通数据成员那样属于某一对象, 使用“类名: :”访问静态的数据成员。静态数据成员必须在类外的其他地方为它提供定义:

<数据类型><类名>::<静态数据成员>=<值>; 如:

```
float Super::st=0.0;
```

在访问权限许可下, 引用静态数据成员的格式为:

<类名>::<静态数据成员>

在类定义中, 前面有 `static` 说明的成员函数称为静态成员函数。静态成员函数属于整个类, 而不是属于类中的某个对象。

在静态成员函数中不能直接访问类中声明的非静态类数据成员, 而可直接引用其静态数据成员或全局变量。若要引用非静态数据成员, 则必须通过对象来引用。

4. 封装

把数据的表示方式同操作的函数组合成一个实体, 称为封装。封装隐藏了抽象的内部实现的细节, 将数据抽象的外部接口与内部实现的细节清楚地分离开。对象内部的代码和数据受到了保护, 其他对象不能直接修改。当要访问类的内部数据时, 应该通过类定义的方法即类接口进行。使用对象方法操作类中数据, 既便于程序的变动也利于减少出错机会。

封装对象具有一个明确的边界, 具有一个描述对象与其他对象相互作用的接口, 它允许类及类外其他程序部分访问和使用这些区域, 而其余大部分是不可见的。用户不知道也不需要知道类隐含有哪些数据, 故也不可能直接访问它们。事实上, 面向对象程序设计遵循了 C++ 提出的顺序、类型定义、安全性的思想。在 C++Builder 5.0 中, 类与单元这两种结构与封装、保护及变量访问有关。

C++Builder 5.0 除了 3 个存取说明符: `public`、`protected` 和 `private` 外, 又增加了两个说明符: `__published` 和 `__automated`。

5. 继承

从一个或多个先前定义的基类中继承函数和数据, 而且可能重新定义或加进新的数据行为, 这项技术叫做继承(或派生)。就是一个对象在继承了其他对象的通用属性的同时又具有一些特殊的性质。

继承性是面向对象程序设计中最重要机制。继承避免了公用代码的重复开发, 减少代码和数据冗余, 同时增加了一致性来减少模块间的接口和界面。通过 C++语言中的继承机制, 可以扩充和改善旧的程序设计以适应新的需求, 这样不仅可以节省程序开发的时间和资源, 并为未来程序设计增添了重复利用程序资源的一种途径。

在向新类中添加新的成员之前, 父类(或基类)中的所有属性和方法已经存在于子类(或派生类)中。继承时, 只需在类的定义开始部分说明新添的内容和其基类即可。如:

```
class TMyObject : public TObject
{
    public:
        __fastcall TMyObject() : TObject() {}
};
```

以上这段代码中的 TMyObject 看起来很简单, 却包含了 TObject 对象的许多方法。

从继承源上分, 继承分为单继承和多继承。

单继承是指每个派生类只直接继承了一个基类的特征。而多继承是指多个基类派生出一个派生类的继承关系, 派生类直接继承了不少一个基类的特征。

派生类有 3 种继承方式: 公有继承(public)、私有继承(private)和保护继承(protected)。

公有继承的特点是基类的公有成员和保护成员作为派生类的成员时, 它们均保持原有的状态, 而其私有成员在派生类中仍然是私有的。

私有继承的特点是基类的公有成员和保护成员都将是派生类的私有成员, 并不能被这个派生类的子类访问。

保护继承的特点是基类的公有成员和保护成员都作为派生类的保护成员, 并且只能被它的派生类成员函数或友元函数访问, 基类的私有成员仍然是私有的。

2.2.4 多态性和虚函数

多态性也是面向对象程序设计的重要特征。它与前面讲述的封装性和继承性构成了面向对象程序设计的 3 大特征。

多态性是指同样的消息被不同类型的对象接收时将导致多种不同的行为。它通常涉及到两个相关的类。这两个类包含相同的方法标识符(即方法名、参数与返回值都相同), 但对该方法的实现不一样。多态性的实现主要有重载、动态联编和虚函数。动态联编是在虚函数的基础之上实现的。

1. 重载

重载一般包括函数重载和运算符重载。函数重载实际上就是指一个标识符可以同时用于为多个函数命名。具体来说，在 C++ 中允许在相同作用域内以相同标识符定义几个不同实现的函数，可以是成员函数，也可以是一般函数。为了排除多义性，定义这种重载函数时要求函数的参数要么至少有一个类型不同，要么个数不同。编译器在选择重载函数时只考虑函数参数表，这就是要靠参数表中参数个数或参数类型的不同选择实现。运算符重载就是一个运算符可同时用于多种运算，相同名字的运算符在不同的场合表现出不同的行为。在 C++ 中，通过重新定义运算符，使其能用于特定类的对象执行特定的功能。如，通过对运算符+、-、*、/的重载，使它们可以完成分数、复数等扩充类的对象的加、减、乘、除运算操作。

2. 动态联编

联编是指计算机程序自身彼此关联的过程。按照联编进行的阶段不同，有两种不同的联编方法：静态联编和动态联编。

静态联编是指在程序编译阶段(运行前)所进行的联编，又称为前期联编或静态束定。静态联编要求在程序编译时就知道调用函数的全部信息，函数的调用速度快、效率高。

与静态联编相对而言，动态联编是在程序运行时进行的联编，这种联编又称后期联编或动态束定。这种联编方式，一直要到程序运行时才能确定调用哪个函数，提供了更好的灵活性和易维护性。

3. 虚函数

虚函数是动态联编的基础，也是重载的另一种表现形式。虚函数是成员函数，而且是非 static 的成员函数。其说明方法如下：

```
virtual<类型说明符><函数名>(<参数表>);
```

其中，由关键字 virtual 说明的函数称为虚函数，指示编译器在运行时确定所要调用的函数，即对该成员函数调用采取动态联编方式。动态联编只能通过指针或引用标识对象来操作虚函数。如果采用一般类型的标识对象来操作虚函数，则采用静态联编方式调用虚函数。

下面的 print() 函数定义为虚函数，能实现动态调用的功能。

```
Class DrawObject{
Public:
    virtual void print();
};
void DrawObject::print()
{
    cout<<"画一个对象。" <<endl;
}
```


4. 纯虚函数

有时，基类只表示一种抽象的概念，不能对虚函数给出有意义的实现，而将其说明为纯虚函数，其具体实现由该基类的派生类去完成。

纯虚函数是一类在类中说明的特殊的虚函数，其说明格式只是在虚函数后面添加关键字“=0”即可，即：

```
class<类名>
{
virtual  <类型说明符>  <函数名>(<参数表>)=0;
.
.
.
};
```

下面是一个使用纯虚函数的例子。

```
#include <iostream.h>
class circle{
    protected:
        int r;
    public:
        void setr(int x) {r=x;}
        virtual void show()=0;
};
class area:public circle{
    public:
        void show()    //重新定义虚函数 show()
        { cout<<"area is "<<3.14*r*r<<endl;}
};

void main()
{
    circle *ptr;
    area A;
    A.setr(10);
    ptr=&A;
    ptr->show();
}
```

程序运行结果如下：

```
area is 314
```

2.3 VCL 的 C++语言支持

C++Builder 5.0 增强了 C++ 的类范围，定义了对类的访问说明。

1. __published

`__published` 范围修饰词允许 C++Builder 在设计时访问组件属性。只有从 TObject 派生的类能有 `__published` 节。这个新的访问限制允许与 `public` 相同的访问权限。用 `__published` 限制的类成员在运行时是可用的(就像 `public` 一样)在设计时也是可用的。`__published` 关键字指定该节的属性在对象观察器中被显示。在 C++Builder 5.0(以下简称为 C++Builder)窗体中的组件都会自动在 `__published` 中声明。

```
class TForm1 : public TForm
{
    __published:    // C++Builder 集成环境管理的组件
        TLabel *Label1;
        TEdit *Edit1;
        TMainMenu *MainMenu1;
        TButton *Button1;
private:    // 用户声明
public:    // 用户声明
    __fastcall TForm1(TComponent* Owner);
};
```

需要向一个窗体类中加入成员变量或方法时，应在 `private` 或 `public` 段中操作，通常情况下，不能手工修改窗体类的 `__published` 段，所有带声明的组件都会对应于窗体的 DFM 文件，因此手工修改可能造成两个文件的不匹配。

2. __automated

`__automated` 与 `public` 或 `__published` 具有相同的访问权限，不同之处是，它用于创建类的 OLE 信息，通常用于创建 OLE 自动服务器。当在 C++Builder 中建立 OLE 自动服务器时，编译器会为成员函数及属性产生 OLE 自动信息，并声明在 `__automated` 段中。

3. __property

该关键字在类声明中包含属性。通过在 C++Builder 集成环境中的对象观察器中设置属性值了解属性的含义。

4. __fastcall

在 C++Builder 的窗体方法中会自动产生 `__fastcall`，这是编译器指示方法参数通过寄

寄存器传输,而不是通过堆栈。这是所有窗体类方法所必须的。如果加入自定义的方法,也需要遵循这一要求。

5. #pragma 资源

#pragma 将文件标识为窗体类文件。如果想向窗体源文件中加入任何变量(不是窗体类本身),应该在#pragma 语句之后加上声明。例如:

```
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
TUserClass *UserVar; //用户在这里添加变量
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
//-----
```

6. AnsiString

C 语言和 C++语言没有字符串数据类型。C 语言中的字符串是字节数组,没有字符串运算符。字符串操作是利用库函数完成的。VCL 库定义了一个 AnsiString 类,它使交换多字节字符变得容易。

在 AnsiString 类中,字符串的表示方法在很大程度上不同于 C/C++字符串。基本的 C/C++字符串仅被表示为空字节标记字符串结尾的字符字节数组。一旦它被创建,没有内在扩大字符串的方法,并且标准库函数不能交换包含有空字符的字符串。

AnsiString 类包含一个指针,记录所容纳的真正的字符串。这个记录的主要元素是字符串的内容、字符串长度和记录参考数目。通过使用字符串数据指针,使运用多个完全相同的字符串的 AnsiString 对象成为可能。

最常用的 AnsiString 构造函数如下:

AnsiString(const char* src);	转换 C 字符串成 AnsiString
AnsiString(char src);	转换一个 C 字符成 AnsiString
AnsiString(int src);	转换一个整数成 AnsiString
AnsiString(double src);	转换双精度数成 AnsiString

AnsiString 类定义了一些操作数和使字符串进行简单转换的函数。表 2.6 列出了符合标准 C 库函数的 AnsiString 类的重要操作数和函数。其中许多函数应用于 AnsiString 应用程序。

表 2.6 AnsiString 类的重要操作数和函数

函 数	描 述	等价的 C 库函数
=	赋值	strcpy
+=	连接	strcat
+	连接	None
==, !=, <, <=, >, >=	比较	strcmp
[]	字符引用	[]操作
c_str	转换成 C 类型	None
Delete ,	删除子字符串	None
Insert	插入子字符串	None
Length	字符串的长度	strlen
LowerCase	转换成小写	strlwr
Pos	确定子字符串的位置	strstr
SetLength	设置字符串的长度	None
SubString	提取字符串	None
ToInt	转换成整数	sprintf
ToDouble	转换成双精度	sprintf
Unique	确保 String 变量惟一	None
UpperCase	全部变成大写	strupr

用 AnsiString 类的子字符串时要注意一些事情。首先, AnsiString 类对照 C/C++ 字符串, 认为字符串的起始位置是 0。其次, 不像其他 C++ 类库的字符串, 不能通过分配子字符串一个值而更新字符串。

AnsiString.c_str 成员函数返回一个指向由 String 表达式参考的真正字符串数据的指针。可以在任何使用 C 类型字符串的地方使用它, 如对 C 库或 WIN32 API 函数的调用。

7. Variant

Variant 类源于 Delphi 和早期 Borland C++ 中的 variant 类型。variant 对象是不定类型的变量, 可以自动表现为不同的类型, 可以是布尔型、日期/时间型、整型、OLE 自动对象、实型以及字符串型, 虽然方便, 但效率非常低, 通常用于管理 OLE 自动服务器对象。

8. Currency

Currency 类实现了一个特殊的浮点指针类型，用于支持 Currency 类型。Currency 值的范围是-922 337 203 685 477.580 8~922 337 203 685 477.580 7。当处理 Currency 类型时，只能使用在 Currency 类中声明的操作符。编译器将忽略任何重载的操作符。

9. Set

Set 类实现了一个数据结构集合。Set 集合只能包含有序值，如字符、整数和枚举类型等。最小值必须大于 0，最大值必须小于 255。

创建一个 Set 集合，必须使用以下 3 个模板参数：Set 类型、Set 元素最小值、Set 元素最大值。

10. OPENARRAY

OPENARRAY 宏(在 sysopen.h 中定义)可作为使用命名的变量把临时动态数组传递给按值获取动态数组的函数的另一个选择。在使用时，必须提供开放数组的类型及附加的值列表(可以是常数、变量)，最多可有 19 个值被传递。如果需要一个更大的数组，必须定义显式变量。VCL 函数也可使用常数数组作为参数类型。然而，在常数数组中的值可以是混合类型。

11. EXISTINGARRAY

EXISTINGARRAY 宏(在 sysopen.h 中定义)允许为动态数组的地方传递一个存在的数组。此数组必须保证正确的类型。此数组作为参数时，可以作为参考来传输(也就是说可以修改数组)，或作为值来传输(会自动创建一个备份，并且可以修改备份)。

12. VCL 异常

这些将在以后章节中单独论述。见 8.2 节。

2.4 CHESS 项目中的数据类型及类的定义

2.3 节的说明可能比较抽象，不大好懂，现结合 CHESS 项目具体说明。这些数据类型

通常放在包含文件里，既有全局数据类型，又有特殊类的数据类型，它们各有千秋，起着独特的作用，仔细研究，方可掌握其中的涵义。

2.4.1 CDefines.h 中的数据类型

首先介绍 CDefines.h 中的数据类型，其中定义了一些全局常量及公用数据类型，主要供搜索棋路树时使用，如下所示：

```

const int crWaitCursor = 1;           //等待光标常量
const BACK = -256;                    //回退步数
const MAXPLY = 45;                    //最大搜索深度
const LOSEVALUE = 0x7D00;             //认输的打分值
const MATEVALUE = 0x7B60;             //被将死的打分值
const DEPTHFACTOR = 0x80;             //深度加权因子
const SQUARE_SIZE = 36;              //棋盘格子的大小，以像素计
const MYFRAMESIZE = 3;                //边框的大小
#define MAXINT 32767                  //最大整数宏定义

typedef int MAXTYPE; /* 估值类型 */
enum PIECETYPE {empty, king, rook, gunner, knight, assist, bishop, pawn};
//枚举类型
enum COLORTYPE {red, black}; //红方、黑方
/*以下是几个别名定义*/
typedef int INDEXTYPE;
typedef int SQUARETYPE;
typedef int EDGESQUARETYPE;
typedef short DEPTHTYPE;
struct MOVETYPE
{
    int newpos1, oldpos; /* newpos1 和 oldpos 格子 */
    short spe;           /* 指示特殊移动 */
    PIECETYPE movpiece, content; /* 移动棋子，吃掉的棋子 */
};

struct BOARDTYPE
{
    PIECETYPE piece;
    COLORTYPE color;
    short index;
    unsigned char attacked;
}; //记录棋盘的数据类型

typedef MOVETYPE PATHTYPE[MAXPLY]; /* 主要棋路变化类型 */
typedef unsigned char *LIBTYPE;    /* 棋谱库 */

struct CHESSCLOCKTYPE
{

```

```
int hour, minute, second, sec100;
int totalhours, totmins, totalsecs, tots100;
int number;
double totaltime;
}; //记录棋钟的数据类型
/*以下是几个别名,主要是为方便用户使用*/
typedef char DIRTYTYPE;
typedef short SETOFPIECE;
typedef unsigned char SIXTYTYPE;
typedef unsigned char REPEATTYPE;

struct NODEVAL
{
    unsigned long nodebase, nodeoffset;
}; //棋路树的节点

enum LEVELTYPE { normal, fullgametime, easygame, infinite, plysearch,
                matesearch, matching }; //级别的枚举类型

struct BOARDIDTYPE
{
    PIECETYPE piece;
    COLORTYPE color;
}; //棋盘的标识

struct PIECETABLE
{
    SQUARETYPE isquare;
    PIECETYPE ipiece;
}; //棋子数目表

struct ATTACKTABTYPE
{
    SETOFPIECE pieceset;
    int direction; /* 棋子的移动方位 */
}; //受到攻击时的记录
```

CDefines.h 中的数据类型主要记录棋盘、棋子数目、棋子的受攻状态、移动状态、棋路树的节点、棋钟等，供形成搜索棋路树和评估局面时使用。

2.4.2 TEditForm 类

在 TEditForm(编辑窗体)类包含文件 EditForm2.h 中定义了 TEditForm 类。它是经过 C++Builder 自动生成后再由用户添加一些数据成员而形成的。当对棋盘进行编辑时需要用到这些数据类型。TEditForm 类的定义如下：

```
//-----
class TEditForm : public TForm
{
    friend class TMainForm;    //定义友元
    __published:    // C++Builder 集成环境管理的组件
        void __fastcall FormPaint(TObject *Sender);
        void __fastcall FormMouseUp(TObject *Sender, TMouseButton Button,
            TShiftState Shift, int X, int Y);
    private:    // 用户声明
        int SelectedItem;    //标识选中的棋子
        RECT EditBarRect;
        RECT EditRect;
        int GetSelectedItem()
        {
            return SelectedItem;
        }
        void SetEditWindowBk();    //设置编辑窗体的背景

    public:    // 用户声明

        __fastcall TEditForm(TComponent* Owner);
    protected:
        void __fastcall Loaded();
        void __fastcall CreateParams(TCreateParams& Params);
    };
//-----
```

在编辑窗体类 TEditForm 中声明了窗体重画函数、鼠标释放回调函数、设置窗体背景函数等，同时友元主窗体类可以存取该类的数据。程序运行时，鼠标释放能获取选中的棋子，并重画编辑窗体。

2.4.3 TColorDlg 类

在 TColorDlg(颜色对话框)类包含文件 ColorDlog.h 中定义了 TColorDlg 类。它是经过 C++Builder 自动生成后再由用户添加一些数据成员而形成的。当需要改变棋盘的颜色时要用到这些数据类型。TColorDlg 类的定义如下:

```
//-----  
class TColorDlg : public TForm  
{  
    __published:    // C++Builder 集成环境管理的组件  
        TBevel *Bevel2;  
        TLabel *Label5;  
        TLabel *Label6;  
        TLabel *Label7;  
        TLabel *Label8;  
        TLabel *Label9;  
        TLabel *Label11;  
        TLabel *Label10;  
        TLabel *Label11;  
        TLabel *Label12;  
        TLabel *Label13;  
        TLabel *Label14;  
        TBevel *Bevel1;  
        TEdit *EditRed;  
        TEdit *EditGreen;  
        TEdit *EditBlue;  
        TLabel *Label2;  
        TLabel *Label3;  
        TLabel *Label4;  
        TLabel *ColorSelection;  
        TScrollBar *ScrollBar_Red;  
        TScrollBar *ScrollBar_Green;  
        TScrollBar *ScrollBar_Blue;  
        TButton *OKBtn;  
        TButton *CancelBtn;  
        void __fastcall OkClick(TObject *Sender);  
        void __fastcall CancelBtnClick(TObject *Sender);  
        void __fastcall ScrollBarChange(TObject *Sender);  
        void __fastcall ColorDlgShow(TObject *Sender);  
};
```

```

        void __fastcall EditExit(TObject *Sender);
private:    // 用户声明
    BYTE SqColors[3]; //记录红、绿、蓝颜色值
    bool GetColorValue(TObject *Sender);
public:    // 用户声明
    __fastcall TColorDlg(TComponent* Owner);
};
//-----

```

颜色对话框运行时,拖动滚动条会引起编辑框中的数字作相应变化;当退出编辑窗体时如果输入的数字正确,则滚动条做相应的拖动,否则就抛出异常,单击 Cancel 按钮将关闭对话框,单击 OK 按钮将重新设置红绿蓝的颜色值。

2.4.4 TInfoForm 类

在 TinfoForm(信息窗体)类包含文件 InfoForm3.h 中定义了 TInfoForm 类。它是经过 C++Builder 自动生成后再由用户添加一些数据成员而形成的。对弈时,程序的状态信息要用到这些数据类型进行显示。TInfoForm 类的定义如下:

```

//-----
class TInfoForm : public TForm
{
    __published:    // 集成环境管理的组件
        TStaticText *BestPath;
        TStaticText *Message;
        TStaticText *Timer;
        TStaticText *RedInfo;
        TStaticText *BlackInfo;
        TStaticText *Value;
        TStaticText *Nodes;
        TStaticText *Seconds;
        TStaticText *Depth;
        TStaticText *Level2;
        TPanel *Panell;
        TLabel *Label1;
        TStaticText *Color1;
        void __fastcall InfoPaint(TObject *Sender);
        void __fastcall FormShow(TObject *Sender);
private:    // 用户声明

```

```
public:    // 用户声明
    RECT InfoRect; //信息矩形
    void SetTurnText(AnsiString text)
    {
        Color1->Caption="轮流: "+AnsiString(text);
    }
    void SetRedInfoText(AnsiString text)
    {
        RedInfo->Caption="红方: "+AnsiString(text);
    }
    void SetBlackInfoText(AnsiString text)
    {
        BlackInfo->Caption="黑方: "+AnsiString(text);
    }
    void SetTimerText(AnsiString text)
    {
        Timer->Caption="时间: "+AnsiString(text);
    }
    void SetLevelText(AnsiString text)
    {
        Level2->Caption="级别: "+AnsiString(text);
    }

    void SetValueText(AnsiString text)
    {
        Value->Caption="值: "+AnsiString(text);
    }
    void SetNodeText(AnsiString text)
    {
        Nodes->Caption="节点: "+AnsiString(text);
    }
    void SetSecondsText(AnsiString text)
    {
        Seconds->Caption="步/秒: "+AnsiString(text);
    }
    void SetDepthText(AnsiString text)
    {
        Depth->Caption="深度: "+AnsiString(text);
    }
```

```

void SetBestPathText(AnsiString text)
{
    BestPath->Caption=text;
}

void SetMessageText(AnsiString text)
{
    Message->Caption=text;
}

void DrawInfoFrame(HDC); //绘制三维信息矩形边框
void Reset();
void IterReset();
void SetInfoWindowBk();
__fastcall TInfoForm(TComponent* Owner);
protected:
    void __fastcall Loaded();
    void __fastcall CreateParams(TCreateParams& Params);
};
//-----

```

在 TInfoForm 类中声明了窗体重画函数、窗体显示函数、设置窗体背景函数等。程序对弈时，将显示轮流对弈状态、所用弈棋时间、用户选择的对弈级别、局面的评估值、搜索树的节点数、搜索量的大小、搜索的深度、提示最好的路线、是否违反棋规等。

2.4.5 TMainForm 类

在 TmainForm(主窗体)类包含文件 MainForm1.h 中定义了 TMainForm 类。它是经过 C++Builder 自动生成后再由用户添加一些数据成员而形成的。当对弈时，程序与用户的交互状态信息要用到这些数据类型进行通讯。TMainForm 类的定义如下：

```

//-----
class TMainForm : public TForm
{
    __published: // C++Builder 集成环境管理的组件
        TMainMenu *TChessMenu;
        TMainMenu *TChessEditMenu;
        TMainMenu *TChessThinkMenu;
        TMenuItem *N3;
        TMenuItem *Game;
        TMenuItem *About;
}

```

```
TMenuItem *Hint1;
TMenuItem *Exit1;
TMenuItem *RestoreGame1;
TMenuItem *NewChessGame1;
TMenuItem *SaveGame1;
TMenuItem *Pass;
TMenuItem *N1;
TMenuItem *N2;
TMenuItem *Edit;
TMenuItem *RedoUndo;
TMenuItem *UndoMove;
TMenuItem *MovePiece;
TMenuItem *Level1;
TMenuItem *Ply1;
TMenuItem *Infinite;
TMenuItem *Easy;
TMenuItem *Mate;
TMenuItem *MoveTime;
TMenuItem *TotalTime;
TMenuItem *Matching;
TMenuItem *Options;
TMenuItem *Sound;
TMenuItem *MouseMove;
TMenuItem *Demo;
TMenuItem *TwoPlayer;
TMenuItem *N30;
TMenuItem *PieceValues;
TMenuItem *MouseDown;
TMenuItem *MouseSelect;
TMenuItem *Display;
TMenuItem *BestPath;
TMenuItem *Reverse;
TMenuItem *Colors;
TMenuItem *Done;
TMenuItem *Clear;
TMenuItem *Cancel;
TMenuItem *Turn;
TMenuItem *RedTurn;
TMenuItem *BlackTurn;
```

```
TMenuItem *Stop;
TOpenDialog *OpenDialog1;
TSaveDialog *SaveDialog1;
TTimer *Timer1;
TMenuItem *SaveGameAs1;
/*以下是主窗体事件回调函数*/
void __fastcall FormCreate(TObject *Sender);
void __fastcall FormShow(TObject *Sender);
void __fastcall FormPaint(TObject *Sender);
void __fastcall FormMouseDown(TObject *Sender, TMouseButton Button,
    TShiftState Shift, int X, int Y);
void __fastcall NewChessGame1Click(TObject *Sender);
void __fastcall RestoreGame1Click(TObject *Sender);
void __fastcall SaveGameAsClick(TObject *Sender);
void __fastcall SaveGame1Click(TObject *Sender);
void __fastcall AboutClick(TObject *Sender);
void __fastcall MovePieceClick(TObject *Sender);
void __fastcall ClearClick(TObject *Sender);
void __fastcall CancelClick(TObject *Sender);
void __fastcall DemoClick(TObject *Sender);
void __fastcall InfiniteClick(TObject *Sender);
void __fastcall MateClick(TObject *Sender);
void __fastcall Ply1Click(TObject *Sender);
void __fastcall TwoPlayerClick(TObject *Sender);
void __fastcall ReverseClick(TObject *Sender);
void __fastcall UndoMoveClick(TObject *Sender);
void __fastcall RedoUndoClick(TObject *Sender);
void __fastcall EasyClick(TObject *Sender);
void __fastcall DoneClick(TObject *Sender);
void __fastcall Hint1Click(TObject *Sender);
void __fastcall PassClick(TObject *Sender);
void __fastcall RedTurnClick(TObject *Sender);
void __fastcall BlackTurnClick(TObject *Sender);
void __fastcall ColorsClick(TObject *Sender);
void __fastcall PieceValuesClick(TObject *Sender);
void __fastcall BestPathClick(TObject *Sender);
void __fastcall MouseDragClick(TObject *Sender);
void __fastcall MouseSelectClick(TObject *Sender);
void __fastcall FormMouseMove(TObject *Sender, TShiftState Shift,
```

```

        int X, int Y);
    void __fastcall SoundClick(TObject *Sender);
    void __fastcall FormDestroy(TObject *Sender);
    void __fastcall MoveTimeClick(TObject *Sender);
    void __fastcall TotalTimeClick(TObject *Sender);
    void __fastcall MatchingClick(TObject *Sender);
    void __fastcall Exit1Click(TObject *Sender);
    void __fastcall FormCloseQuery(TObject *Sender, bool &CanClose);
    void __fastcall Timer1Timer(TObject *Sender);
private:    // User declarations

    TForm* ChildForms[3];

    void SetMainWindowBk(); //调用位图文件, 绘制主窗体背景
    void SaveGameAs(); //保存棋盘
    void EndGame(); //结束游戏
    void DoEdit(); //处于编辑状态
    void ComputersTurn(); //轮到机器下
    void UnCheckLevelMenu(LEVELTYPE); //动态地改变菜单
    void NormalSetup();
    void CheckLevelMenu(LEVELTYPE);
/*消息映射, 响应 PostMessage 或 SendMessage 函数*/
    MESSAGE void ELError(TMessage &Msg);
    MESSAGE void FileNew(TMessage &Msg);
    MESSAGE void Exit(TMessage &Msg);
protected:

public:    // User declarations
    __fastcall TMainForm(TComponent* Owner);
    enum TURNTYPE { player, computer };
    bool blNewChessGame; // 是否开始新的游戏
    char *FileName; // 上次存储的文件名
    RECT MainWndRect; // 主窗体的矩形范围
    RECT InfoAreaRect; // 信息窗体的矩形范围
    TURNTYPE WhoseTurn; // 轮到是用户还是计算机?
    SQUARETYPE MoveStartSquare; // 开始移动的方格
    SQUARETYPE MoveEndSquare; // 结束移动的方格
    COLORTYPE CurPlayer; // 当前的用户
    bool GotStartSquare; // 用户选择了一个活动棋子进行移动?

```

```

bool Modified;           // 棋盘一直处于编辑模式是否已改变?
bool EditingChessBoard;  // 用户是否在编辑棋盘?
bool Dragging;           // 处于拖动模式?
BOARDTYPE *SaveChessBoard;

int MouseX,MouseY; //记录鼠标位置
/*
重载消息处理方法
*/
BEGIN_MESSAGE_MAP
    MESSAGE_HANDLER(IDM_ERROR, TMessage, ELError);
    MESSAGE_HANDLER(IDM_FILENEW, TMessage, FileNew);
    MESSAGE_HANDLER(IDM_EXIT, TMessage, Exit);
END_MESSAGE_MAP(TForm);
};
//-----

```

在 TMainForm 类中声明了窗体创建函数、显示函数、重画函数、及大量菜单的回调函数等，程序运行时接收用户的输入信息，根据新的输入设置棋盘背景、调整对弈状态等。

2.4.6 TPieceValueDlg 类

在 TPieceValueDlg(棋子值对话框)类包含文件 PieceValueDlog.h 中定义了 TPieceValueDlg 类。它是经过 C++Builder 自动生成后再由用户添加一些数据成员而成的。当对弈时，如果用户需要改变棋子值时要用到这些数据类型。TPieceValueDlg 类的定义如下：

```

//-----
class TPieceValueDlg : public TForm
{
__published:    // C++Builder 集成环境管理的组件
    TButton *OKBtn;
    TButton *CancelBtn;
    TButton *Default;
    TBevel *Bevell;
    TScrollBar *Srook;
    TScrollBar *Sknight;
    TScrollBar *Sbishop;
    TScrollBar *Sassist;
    TScrollBar *Sgunner;
    TScrollBar *Spawn;
    TEdit *Eassist;

```



```
TEdit *Ebishop;
TEdit *Eknight;
TEdit *Erook;
TEdit *Egunner;
TEdit *Epawn;
TLabel *Label3;
TLabel *Label4;
TLabel *Label5;
TLabel *Label6;
TLabel *Label7;
TLabel *Label8;

/*事件驱动函数*/
void __fastcall ScrollBar_Change(TObject *Sender);
void __fastcall EditExit(TObject *Sender);
void __fastcall FormShow(TObject *Sender);
void __fastcall OKBtnClick(TObject *Sender);
void __fastcall CmDefault(TObject *Sender);
void __fastcall CancelBtnClick(TObject *Sender);
private:    // 用户声明
    int Values[6]; //记录士、相、马、车、炮、兵的值
    bool GetColorValue(TObject *Sender);
public:     // 用户声明
    __fastcall TPieceValueDlg(TComponent* Owner);
};
//-----
```

在棋子值对话框运行时，拖动滚动条就会引起编辑框中的数字作相应变化，当退出编辑框时如果输入的数字正确，则滚动条做相应的拖动，否则就抛出异常，单击 **Cancel** 按钮将关闭对话框，单击 **Default** 按钮将设置每个棋子的值为默认状态，单击 **OK** 按钮将重新设置每个棋子的值。

第3章 集成开发环境

C++Builder 是一个面向对象的可视化编程平台，应用于 32 位 Windows 应用程序的快速高效开发。集成开发环境英文简称为 IDE，它提供了所有必需的用于设计、开发、试验、调试和发布应用程序的工具。

双击 C++Builder 5.0 的图标或者在【开始】|【程序】菜单中选择 C++Builder 5.0 应用程序选项，就可以进入 C++Builder 5.0 的开发环境，如图 3.1 所示。

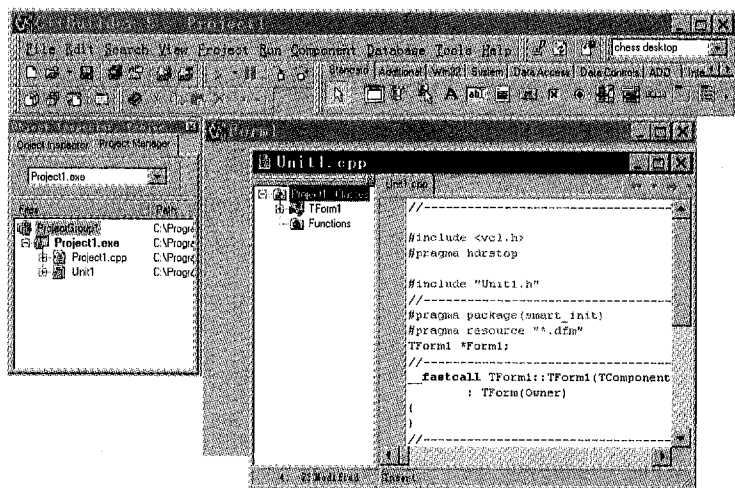


图 3.1 集成开发环境

C++Builder 的开发环境，主要包括窗体(Form)、菜单栏(Menu Bar)、工具栏(Speed Bar)、组件栏(Component palette)、对象观察器(Object Inspector)、项目管理器(Project Manager)、代码编辑器(Code Editor)、图像编辑器(Image Editor)、桌面工具栏(Desktops toolbar)。下面简要叙述各个部分的功能。

3.1 窗体

窗体是 C++Builder 5.0 程序设计中重要的工具，大多数应用程序的开发都要用到窗体，有的应用程序甚至需要多个窗体，这是因为窗体是应用程序最终面向用户的窗口，各种图形、图像、数据都是通过窗体或窗体中的组件进行输入输出的。

在 C++Builder 5.0 中新建一个窗体有两种方法：

- 选择 File | New Application 命令，程序将自动新建一个空白窗体。
- 选择 File | New Form 命令，在原有的项目上增加一个空白窗体。

如图 3.2 所示,设计开始时的窗体是一个空的窗体。我们开发窗体的过程包括在窗体上放置组件设置组件属性,最后为窗体事件或组件事件编写事件处理过程。

当我们保存新建的窗体文件时,程序将弹出一个对话框,要求给出保存的路径。每个窗体都对应一个.cpp、.dfm 和.h 文件,三者成为一个窗体单元,它们位于同一目录下,而且是一次性生成的。其中.cpp 文件主要记录函数定义,.dfm 是 C++Builder 特有的二进制文件,需要借助 C++Builder 才能察看,.h 文件主要记录了数据结构(如窗体类等)。

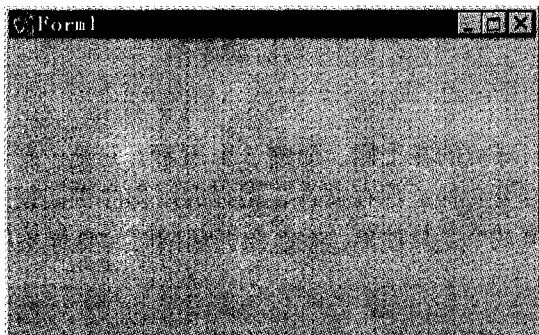


图 3.2 空的窗体

3.2 菜单栏

菜单栏位于窗口的最上面如图 3.3 所示。它提供了 C++Builder 5.0 的全部操作功能,包括 File、Edit、Search、View、Project、Run、Component、Database、Tools 和 Help 共 10 个主菜单。只要用户对 Windows 或者某一种面向对象的语言熟悉,就能很快掌握 C++Builder 5.0 的菜单功能。其功能如表 3.1 所示。



图 3.3 菜单栏

表 3.1 菜单栏列表

菜单名称	功 能
File	文件的打开、关闭、新建、删除、保存、打印等
Edit	常用的编辑命令,如:撤消、恢复、剪切、拷贝、粘贴、删除、组件的对齐等
Search	文本的查找、替换等
View	查看项目、对象、类、断点、组件、单元、工具、窗体等对象
Project	添加项目、类库,查看项目源文件、删除已有的项目、类库及对项目文件的编译链接等
Run	项目文件的运行、调试等
Component	对组件库的管理、安装、定制等

续表

菜单名称	功 能
Database	与本地或远程数据库相关的操作命令组
Tools	对编程环境进行定制的命令组
Help	在线帮助命令

3.3 工具栏

工具栏位于集成开发环境的左上侧,如图 3.4 所示。它把一些常用菜单命令用工具栏上的快捷按钮实现,如打开文件、保存文件、运行应用程序等。使用工具栏比使用菜单更快捷,当你将鼠标停留至某个工具片刻,该按钮突起同时会有提示出现。

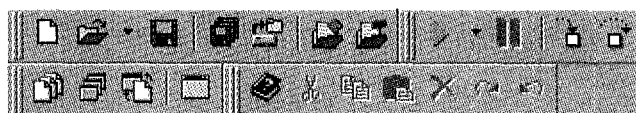


图 3.4 工具栏

在安装 C++Builder 5.0 后的默认工具栏中没有 Cut、Redo、Undo 等快捷工具,用户可自己进行添加。方法是:

- (1) 选择 View | Toolbars | Customize 命令或在某个工具栏上单击,同样弹出 Customize 对话框。
- (2) 在弹出的对话框中打开 Commands 选项卡,如图 3.5 所示。
- (3) 用鼠标按住所选取的工具不放,将其拖至主界面的工具栏中的适当位置,松手即可。要从工具栏上去掉某个按钮,只要把要去掉的按钮拖离工具栏即可。

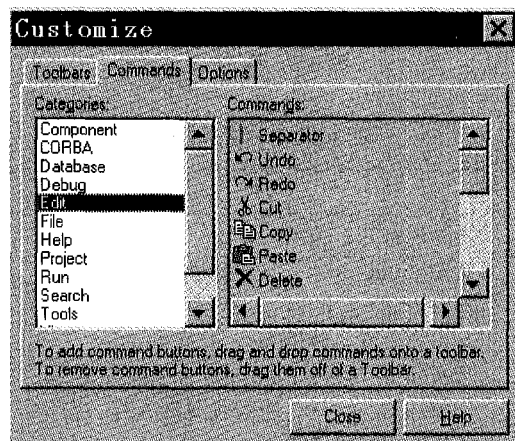


图 3.5 自定义工具栏

3.4 组件栏

组件栏是 C++Builder 5.0 的可视化组件库(VCL), 是程序的主要工作区。它位于集成开发环境的右上侧。C++Builder 5.0 专业版共有 19 个选项卡, 包括 Standard、Additional、Win32、System、Data Access、Data Controls、ADO、InterBase、Midas、Internet Express、Internet、FastNet、Decision Cube、QReport、Dialogs、Win3.1、Samples、ActiveX、Servers, 其中如 Midas 选项卡、InternetExpress 选项卡、Servers 选项卡是在 C++Builder 4.0 的基础上增加的。C++Builder 5.0 的组件栏共提供了 200 多个可视化组件。熟练掌握常用组件的使用方法, 再结合联机帮助, 使用组件编程将是一项轻松的任务。

1. Standard 组件选项卡

Standard 组件选项卡如图 3.6 所示, 其中各组件的功能如表 3.2 所示。



图 3.6 Standard 选项卡

表 3.2 Standard 选项卡中各组件的功能

组 件	功 能
Frames	打开一个对话框, 列出当前项目中的所有框架
MainMenu	建立窗体主菜单
PopMenu	建立弹出式菜单
Label	建立一个标签
Edit	建立一个用户单行文本编辑框
Memo	建立一个多行文本编辑器
Button	建立一个标准按钮
CheckBox	建立一个复选框
RadioButton	建立一个单选(互斥)按钮
ListBox	建立列表框, 可带滚动条
ComboBox	建立一个组合列表框
ScrollBar	建立一个滚动条
GroupBox	建立分组选项框
RadioGroup	建立单选按钮的分组选项框
Panel	建立一个可以容纳其他组件的面板
ActionList	建立动作响应列表, 可将对象的动作放在列表中进行统一处理

2. Additional 组件选项卡

Additional 组件选项卡如图 3.7 所示，其中各组件的功能如表 3.3 所示。



图 3.7 Additional 组件选项卡

表 3.3 Additional 选项卡中各组件的功能

组 件	功 能
BitBtn	建立一个位图按钮
SpeedButton	建立一个加速按钮，该按钮也可以显示位图，它通常被放在面板组件上形成工具栏
MaskEdit	建立具有指定格式的文本编辑框
StringGrid	建立按行列格式显示字符串的栅格
DrawGrid	建立一个自绘栅格，类似于 StringGrid
Image	显示.bmp、.ico 等格式图像
Shape	显示几何图形
Bevel	建立具有三维效果的直线或边框
ScrollBox	建立自动含有滚动条的矩形区域
CheckListBox	建立带有检查框的滚动列表框
Splitter	建立组件之间的分隔条
StaticText	建立静态文本框
ControlBar	建立组件容器
ApplicationEvents	截取应用程序级各种事件
Charts	建立数据图表

3. Win32 组件选项卡

Win32 组件选项卡如图 3.8 所示，其中各组件的功能如表 3.4 所示。



图 3.8 Win32 组件选项卡

表 3.4 Win32 选项卡中各组件的功能

组 件	功 能
 TabControl	建立表类型的多选项卡组件
 PageControl	建立对话框类型的多选项卡组件
 ImageList	建立一个图像列表, 每个图像都有各自的序号
 RichEdit	建立一个能够编辑多种字体、多种格式的 RTF 文件编辑器
 TrackBar	建立一个跟踪条
 ProgressBar	建立一个进度显示条
 UpDown	建立一个增减数值的上下箭头按钮
 HotKey	建立一个运行期动态生成热键输入编辑框
 Animate	建立一个无声动画显示组件, 包括未压缩的 AVI 文件和按 RLE 压缩的 AVI 剪辑
 DateTimePicker	建立一个可以输入日期和时间的列表框
 MonthCalendar	建立一个日历
 TreeView	建立一个多层次树状视图
 ListView	建立一个列表视图
 HeaderControl	建立一个列表加上表头
 StatusBar	建立一个状态栏
 ToolBar	建立一个工具条
 CoolBar	建立一个窗口化显示组件的工具栏, 段与段之间有可以拖动的分隔线
 PageScroller	建立一个可以滚动的容器

4. System 组件选项卡

System 组件选项卡如图 3.9 所示, 其中各组件的功能如表 3.5 所示。



图 3.9 System 组件选项卡

表 3.5 System 选项卡中各组件的功能

组 件	功 能
 Timer	建立一个定时器
 PictureBox	建立一个绘画框
 MediaPlayer	建立多媒体(视频和音频)播放器
 OleContainer	建立一个 OLE 容器, 用来链接或嵌入一个 OLE 对象
 DdeClientConv	建立与 DDE 服务器的连接

续表

组 件	功 能
 DdeClientItem	在 DDE 对话框中设置客户端选项
 DdeServerConv	建立与 DDE 客户的连接
 DdeServerItem	在 DDE 对话框中设置服务器端的选项

5. Data Access 组件选项卡

Data Access 组件选项卡如图 3.10 所示, 其中各组件的功能如表 3.6 所示。

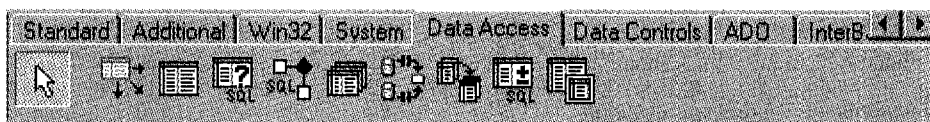


图 3.10 Data Access 组件选项卡

表 3.6 Data Access 选项卡中各组件的功能

组 件	功 能
 DataSource	数据源组件, 把数据库与数据组件的连接起来
 Table	数据表组件, 用于访问指定的数据库表格
 Query	数据查询组件, 通过 SQL 语句进行本地或远程的数据查询
 StoredProc	在服务器上执行数据库存储过程
 Database	在数据库应用程序中与数据库服务器建立连接
 Session	会话组件, 用于管理程序中多个数据库连接
 BatchMove	对数据库表结构及数据进行复制或移动
 UpdateSQL	在进行高速缓存时更新只读数据集
 NestedTable	在嵌套的数据集中检索数据

6. Data Controls 组件选项卡

Data Controls 组件选项卡如图 3.11 所示, 其中各组件的功能如表 3.7 所示。

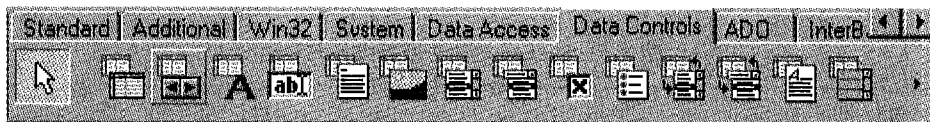


图 3.11 Data Controls 组件选项卡

表 3.7 Data Controls 选项卡中各组件的功能

组 件	功 能
 DBGrid	以行和列网格方式显示和编辑数据
 DBNavigator	用于数据库导航
 DBText	以只读方式显示当前记录的内容
 DBEdit	显示、编辑当前记录指定数据内容
 DBMemo	以备忘形式多行显示、编辑当前记录内容
 DBImage	显示数据图像
 DBListBox	以列表方式显示数据
 DBComboBox	以组合列表框方式显示数据
 DBCheckBox	以复选框方式显示数据
 DBRadioGroup	以单选框、分组框方式对数据进行操作
 DBLookupListBox	以列表方式显示查询数据列表
 DBLookupComboBox	以组合列表框方式显示查询数据列表
 DBRichEdit	以 RTF 格式显示并可编辑数据库中的数据
 DBCtrlGrid	以网格格式显示数据库数据, 各网格属性可以单独设置
 DBChart	以数据图表方式显示数据库数据

7. ADO 组件选项卡

ADO 组件选项卡如图 3.12 所示, 其中各组件的功能如表 3.8 所示。



图 3.12 ADO 组件选项卡

表 3.8 ADO 选项卡中各组件的功能

组 件	功 能
 ADOConnection	建立与 ADO 数据库的连接并提供对事务的支持
 ADOCommand	直接对 ADO 数据库执行 SQL 命令, 但并不返回结果
 ADODataSet	代表 ADO 数据库中一个或几个数据集
 ADOTable	代表 ADO 数据库中一个数据表, 可替换原来的数据表
 ADOQuery	使用 SQL 语句查询 ADO 数据库, 可替换原来的查询
 ADOStoredProc	允许应用程序访问服务器中的存储过程, 可替换原来的存储过程
 RDSConnection	连接远程的应用服务器, 该组件可以保持数据的一致性

8. InterBase 组件选项卡

InterBase 组件选项卡如图 3.13 所示, 其中各组件的功能如表 3.9 所示。



图 3.13 InterBase 组件选项卡

表 3.9 InterBase 选项卡中各组件的功能

组 件	功 能
IBTable	从单个数据库表中获取数据, 并可进行定制, 相当于 Table
IBQuery	使用 SQL 语句从 ADO 物理数据库中获取相关数据
IBStoredProc	执行一个 InterBase 数据库的存储过程, 不返回结果
IBDatabase	建立与本地数据库的连接、管理事务和提供连接参数
IBTransaction	提供对一个或多个数据库连接进行控制的各种事务
IBUpdateSQL	用于对只读数据库的更新, 相当于 UpdateSQL
IBDataSet	使用 IBDataSet 执行 SQL 语句
IBSQL	直接执行 SQL 语句, 并可高速地存取数据, 没有返回结果
IBDatabaseInfo	返回获取数据库的信息
IBSQLMonitor	监视对数据库服务器的动态查询
IBEvents	给应用程序提供一个对服务器已发生事件进行响应的方法

9. Midas 组件选项卡

Midas 组件选项卡如图 3.14 所示, 其中各组件的功能如表 3.10 所示。



图 3.14 Midas 组件选项卡

表 3.10 Midas 选项卡中各组件的功能

组 件	功 能
ClientDataSet	建立一个与数据库无关的数据集
DCOMConnection	将客户程序与远程服务器按 DCOM 方式连接
SocketConnection	将客户程序与远程服务器按 TCP/IP 方式连接
DataSetProvider	将数据编码至包中, 并发送至客户应用程序, 并接收客户端的响应
SimpleObjectBroker	从服务器程序列表选择一个服务器用于连接组件建立与该服务器的连接, 以平衡负载
WebConnection	将客户程序与远程服务器按 HTTP 方式连接

10. InternetExpress 组件选项卡

InternetExpress 组件选项卡如图 3.15 所示, 其中各组件的功能如表 3.11 所示。

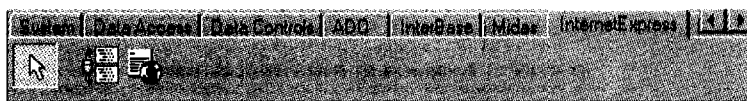


图 3.15 InternetExpress 组件选项卡

表 3.11 InternetExpress 选项卡中的组件的功能

组 件	功 能
XMLBroker	从被请求服务器上接受 XML 数据包, 并且可以接收该服务器的 HTTP 消息
MidasPageProducer	将被请求服务器上活动数据库的信息生成 HTML 页, 可以识别 XML 数据

11. Internet 组件选项卡

Internet 组件选项卡如图 3.16 所示, 其中各组件的功能如表 3.12 所示。



图 3.16 Internet 组件选项卡

表 3.12 Internet 选项卡中各组件的功能

组 件	功 能
ClientSocket	使应用程序转化为 TCP/IP 客户
ServerSocket	使应用程序转化为 TCP/IP 服务器
WebDispatcher	使一个普通的数据模块转化为 Web 模块
PageProducer	使一个 HTML 命令转化为 Web 浏览器能识别的 HTML 命令字符串
QueryTableProducer	把对数据库的查询结果转换为一组 HTML 命令
DataSetTableProducer	把数据库表格转换为 HTML 表格显示出来
DataSetPageProducer	基于输入模板, 生成 HTML 命令的字符串
CppWebBrowser	在类似网络浏览器的浏览视图中显示 HTML 文档

12. FastNet 组件选项卡

FastNet 组件选项卡如图 3.17 所示, 其中各组件的功能如表 3.13 所示。



图 3.17 FastNet 组件选项卡

表 3.13 FastNet 选项卡中各组件的功能

组 件	功 能
NMDayTime	从网络时间服务器上获取时间和日期
NMMsg	使用 TCP/IP 协议在网络上发送 ASCII 文本
NMMsgServ	接受由 TNMMsg 组件发送的消息
NMEcho	发送文本给服务器端，也可以接收回来
NMFTP	执行 FTP 协议
NMHTTP	与 Internet/Intranet 产生 HTTP 协议
NMNNTP	执行 Internet 和 Intranet 新闻服务器上的 NNTP 协议
NMStrm	与 TNMStrmServ 配套，通过 Internet 或 Intranet 发送数据流
NMStrmServ	接受由 NMStrm 组件发送过来的数据流
NMPOP3	执行 POP3(邮局)协议
NMSMTP	用于访问 SMTP 服务器，同时可以发送邮件
NMTime	从服务器上取得时间和日期
NMUDP	通过 Internet 或 Intranet 实现 UDP 协议，不区分服务器端和客户端
NMURL	解码 URL 数据为可读字符，或把字符串编码成 URL
NMUUPProcessor	加密、解密文件
Powersock	为用户提供开发多线程服务器的基类，不单独使用
NMGeneralServer	作为开发多线程网络服务程序的基类，不单独使用
NMFinger	从 Internet 的 Finger 服务器上查找某个用户信息

13. Decision Cube 组件选项卡

Decision Cube 组件选项卡如图 3.18 所示，其中各组件的功能如表 3.14 所示。



图 3.18 Decision Cube 组件选项卡

表 3.14 Decision Cube 选项卡中各组件的功能

组 件	功 能
DecisionCube	进行多种数据结构的存储，其来源通常是数据集
DecisionQuery	提供对决策数据集进行 SQL 查询，并返回结果

续表

组 件	功 能
 DecisionSource	提供多种数据表格或多种图表的当前枢纽状态表
 DecisionPivot	通过单击组件中的按钮决定重新组织多种数据或字段
 DecisionGrid	以表格的形式显示一维或多维数据
 DecisionGraph	将多种表格中的数据以图表的形式显示出来

14. QReport 组件选项卡

QReport 组件选项卡如图 3.19 所示, 其中各组件的功能如表 3.15 所示。



图 3.19 QReport 组件选项卡

表 3.15 QReport 选项卡中各组件的功能

组 件	功 能
 QuickRep	报表控制器, 产生最基本的报表
 QRSubDetail	在子数据报表中建立数据集
 QRStringsBand	在报表中建立包含字符串的区域
 QRBand	不仅可设置需要绑定的字符串, 还可规定绑定的类型, 类似于面板
 QRChildBand	把段分为几个子段
 QRGroup	建立分组区域
 QRLabel	用于在报表中显示标签文字
 QRDBText	显示数据库中字段的值
 QRExpr	计算和输出以计算表达式为主的数据信息
 QRSysData	显示系统数据(如日期、时间、报表的标题等)
 QRMemo	显示多行非数据字段的报表文本
 QRExprMemo	使用报表格式产生结果
 QRRichText	允许以 RTF 格式显示报表文字信息
 QRDBRichText	在报表上显示或加入数据库备注字段中的 RTF 格式的数据信息
 QRShape	用于在报表上输出各种简单几何图形
 QRImage	用于在报表上显示图像
 QRDBImage	用于在报表上显示数据库字段中的图像
QRCompositeReport	用于把更多的报表组织起来

续表

组 件	功 能
 QRPreview	预览报表的打印效果
 QRTextFilter	以文本格式输出报表的内容
 QRCSVFilter	以逗号隔开的文本格式输出报表的内容
 QRHTMLFilter	以 HTML 方式输出报表
 QRChart	用于在报表上输出统计图形

15. Dialogs 组件选项卡

Dialogs 组件选项卡如图 3.20 所示，其式各中各组件的功能如表 3.16 所示。



图 3.20 Dialogs 组件选项卡

表 3.16 Dialogs 选项卡中各组件的功能

组 件	功 能
 OpenFileDialog	建立打开文件对话框
 SaveDialog	建立保存文件对话框
 OpenPictureDialog	建立打开图形文件对话框，可以预览图像
 SavePictureDialog	建立保存图形文件对话框，保存时可以预览图像
 FontDialog	建立字体选择对话框
 ColorDialog	建立颜色设置对话框
 PrintDialog	建立打印对话框
 PrintSetupDialog	建立打印机设置对话框
 FindDialog	建立查找对话框
 ReplaceDialog	建立替换对话框

16. Win 3.1 组件选项卡

Win 3.1 组件选项卡如图 3.21 所示，其中各组件的功能如表 3.17 所示。

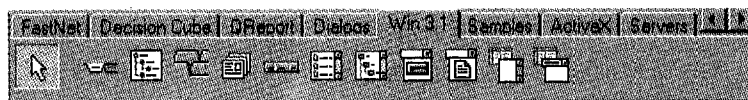


图 3.21 Win 3.1 组件选项卡

表 3.17 Win3.1 选项卡中各组件的功能

组 件	功 能
 TabSet	建立表类型的多选项卡组件，与 Notebook 配合使用
 Outline	建立树状提纲结构的显示
 TabbedNotebook	建立对话框类型的多选项卡组件
 NoteBook	建立笔记本类型的多选项卡组件，与 TabSet 配合使用
 Header	建立多栏目表头
 FileListBox	建立磁盘文件列表框组件
 DirectoryListBox	建立驱动器目录列表框组件
 DriveComboBox	建立驱动器组合框组件
 FilterComboBox	建立文件过滤器组合框组件
 DBLookupList	以列表方式显示查询数据列表
 DBLookupCombo	以组合框方式显示查询数据列表

17. Samples 组件选项卡

Samples 组件选项卡如图 3.22 所示，其中各组件的功能如表 3.18 所示。



图 3.22 Samples 组件选项卡

表 3.18 Samples 选项卡中各组件的功能

组 件	功 能
 Pie	建立一个统计饼图
 TrayIcon	建立一个托盘图标
 PerformanceGraph	建立一个方格图
 CspinButton	建立一个类似于 UpDown 的按钮
 CspinEdit	相当于 SpinButton 和 Edit 的组合
 CcolorGrid	建立一个颜色栅格，可以选择前景、背景颜色
 Cgauge	建立一个模拟仪表
 CdirectoryOutline	建立一个驱动器的大纲目录结构
 Ccalendar	建立一个日历，可以向前或向后浏览
 IBEventAlerter	建立一个事件报警器

18. ActiveX 组件选项卡

ActiveX 组件选项卡如图 3.23 所示, 其中各组件的功能如表 3.19 所示。

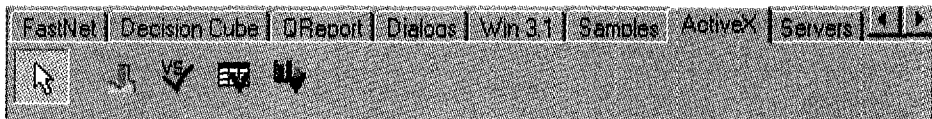


图 3.23 ActiveX 组件选项卡

表 3.19 ActiveX 选项卡中各组件的功能

组 件	功 能
ChartFX	建立功能强大可定制多种风格的图表
VSSpell	用于单词拼写检查
FIBook	建立一个简单扩展显示的表格
VtChart	建立三维图表

19. Servers 组件选项卡

如图 3.24 所示的 Servers 组件选项卡提供了对通常的 COM 服务器的封装, 这些组件都是从 ToleServers 继承下来, 操纵 Microsoft Office 程序和文档。它们都是通过引入类型库创建并已经安装好的服务器程序(通过这些组件可以与 COM 服务器建立连接, 如: 与 Word 文档、Excel 表格、Access 数据库、PowerPoint 多媒体、Outlook 邮件的连接等)。

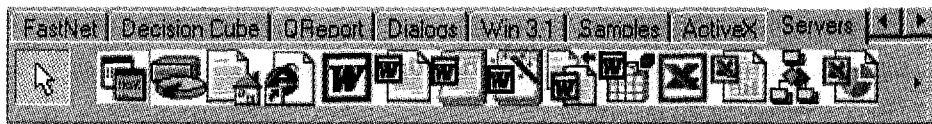


图 3.24 Servers 组件选项卡

3.5 对象观察器

图 3.25 展示了对对象观察器(Object Inspector)。一个窗体上有多少组件以及组件的属性和事件, 可以直接从对象观察器上观察, 也可以设置组件的属性和与该组件相关的消息事件处理响应。对象观察器包括以下 3 个部分: 下拉式列表框、属性选项卡、事件选项卡。对象观察器的上部是下拉式列表框, 通过单击下拉式列表的向下箭头, 就可以观察窗体上所有的组件, 单击组件选项, 就可以在属性选项卡和事件选项卡进行设置。

- 属性选项卡(Properties): 用来设置窗体和窗体中组件属性。例如, 将窗体 Form1 的 Caption 属性设置为“中国象棋”, 就会发现窗体的标题由 Form1 变为中国象棋。

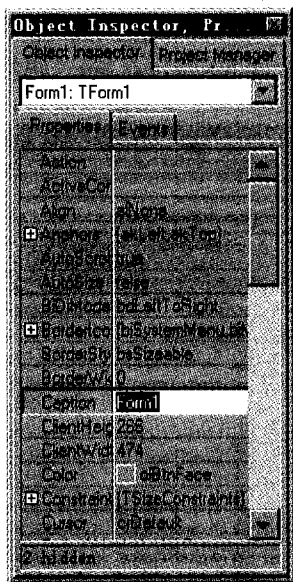


图 3.25 对象观察器

- 事件选项卡(Events): 用于建立事件句柄。举例来说: 首先在窗体上单击激活事件选项卡, 在 OnCloseQuery 事件的空白处双击, C++Builder 会在代码编辑器中自动为用户生成事件处理过程函数框架, 如图 3.26 所示。

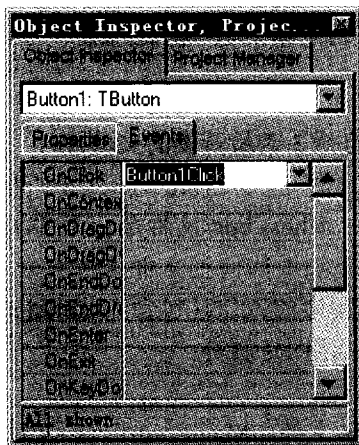


图 3.26 事件选项卡

3.6 项目管理器

在 C++Builder 5.0 中, 开始一个新的应用程序就称为开始一个新的项目, 系统会自动生成许多文件, C++Builder 5.0 通过使用项目管理来管理项目中各类文件的增加、删除和打开。开始一个新的项目时, 项目管理器并不出现在集成环境中, 而是通过选择 View |

Project Manager 命令来激活。以下举例说明观察项目管理器。选择 File | Open Project 选中命令，在打开的对话框中选中 c:\BcbChess\chess.bpr，再激活项目管理器，如图 3.27 所示。

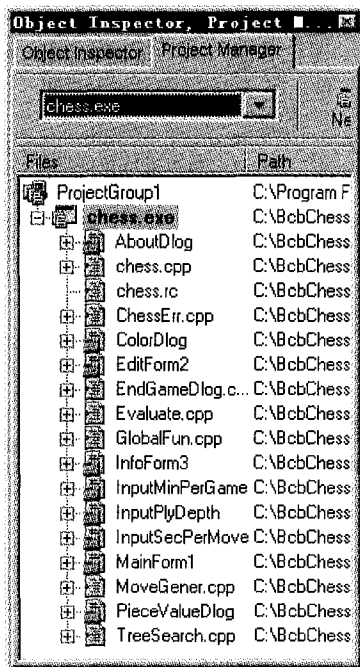


图 3.27 项目管理器

C++Builder 应用程序通常包括 3 类最基本的文件：

- 项目文件

该文件包含 3 个文件：Chess.bpr 文件是一个文本文件，记录了各种资源信息，通过项目文件，系统就知道如何把文件连接起来，最终生成用户可执行文件；Chess.cpp 文件是应用程序的入口，包含 WinMain()函数；Chess.res 是资源文件，包含图标资源、光标、位图资源等。

- 窗体类文件

该文件通常也包含 3 种文件：MainForm1.cpp 文件包含窗体及窗体消息处理的代码；MainForm1.dfm 是系统为描述窗体而创建的文件，一般用户不必对它进行编辑(文本状态时可以编辑)；MainForm1.dfm 文件是二进制文件，它主要用于提供如何绘制窗体的信息，例如窗体的大小、位置、边框，窗体中组件的位置、形态、字体、颜色等等；MainForm1.h 文件是窗体的头文件，是对窗体的定义说明。非可视化窗体类文件没有.dfm 文件。

- 单元类文件

该文件不依附于具体的窗体，但它可以被窗体过程调用，包括 ChessErr.cpp 和 ChessErr.h 两种文件。通过 New 工具，用户可以添加新的文件，也可以通过 Remove 按钮删除选定的文件。

3.7 代码编辑器

代码编辑器帮助程序员查看和编写应用程序的源代码。代码编辑器是一个功能完善的 ASCII 文件编辑器，程序员在代码编辑器中可以编辑任何的 C++Builder 源文件和 ASCII 文件。代码编辑器如图 3.28 所示，由代码正文窗格、装订区和信息窗格组成。

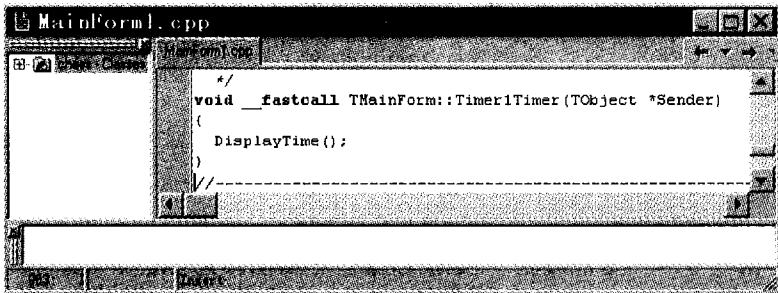


图 3.28 代码编辑器

当在文本窗格区中单击右键，会弹出快捷菜单，快捷菜单包括了许多常用的编辑选项、调试选项以及其他选项。

3.8 图像编辑器

Windows 程序的图形资源通常包括图标、位图和光标。图标(Icons)是一种小位图，通常只有 32×32 或 16×16 像素大小，使用有限的颜色集合，用于图形化地表示窗体与应用程序。光标(Cursor)也是一种小位图，只有 32×32 像素大小，一般使用 4 种颜色(黑、白、透明和反白)，可以用于显示屏幕上鼠标的位置。在 C++Builder 中支持 18 个预定义光标集合，而且还可以添加自己定义的光标。作者这次制作了一个 256 色的光标(具体做法见第 5 章“创建和使用自己的光标”一节)。

C++Builder 中图像编辑器是一个能够生成和编辑位图(.bmp)、图标(.ico)和光标(.cur)的程序。它还可以生成一个资源文件(.res)，该资源文件中可以包含多个位图、图标和光标。可以在 C++Builder 的 Tools 菜单中选择相应的命令 Image Editor，打开图像编辑器。用图像编辑器生成位图、图标和光标非常简单。具体做法是：打开图像编辑器，选择 File | New 命令，然后从弹出菜单中选择资源文件，这时就显示出一个项目窗口。用鼠标右键点击 Contents，弹出一个快捷菜单，可以选择建立位图、光标、图标。显示窗口如图 3.29 所示。绘制所需要的图形以后，保存资源内容，图像资源编辑器就会将其编译成一个二进制资源文件。随后在应用程序中就可以使用这些资源文件了。

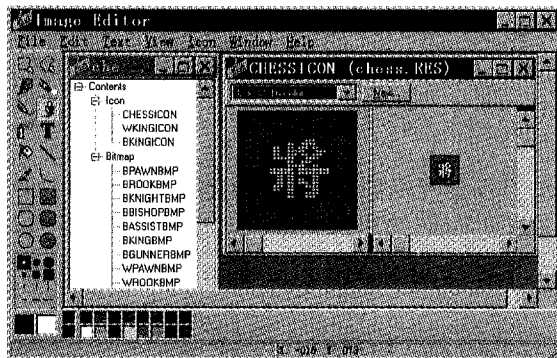


图 3.29 图像编辑器

除使用图像编辑器生成资源文件外，另外有一个较好的方法是间接编译生成资源文件，当然得先做好单个的图形文件，也可从网上下载。利用 C++Builder 提供的命令行应用程序 Brcc32.exe 或 Brc32.exe，该程序位于 C++Builder 的 Bin 目录下。利用 Brcc.exe 或 Brc32.exe 建立资源文件，首先需在 C++Builder 主菜单中选择 File | New 命令打开 Object Repository，然后在 New 选项卡中选择 Text 图标。这时会在 C++Builder 的代码编辑器中产生一个空白文件(当然，也可用别的文本编辑器)，在空白文件中编写代码如下：

```
ChessIcon  ICON  Chess.ico
RkingIcon  ICON  Rking.ico
BkingIcon  ICON  Bking.ico

BPawnBmp  BITMAP  Bpawn.bmp
BRookBmp  BITMAP  Brook.bmp
BKnightBmp  BITMAP  Bkngt.bmp
BBishopBmp  BITMAP  Bbishop.bmp
BAssistBmp  BITMAP  Bassist.bmp
BKingBmp  BITMAP  Bking.bmp
BGunnerBmp  BITMAP  Bgunner.bmp

RPawnBmp  BITMAP  Rpawn.bmp
RRookBmp  BITMAP  Rrook.bmp
RKnightBmp  BITMAP  Rkngt.bmp
RBishopBmp  BITMAP  Rbishop.bmp
RAssistBmp  BITMAP  Rassist.bmp
RKingBmp  BITMAP  Rking.bmp
RGunnerBmp  BITMAP  Rgunner.bmp

PMaskBmp  BITMAP  Pmask.bmp
RMaskBmp  BITMAP  Rmask.bmp
```

```
KMaskBmp BITMAP Ktmask.bmp  
BMaskBmp BITMAP Bmask.bmp  
AMaskBmp BITMAP Amask.bmp  
KMaskBmp BITMAP Kmask.bmp  
GMaskBmp BITMAP Gmask.bmp
```

```
BKBrushBmp BITMAP BkBrush.bmp
```

将上面建立的文件以 RC 扩展名保存。

建立 RC 文件以后,进入 MS-DOS 方式,在 C++Builder 的 Bin 目录下找到 Brcc32.exe 或 Brcc32.exe 命令行应用程序,然后编译资源,代码如下:

```
Brcc32 c:\BcbChess\chess.rc
```

或

```
Brc32 c:\BcbChess\chess.rc
```

这时在与 chess.rc 相同的目录下会自动产生资源文件 chess.res。

在建立图形资源文件以后,有两种方式可以将资源文件包含到应用程序中,第一种方法可以通过在 C++Builder 主菜单中选择 Project | Add to Project 命令将资源文件加入到应用程序中,这样加入的资源可以在应用程序的任何单元中都可以使用。这时如果打开项目文件,会发现项目文件的头文件中#pragma hdrstop 下增加了如下一行代码:

```
USERES("chess.res");
```

除了通过菜单命令加入资源文件以外,也可以在各单元文件中通过手工加入资源。例如要将资源文件 chess.res 和 MyRc.res 手工加入到单元文件中,加入代码如下:

```
#pragma package(smart_init)  
#pragma resource "*.dfm"  
  
#pragma resource "Chess.res"  
#pragma resource "MyRc.res"
```

3.9 桌面工具栏

如图 3.30 所示的桌面工具栏是 C++Builder 5.0 的一项功能。



图 3.30 桌面工具栏

桌面工具栏的功能主要：选择保存过的桌面设置、保存当前的桌面的设置、将当前的桌面设置为调试桌面。

用户可以根据需要定义和保存多种桌面布局，选择“Save Desktop”输入一个文件名即可。每一个桌面文件都被加入到“Desktops”工具栏的下拉列表中。

要选择一种桌面布局，可以在“Desktops”工具栏的下拉列表中选择一列表文件，也可以选择 View | Desktops 命令，再选择一个桌面文件。

要删除一种桌面布局，可以使用“Desktops”命令中的“Delete”。

3.10 使用联机帮助

不论是用开发工具开发应用程序还是学习这门开发工具，使用帮助系统都是必不可少的。任何一本介绍开发工具的书都不如帮助系统考虑得周全也不及帮助系统解释得准确。联机帮助是最权威最全面的参考书。通过单击“About”菜单，可以连至 Borland 公司的网页上，访问 C++Builder 的最新发布信息、升级信息或网上编程经验交流等。

使用联机帮助有 3 种方法：

- 在帮助目录中浏览帮助主题
- 通过索引查找帮助主题
- 显示与上下文有关的帮助信息

下面就这 3 种方法予以介绍。

1. 在帮助目录中浏览帮助主题

- (1) 选择 Help | C++Builder Help 命令，或直接按 F1 键，将出现如图 3.31 所示的 Borland C++Builder Help 对话框。
- (2) 选中目录标签。
- (3) 在【目录】选项卡中，选中主题，单击扩展主题，逐步查找，直到找到需查找的主题。

2. 通过索引查找帮助主题

- (1) 选择 Help | C++Builder Help 命令，或直接按 F1 键。
- (2) 选中索引标签。将出现如图 3.32 所示的 Borland C++Builder Help 对话框的【索引】选项卡。
- (3) 可在索引输入的文本框中键入需要查询的关键字(如组件名称、方法或函数名称等)，即可获得帮助文档。例如我们需查找 VCL 库提供的画布类，键入 Canvas，如果已经知道画布类 Canvas，还需要知道画布类提供的处理方法，这时候将鼠标指向需要了解的对象名称，就可获取该对象的帮助信息。

3. 显示与上下文有关的帮助信息

更快捷的方式是不必搜寻 Help 菜单，只要将鼠标指向需要了解的对象名称，再按下

<F1>键,可直接获取该对象许多有价值的联机帮助信息,不啻于一本方便的参考书。与上下文相关的部分是:组件栏中的组件、C++Builder 中的每个窗口(如代码编辑窗口,项目管理窗口等)、窗口中的对象、C++Builder 关键字(语句、函数、属性、方法、事件等)、调试过程中的错误信息等。

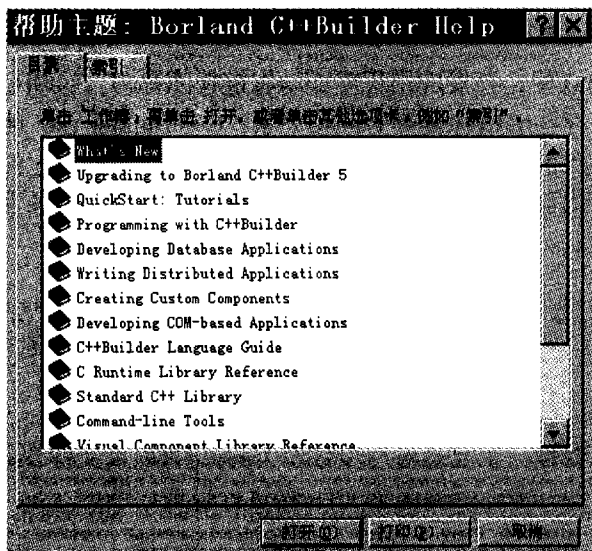


图 3.31 帮助主题

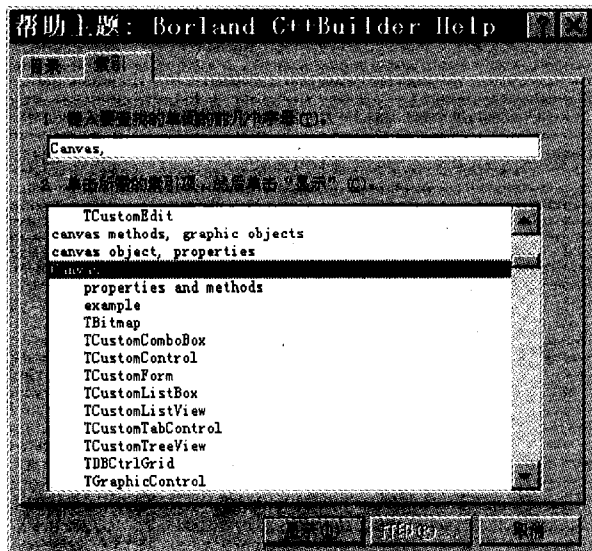


图 3.32 帮助索引

第 4 章 运行 CHESS 项目

在 C++Builder 5.0 中, 开发应用程序或动态链接库称为一个项目。在一个大型的项目中, 文件与文件之间存在着错综复杂的依赖关系, C++Builder 5.0 能够自动管理各种类型的文件, 保证目标文件的建立总是基于最新的代码, 能够控制程序的运行、监视程序的输出。在这一章中, 我们通过先设立一个简单的基础, 然后在此基础上进一步扩展其功能来逐步完善 CHESS 项目。

4.1 管理 CHESS 项目

对一个项目进行管理, 主要是对与项目相关的文件(项目文件.bpr 或项目组文件.bpg、窗体文件.cpp 与.h 文件、窗体回调函数文件等)进行管理。

对一个项目经过彻底的了解、分析, 作好顶层规划后, 接下来就要着手编辑代码。C++Builder 可视化编程思路是: 设计项目的主界面, 应该安放哪些组件, 设计菜单, 对事件进行处理。

4.1.1 开始 CHESS 项目

建立一个新的项目有多种途径, 用户可以自由选择。

- (1) 选择 File | New Application 命令就可以开始一个新的项目, 也可以选择 File | New 命令, 弹出 New Items 对话框, 如图 4.1 所示。

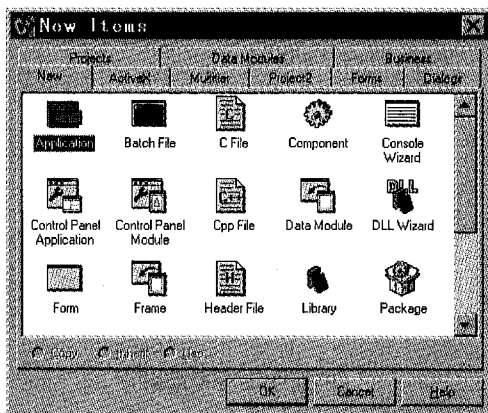


图 4.1 开始一个新的项目

New Intms 对话框中的各个标签及选项卡中的选项显示了 C++Builder 可以创建各种类型的程序单元。New 选项卡中的 Application 的含义是创建一个新的应用项目，包括一个窗体(.dfm)、窗体类文件(.cpp 和.h)和一个项目文件(.bpr)。

(2) 选中 Application，单击 OK 按钮开始一个新的项目，界面如图 4.2 所示。

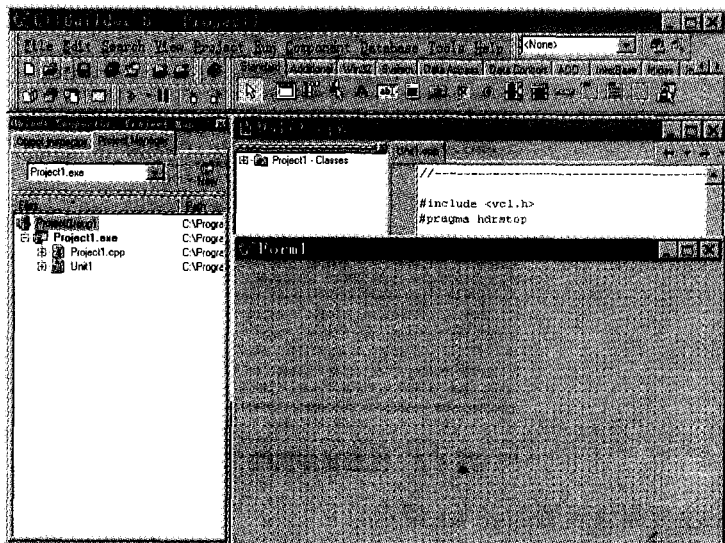


图 4.2 新建项目的开始界面

(3) 选取“Save Project As...”保存项目文件，程序将自动依次弹出“Save Unit1 As”对话框，键入文件名“MainForm1.cpp”保存单元文件，在“Save Project1 As”对话框，键入文件名“Chess.bpr”保存项目文件。

这样，程序将自动生成 Chess.cpp、Chess.res、Chess.bpr、MainForm1.cpp、MainForm1.h、MainForm1.dfm 等 6 个文件。下次只要双击 Chess.bpr 图标就可打开 CHESS 项目。要说明的是，这只是建立了 CHESS 项目的基础部分，但在此基础上，通过后面将要述及的扩展 CHESS 项目功能，就可以建立起宏伟壮丽的 CHESS 项目大厦了。

4.1.2 设计对弈主窗体

MainForm 是整个中国象棋程序对弈的主窗体，也是另两个 EditForm 和 InfoForm 窗体的父窗体。

1. 主窗体的主要属性

当我们新建 CHESS 项目时，C++Builder 就自动弹出一个窗体，它的许多默认属性可以直接使用。如果觉得不合适，可以用鼠标拖动窗体边框改变窗体大小直到满意为止。也可以在对象管理器中通过键盘输入属性的值完成。具体操作如下：

(1) 选择 View | Object Inspector 命令弹出 Object Inspector 对话框的 Properties 选项卡。

- (2) 在相应的属性中选取您想要的属性值，如图 4.3 所示。例如设 biMaximize 为 false 将关闭窗口最大化功能。

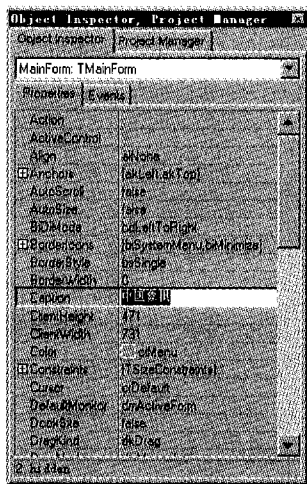


图 4.3 设置窗体属性

MainForm 的主要属性设置如表 4.1 所示。

表 4.1 主窗体的主要属性

属 性	属 性 值
Left	206
Top	233
HorzScrollBar.Style	SsFlat
BorderIcons	[biSystemMenu, biMinimize]
BorderStyle	BsSingle
Caption	中国象棋
ClientHeight	458
ClientWidth	726
Color	ClMenu
Ctl3D	False
Font.Charset	DEFAULT_CHARSET
Font.Color	ClWindowText
Font.Height	-11
Font.Name	MS Sans Serif
Menu	TchessMenu
OldCreateOrder	False
Position	PoScreenCenter
PixelsPerInch	96
TextHeight	13

当设计好主窗体的主要属性后,接下来设计菜单(具体设计过程见 4.1.3 节“交互菜单设计”)。方法是:单击“Standard 组件选项卡”中的“MainMenu”组件后,在主窗体中再单击鼠标,这样就出现了菜单组件,双击该组件,就可以编辑菜单了。MainForm 中包括 3 个主菜单 TChessMenu、TChessThinkMenu、TChessEditMenu,如表 4.2 所示。

表 4.2 主窗体的 3 个主菜单

主菜单	菜 单	属性 AutoHotkeys
TChessMenu	游戏	MaManual
	编辑	
	级别	
	选项	
	显示	
TChessEditMenu	完成	MaManual
	清除	
	轮流	
	忽略	
TChessThinkMenu	停止	MaManual

最后在 MainForm 中添加几个可视化组件:公用对话框组件 OpenFileDialog1、公用存储对话框组件 SaveDialog1、定时器 Timer1。设计好的主窗体界面如图 4.4 所示。不过和运行时的主窗体(如图 4.5 所示)比较有很多不一样。

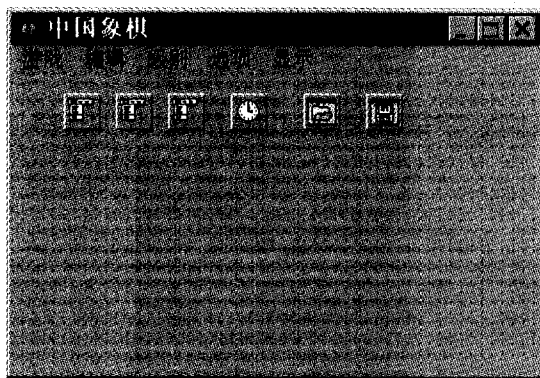


图 4.4 MainForm 主窗体的设计界面

2. 窗体的主要事件

CHESS 主窗体只用到标准组件的内置事件(如图 4.6 所示),没有定义新的事件。以下事件是窗体的默认事件。

- OnCloseQuery 单击系统关闭 X 按钮或退出菜单时请求窗口关闭事件。
- OnCreate MainForm 创建时的事件。
- OnDestroy MainForm 撤消时的事件。

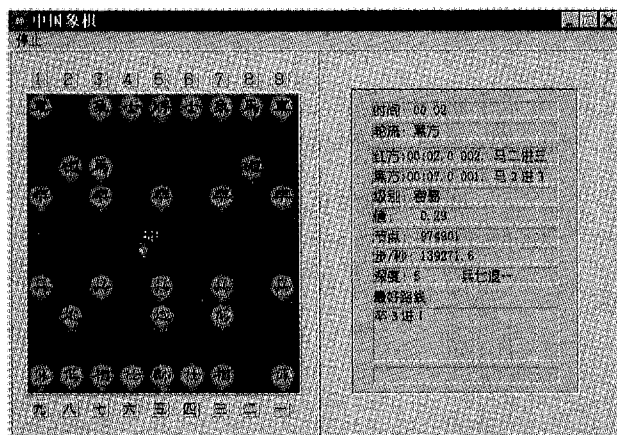


图 4.5 MainForm 主窗体运行时的界面。

- OnMouseDown 在 MainForm 中单击鼠标时的事件。
- OnMouseMove 在 MainForm 中移动鼠标时的事件。
- OnPaint MainForm 窗体重画事件。
- OnShow MainForm 窗体显示事件。

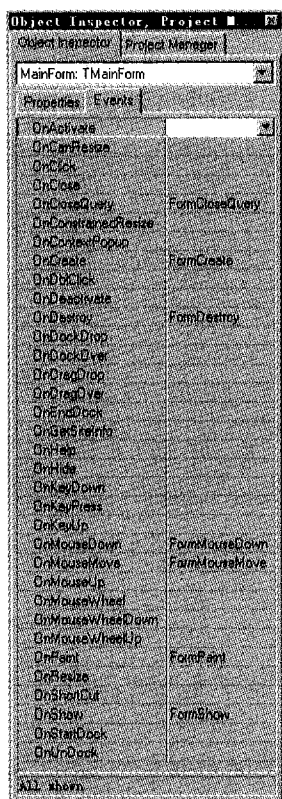


图 4.6 主窗体的内置事件

3. 窗体的方法

窗体事件回调一开始是空的,即什么也不干。CHESS 事件除了系统事件处理程序外,还有大量的菜单事件处理程序。CHESS 事件处理程序是通过调用定义的方法,执行相应的动作的。在某种程度上,方法也称为函数。定义窗体的方法过程如下:

- (1) 切换到与主窗体 MainForm 相对应的对象观察器的 Events 选项卡。
- (2) 在事件表格的相应事件的右边一栏双击鼠标就会自动弹出代码编辑窗口,这样用户就可以编写响应代码了,如图 4.7 所示。
- (3) 对于菜单事件,需切换到与菜单相对应的对象观察器的 Events 选项卡,同样双击鼠标就会自动弹出代码编辑窗口,如图 4.8 所示。

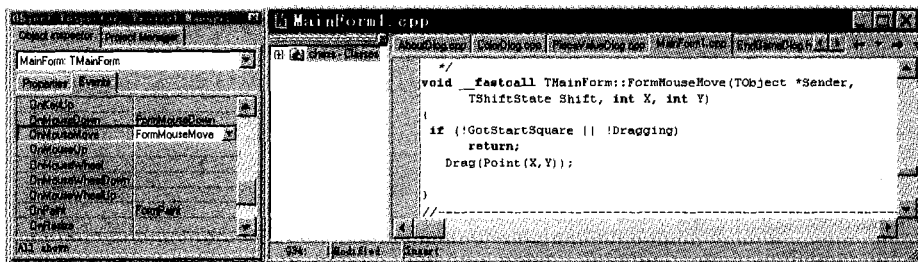


图 4.7 编写主窗体 OnMouseMove 事件回调函数 FormMouseMove 的代码

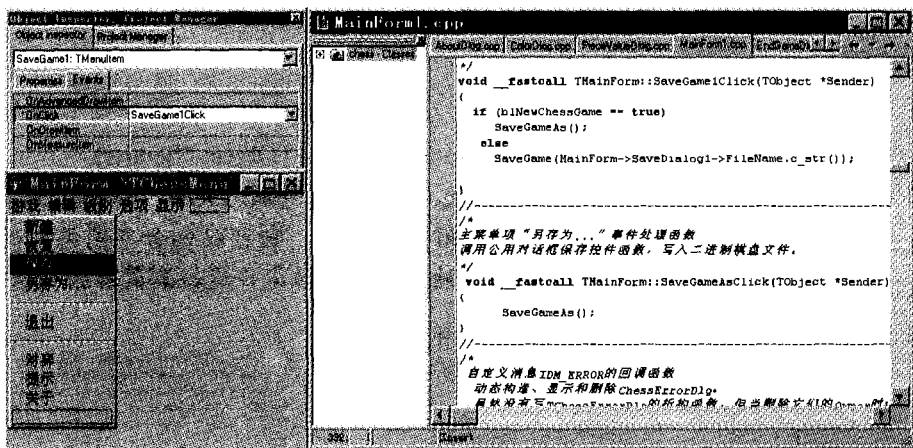


图 4.8 编写“保存”菜单事件 OnClick 的回调函数 SaveGame1Click 代码

事件响应代码总是由应用程序开发人员编写的。

下列函数事实上出现在主窗体 MainForm 的单元文件 MainForm1.cpp 中。注意到 MainForm1.cpp、MainForm1.h、MainForm1.dfm 是三位一体的。

- (1) 函数名称: void __fastcall TMainForm::FormCreate(TObject *Sender)

说明: 主窗体必须负责创建子窗体,前面两行分别创建了两个子窗体,接下来的两行将这些子窗体放到一个数组中。然后,通过一个循环依次显示这些子窗体。使用这项技术时,不要让系统自动创建这些子窗体。为此,需要选择 Project/

Options 命令打开 Project Options 对话框, 切换到 Forms 选项卡, 然后将这些子窗体从左边列表框移到 “Available forms” 列表框中。

函数定义如下:

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
    InfoForm = new TInfoForm(this);
    EditForm = new TEditForm(this);
    ChildForms[1] = EditForm;
    ChildForms[2] = InfoForm;
    for (int i=1;i<3;i++)
        ChildForms[i]->Show();

    SoundOn = true;
    FileName = new char[MAXPATH];
    WhoseTurn = player;
    CurPlayer = red;
    ComputerColor = black;
    Editing = EditingChessBoard = GotStartSquare = false;
    blNewChessGame = true;
}

CheckLevelMenu(LEVELTYPE);
//-----
```

(2) 函数名称: void __fastcall TMainForm::FormShow(TObject *Sender)

说明: 这是主窗体显示时事件处理函数。从资源文件载入光标及位图。光标及位图保存在资源文件(.res)内有两个好处: 第一, 它可以包含不同类型的多种资源; 第二, C++Builder 可以方便地把一个资源文件连接到最后的可执行文件。

注意: "::<"在变量前表明该变量是全局(外部)变量,
"::"在函数前表明该函数是全局(外部)函数。

函数定义如下:

```
void __fastcall TMainForm::FormShow(TObject *Sender)
{
    HDC hDC;
    hDC = GetDC(MainForm->Handle);
    TEXTMETRIC tm;
    GetTextMetrics(hDC, &tm);
    CHARSIZE = short(tm.tmAveCharWidth);
    LINESIZE = short(tm.tmHeight + tm.tmExternalLeading);
```

```

        ReleasedDC(MainForm->Handle, hDC);
        BORDERSIZE = LINESIZE + MYFRAMESIZE;
        If((GetSystemMetrics(SM_CXSCREEN)<800)|| (GetSystemMetrics(SM_
CYSCREEN)<600)) MessageBox(NULL, "屏幕分辨率低于 800×600!", "警告",
MB_ICONINFORMATION | MB_OK);
        Screen->Cursors[crWaitCursor]=LoadCursor(HInstance,
"IDC_WCUR");
        //把自定义光标装入 Screen 对象
        ::PieceBmpArray[pawn-1][red]=LoadBitmap(HInstance, "RPawnBmp");
        ::PieceBmpArray[pawn-1][black] = LoadBitmap(HInstance, "Bpawn
Bmp");
        ::PieceBmpArray[rook-1][red] = LoadBitmap(HInstance, "RRookBmp");
        ::PieceBmpArray[rook-1][black] = LoadBitmap(HInstance, "Brook
Bmp");
        ::PieceBmpArray[knight-1][black] = LoadBitmap(HInstance, "BKnightBmp");
        ::PieceBmpArray[bishop-1][black] = LoadBitmap(HInstance, "BBishopBmp");
        ::PieceBmpArray[assist-1][black] = LoadBitmap(HInstance, "BAssistBmp");
        ::PieceBmpArray[gunner-1][black] = LoadBitmap(HInstance, "BGunnerBmp");

        ::PieceBmpArray[knight-1][red] = LoadBitmap(HInstance, "RKnightBmp");
        ::PieceBmpArray[bishop-1][red] = LoadBitmap(HInstance, "RBishopBmp");
        ::PieceBmpArray[assist-1][red] = LoadBitmap(HInstance, "RAssistBmp");
        ::PieceBmpArray[gunner-1][red] = LoadBitmap(HInstance, "RGunnerBmp");
        ::PieceBmpArray[king-1][red] = LoadBitmap(HInstance, "RKingBmp");
        ::PieceBmpArray[king-1][black] = LoadBitmap(HInstance, "BKingBmp");
        ::MaskArray[pawn-1] = LoadBitmap(HInstance, "PMaskBmp");
        ::MaskArray[rook-1] = LoadBitmap(HInstance, "RMaskBmp");
        ::MaskArray[knight-1] = LoadBitmap(HInstance, "KMaskBmp");
        ::MaskArray[bishop-1] = LoadBitmap(HInstance, "BMaskBmp");
        ::MaskArray[assist-1] = LoadBitmap(HInstance, "AMaskBmp");
        ::MaskArray[gunner-1] = LoadBitmap(HInstance, "GMaskBmp");
        ::MaskArray[king-1] = LoadBitmap(HInstance, "KMaskBmp");

        MainWndRect= MainForm->GetClientRect();
        MainForm->Menu=TCChessMenu;    //设置主窗体主菜单
        Talk();
        MainForm->CheckLevelMenu(Level);
        hColorBrush = CreateSolidBrush( RGB(ColorSquareColors[0], ColorSquare
Colors[1], ColorSquareColors[2]));

```

```

Dragging = true;

Screen->Cursor = TCursor(crArrow); //分配箭头光标给应用程序
ChildForms[2]->BringToFront();
}

```

//-----

(3) 函数名称: void TMainForm::SetMainWindowBk()

说明: 设置主窗体背景。通过文件方式载入位图。由于位图没有保存在资源文件(.res)内, C++Builder 不能把从文件装入的位图连接到最后的可执行文件。因此, 必须确保在 c:\BcbChess 下有必需的位图文件。

很明显, 使用 VCL 函数比原始的 API 函数要方便得多。使用后, 应将 Canvas->Brush->Bitmap 这个特性设为 NULL, 以释放位图。如果不删除位图, 将导致程序内存不足。

函数定义如下:

```

void TMainForm::SetMainWindowBk()
{
    Graphics::TBitmap *BLKBrushBmp = new Graphics::TBitmap();
    BLKBrushBmp->LoadFromFile("c:\\BcbChess\\BKBrush.bmp");
    MainForm->Canvas->Brush->Bitmap=BLKBrushBmp;
    MainForm->Canvas->FillRect(MainForm->GetClientRect());
    MainForm->Canvas->Brush->Bitmap=NULL;
    delete BLKBrushBmp;
}

```

//-----

(4) 函数名称: void __fastcall TMainForm::FormPaint(TObject *Sender)

说明: 主窗体重画事件处理函数。ReleaseDC 函数释放设备描述句柄, 不然, 会导致系统资源不足。

函数定义如下:

```

void __fastcall TMainForm::FormPaint(TObject *Sender)
{
    SetMainWindowBk();
    HDC PaintDC = GetDC(MainForm->Handle);
    DrawFrame(PaintDC, MainWndRect);
    ReleaseDC(MainForm->Handle, PaintDC);
    PaintDC = GetDC(InfoForm->Handle);
    DrawFrame(PaintDC, InfoAreaRect);
    ReleaseDC(InfoForm->Handle, PaintDC);
    PrintChessBoard();
}

```



```

    }
//-----

```

(5) 函数名称: void __fastcall TMainForm::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift, int X, int Y)

说明: 单击鼠标时事件处理函数。对弈状态时, 单击鼠标左键, 可以抓住棋子进行移动, 第二次单击时, 释放棋子。编辑状态时, 单击鼠标左键, 可以使编辑窗体中的棋子透明显示, 第二次在主窗体单击时, 可以添加棋子, 再单击时, 擦除棋子。

函数定义如下:

```

void __fastcall TMainForm::FormMouseDown(TObject *Sender,
    TMouseButton Button, TShiftState Shift, int X, int Y)
{
    POINT point;
    if (Button != mbLeft)
        return;
    MouseX=X;
    MouseY=Y;
    if (EditingChessBoard)
    {
        DoEdit();
        return;
    }

    if (WhoseTurn == computer && NoComputerMove == false)
        return;
    point=Point(MouseX,MouseY);
    MainForm->ScreenToClient( point);

    if (!GotStartSquare)
    {
        MoveStartSquare = GetValidSquare(point, CurPlayer, true);
        if (MoveStartSquare == -1)
            return;
        GotStartSquare = true;
        if (!Dragging)
            DrawInvertedBitmap(MoveStartSquare);
        else
            DragStart(MoveStartSquare, point);
    }
}

```

```

        }
    else
    {
        MoveEndSquare = GetValidSquare(point, CurPlayer, false);
        GotStartSquare = false;
        if (MoveEndSquare == -1)
        {
            if (Dragging)
                DragEnd(false);
            Warning("错误的移动!");
            DrawNormalBitmap(MoveStartSquare);
            return;
        }
        if (!MoveCheck(MoveStartSquare, MoveEndSquare))
        {
            if (Dragging)
                DragEnd(false);
            DrawNormalBitmap(MoveStartSquare);
            return;
        }
        GotValidMove = true;
        UndoMove->Enabled = true;
        if (GameOver)
            EndGame();
        else if (NoComputerMove == false)
            ComputersTurn();
        else if (MultiMove)
        {
            CurPlayer = (CurPlayer == red) ? black : red;
        }
    }
}

//-----

```

(6) 函数名称: void TMainForm::ComputersTurn()

说明: 轮到计算机计算时的回调函数。程序处于机器控制状态。主菜单发生改变。光标发生改变。

函数定义如下:

```
void TMainForm::ComputersTurn()
```

```

{
    WhoseTurn = computer;

    Screen->Cursor = TCursor(crWaitCursor); //分配等待光标给应用程序
    MainForm->Menu=TChessThinkMenu; //设置主窗体主菜单
    ProgramMove();

    MainForm->Menu=TChessMenu; //动态设置主窗体主菜单
    WhoseTurn = player;

    Screen->Cursor = TCursor(crArrow); //分配箭头光标给应用程序
    if (GameOver)
        EndGame();
}

//-----

```

(7) 函数名称: void __fastcall TMainForm::NewChessGame1Click(TObject *Sender)

说明: 主菜单项“新建”事件处理函数。主窗体进入初始对弈状态。同时清除信息窗体中的提示信息。

函数定义如下:

```

void __fastcall TMainForm::NewChessGame1Click(TObject *Sender)
{
    Timer1->Enabled=false;
    InfoForm->Timer->Caption="时间: ";
    UndoMove->Enabled = false;
    RedoUndo->Enabled = false;

    blNewChessGame = true;
    NewChessGame();
    PrintChessBoard();
    CurPlayer = Player;
    ComputerColor = Opponent;
}

//-----

```

(8) 函数名称: void TMainForm::FileNew(TMessage &Msg)

说明: 自定义 IDM_FILENEW 消息回调函数。单击动态对话框“YES”按钮发送一个消息, 相当于单击主菜单项“新建”。

函数定义如下:

```

void TMainForm::FileNew(TMessage &Msg)
{
    MainForm->NewChessGame1Click(NewChessGame1);
}

//-----

```

(9) 函数名称: void __fastcall TMainForm::RestoreGame1Click(TObject *Sender)

说明: 主菜单项“恢复”事件处理函数。调用公用对话框打开组件函数, 读入二进制棋盘文件。

函数定义如下:

```
void __fastcall TMainForm::RestoreGame1Click(TObject *Sender)
{
    if (OpenDialog1->Execute())
    {
        blNewChessGame = false;
        RestoreGame(MainForm->OpenDialog1->FileName.c_str());
        CurPlayer = Player;
        ComputerColor = Opponent;
        RedoUndo->Enabled = false;    //重下菜单变灰
        UndoMove->Enabled = false;    //悔棋菜单变灰
    }
}
//-----
```

(10) 函数名称: void TMainForm::SaveGameAs()

说明: 自定义函数。调用公用对话框保存组件函数, 写入二进制棋盘文件。

函数定义如下:

```
void TMainForm::SaveGameAs()
{
    if (SaveDialog1->Execute())
    {
        blNewChessGame=false;
        SaveGame(MainForm->SaveDialog1->FileName.c_str());
    }
}
//-----
```

(11) 函数名称: void __fastcall TMainForm::SaveGame1Click(TObject *Sender)

说明: 主菜单项“保存”事件处理函数。调用公用对话框保存组件函数, 写入二进制棋盘文件。

函数定义如下:

```
void __fastcall TMainForm::SaveGame1Click(TObject *Sender)
{
    if (blNewChessGame == true)
        SaveGameAs();
}
```

```

    else
        SaveGame(MainForm->SaveDialog1->FileName.c_str());

}
//-----

```

(12) 函数名称: void __fastcall TMainForm::SaveGameAsClick(TObject *Sender)

说明: 主菜单项“另存为...”事件处理函数。调用公用对话框保存组件函数, 写入二进制棋盘文件。

函数定义如下:

```

void __fastcall TMainForm::SaveGameAsClick(TObject *Sender)
{
    SaveGameAs();
}
//-----

```

(13) 函数名称: void TMainForm::EMError(TMessage &Msg)

说明: 自定义消息 IDM_ERROR 的回调函数。动态构造、显示和删除 ChessErrorDlg。

虽然没有写 TChessErrorDlg 的析构函数, 但当删除它们的 Owner 时 VCL 自动删除了组件。如果包含非可视化对象的指针, 则需要增加 ChessErrorDlg 的析构函数。

函数定义如下:

```

void TMainForm::EMError(TMessage &Msg)
{
    TChessErrorDlg *ChessErrorDlg = new TChessErrorDlg(Application, (AnsiString)
buffer);
    ChessErrorDlg->ShowModal();
    delete ChessErrorDlg;
}

```

(14) 函数名称: void __fastcall TMainForm::AboutClick(TObject *Sender)

说明: 主菜单项“关于”事件处理函数。调用自定义静态对话框函数, 无模态显示版权信息。

函数定义如下:

```

void __fastcall TMainForm::AboutClick(TObject *Sender)
{
    AboutDlg->Show();
}
//-----

```

(15) 函数名称: void TMainForm::DoEdit()

说明: 自定义编辑函数。读取光标位置, 把选择的棋子在主窗体上的合法位置上显示出来。

函数定义如下:

```
void TMainForm::DoEdit()
{
    POINT point;
    SQUARETYPE Square;
    int SelectedItem;
    Modified = true;
    point=Point(MouseX,MouseY);
    MainForm->ScreenToClient(point);
    Square = GetValidSquare(point, black, false);

    if (ChessBoard[Square].piece != empty)
    {
        ChessBoard[Square].piece = empty;
        UpdateChessBoard();
        return;
    }
    SelectedItem = EditForm->GetSelectedItem();

    ChessBoard[Square].piece = (PIECETYPE)(SelectedItem % 7 + 1);
    ChessBoard[Square].color = (SelectedItem < 7) ? red : black;
    UpdateChessBoard();
}

//-----
```

(16) 函数名称: void __fastcall TMainForm::MovePieceClick(TObject *Sender)

说明: 菜单项“摆子”的回调函数。EditForm->BringToFront()将编辑子窗体 EditForm 推到前端。CheckMenuItem 有 3 个参数, 其中第一个参数为菜单句柄, 从菜单 TChessEditMenu 句柄属性 Handle 获取; 第二个参数告诉菜单项是用命令 ID 指明的; 第三个参数它告诉菜单项是应该执行哪一些操作。在这种情况下, MF_CHECKED 告诉 CheckMenuItem 菜单项被打勾, MF_UNCHECKED 告诉 CheckMenuItem 菜单项勾被取消。

函数定义如下:

```
void __fastcall TMainForm::MovePieceClick(TObject *Sender)
{
```

```

    SaveChessBoard = new BOARDTYPE[0x99];
    if (SaveChessBoard == NULL)
    {
        Error("没有足够的内存");
        return;
    }

    memcpy(SaveChessBoard, ChessBoard, sizeof(BOARDTYPE) * 0x99);
    Editing = EditingChessBoard = true;
    MainForm->Menu=TCChessEditMenu; //动态设置主窗体主菜单
    Modified = false;
    if (CurPlayer == red)
    {
        CheckMenuItem(TCChessEditMenu->Handle, BlackTurn->Command, MF_UNCHECKED);
        CheckMenuItem(TCChessEditMenu->Handle, RedTurn->Command, MF_CHECKED);
    }
    else
    {
        CheckMenuItem(TCChessEditMenu->Handle, BlackTurn->Command, MF_CHECKED);
        CheckMenuItem(TCChessEditMenu->Handle, RedTurn->Command, MF_UNCHECKED);
    }

    EditForm->BringToFront();
}
//-----

```

(17) 函数名称: void __fastcall TMainForm::ClearClick(TObject *Sender)

说明: 主菜单项“清除”事件处理函数。把主窗体中棋盘上棋子全部擦除。

函数定义如下:

```

void __fastcall TMainForm::ClearClick(TObject *Sender)
{
    SQUARETYPE sq;

    Modified = true;
    for (sq = 0; sq <= 0x98; sq++)
        ChessBoard[sq].piece = empty;
    UpdateChessBoard();
}
//-----

```

(18) 函数名称: void __fastcall TMainForm::CancelClick(TObject *Sender)

说明： 主菜单项“忽略”事件处理函数。把主窗体中棋盘上编辑过的棋子全部作废。函数定义如下：

```
void __fastcall TMainForm::CancelClick(TObject *Sender)
{
    memcpy(ChessBoard, SaveChessBoard, sizeof(BOARDTYPE) * 0x99);
    ResetNewPos();
    NormalSetup();
}
//-----
```

(19) 函数名称： void TMainForm::NormalSetup()

说明： 自定义正常设置函数。InfoForm->BringToFront()将信息子窗体 InfoForm 推到前端。恢复棋子在主窗体上的原始位置。

函数定义如下：

```
void TMainForm::NormalSetup()
{
    InfoForm->BringToFront();
    MainForm->Menu=TChessMenu; //动态设置主窗体主菜单
    delete[] SaveChessBoard;
    Editing = EditingChessBoard = false;
    UpdateChessBoard();
}

//-----
```

(20) 函数名称： void __fastcall TMainForm::DemoClick(TObject *Sender)

说明： 主菜单项“演示模式”事件处理函数。程序轮流成为红方、黑方，即自己与自己对弈。

函数定义如下：

```
void __fastcall TMainForm::DemoClick(TObject *Sender)
{
    extern bool AutoPlay;
    bool IsEasy = false;
    if (Level == easygame)
    {
        IsEasy = true;
        Level = normal;
        HideAttacks();
    }
}
```



```

    AutoPlay = true;
    ComputersTurn();
    if (IsEasy)
    {
        Level = easygame;
        UpdateChessBoard();
    }
    CurPlayer = Player;
    ComputerColor = Opponent;
    PrintCurLevel();
}
//-----

```

(21) 函数名称: void __fastcall TMainForm::InfiniteClick(TObject *Sender)

说明: 主菜单项“无限制”事件处理函数。改变信息窗体中信息提示为无限制程序
无限制地加大深度搜索最好的一步棋。

函数定义如下:

```

void __fastcall TMainForm::InfiniteClick(TObject *Sender)
{
    if (Level != infinite)
    {
        UnCheckLevelMenu(Level);
        CheckMenuItem(TChessMenu->Handle, Infinite->Command, MF_CHECKED);
    }
    Level = infinite;
    MaxLevel = MAXPLY;
    PrintCurLevel();
}
//-----

```

(22) 函数名称: void __fastcall TMainForm::MateClick(TObject *Sender)

说明: 主菜单项“将死搜索”事件处理函数。改变信息窗体中信息提示为将死搜索,
程序无限制地加大深度直到搜索到能将死的一步棋。当棋子比较多时, 很难
搜索到终局(将死)。

函数定义如下:

```

void __fastcall TMainForm::MateClick(TObject *Sender)
{
    if (Level != matesearch)
    {
        UnCheckLevelMenu(Level);
    }
}

```

```

    CheckMenuItem(TChessMenu->Handle, Mate->Command, MF_CHECKED);
}

Level = matesearch;
PrintCurLevel();
ComputersTurn();

}

//-----

```

(23) 函数名称: void __fastcall TMainForm::Ply1Click(TObject *Sender)

说明: 主菜单项“回合搜索”事件处理函数。弹出“设置回合数”对话框。在主菜单项“回合搜索”前打勾。改变信息窗体中信息提示为所设置的回合数。程序加大深度直到搜索到指定深度的一步棋。

函数定义如下:

```

void __fastcall TMainForm::Ply1Click(TObject *Sender)
{
    char *PlySearchDepth = new char[40];
    sprintf(PlySearchDepth, "%d", MaxLevel);
    if (InputPlyDlg->ShowModal() == mrOk)
    {
        int NewPlyDepth = StrToInt(InputPlyDlg->Edit1->Text);
        if (NewPlyDepth > 0)
        {
            if (Level != plysearch)
            {
                UnCheckLevelMenu(Level);
                CheckMenuItem(TChessMenu->Handle, Ply1->Command, MF_CHECKED);
            }
            MaxLevel = (BYTE)((NewPlyDepth > MAXPLY) ? MAXPLY : NewPlyDepth);
            Level = plysearch;
            PrintCurLevel();
        }
        else
            Error("数据不正确。请重输。");
    }
    delete PlySearchDepth;
}

//-----

```

(24) 函数名称: void __fastcall TMainForm::TwoPlayerClick(TObject *Sender)

说明：主菜单项“两人对弈”事件处理函数。翻转主菜单项“两人对弈”为“单独对弈”。

函数定义如下：

```
void __fastcall TMainForm::TwoPlayerClick(TObject *Sender)
{
    static LEVELTYPE OldLevel = normal;
    MultiMove = !MultiMove;
    if (MultiMove)
    {
        TwoPlayer->Caption= "单独对弈";    //动态改变菜单项文本
        OldLevel = Level;
        EnableMenuItem(TChessMenu->Handle, 2, MF_GRAYED | MF_BYPOSITION);
        DrawMenuBar(MainForm->Handle);
        Level = normal;
        PrintCurLevel();
    }
    else
    {
        TwoPlayer->Caption = "两人对弈";    //动态改变菜单项文本
        EnableMenuItem(TChessMenu->Handle, 2, MF_ENABLED | MF_BYPOSITION);
        /*
        第一个参数为菜单句柄，从菜单 TChessMenu 句柄属性 Handle 获取。
        第二个参数指明菜单的位置，即第三项。
        第三个参数是一个标志集合。它告诉菜单项是应该执行哪一些操作和如何被指明的。在这种情况下，
        MF_ENABLED 告诉 EnableMenuItem 菜单项激活。MF_BYPOSITION 告诉 EnableMenuItem 菜单项是用位置指明的。
        */
        DrawMenuBar(MainForm->Handle);
        Level = OldLevel;
        PrintCurLevel();
    }
}

//-----
```

(25) 函数名称：void __fastcall TMainForm::ReverseClick(TObject *Sender)

说明：主菜单项“反转棋盘”事件处理函数。对换红方和黑方的棋盘。

函数定义如下：

```
void __fastcall TMainForm::ReverseClick(TObject *Sender)
```

```
{
    Turned = !Turned;
    PrintChessBoard();
}
```

```
//-----
```

(26) 函数名称: void __fastcall TMainForm::UndoMoveClick(TObject *Sender)

说明: 主菜单项“悔棋”事件处理函数。至少要动过一次棋子, 该菜单项才由灰色变为黑色(激活状态)。

函数定义如下:

```
void __fastcall TMainForm::UndoMoveClick(TObject *Sender)
```

```
{
    if (ComputerThinking)
    {
        MessageToPost = IDM_UNDO;
        return;
    }
    if (!Undo())
        UndoMove->Enabled = false;
        RedoUndo->Enabled = true;

    CurPlayer = Player;
    ComputerColor = Opponent;
}
```

```
//-----
```

(27) 函数名称: void __fastcall TMainForm::RedoUndoClick(TObject *Sender)

说明: 主菜单项“重下”事件处理函数。至少要悔过一次棋子, 该菜单项才由灰色变为黑色(激活状态)。

函数定义如下:

```
void __fastcall TMainForm::RedoUndoClick(TObject *Sender)
```

```
{
    if (ComputerThinking)
    {
        MessageToPost = IDM_REDO;
        return;
    }
}
```

```

if (!Redo())
    RedoUndo->Enabled = false;
    UndoMove->Enabled = true;

```

```

CurPlayer = Player;
ComputerColor = Opponent;
}

```

//-----

(28) 函数名称: void TMainForm::CheckLevelMenu(LEVELTYPE level)

说明: 自定义子菜单项加勾函数。选中子菜单项“容易”、“每次移动时间”、“总时间”、“匹配”、“无限制”、“回合搜索”、“将死搜索”之一时, 在该项前打勾。

函数定义如下:

```

void TMainForm::CheckLevelMenu(LEVELTYPE level)
{
    int CheckItem = Easy->Command;
    switch (level)
    {
        case normal :
            CheckItem = MoveTime->Command;
            break;
        case fullgametime :
            CheckItem = TotalTime->Command;
            break;
        case plysearch :
            CheckItem = Ply1->Command;
            break;
        case easygame :
            CheckItem = Easy->Command;
            break;
        case infinite :
            CheckItem = Infinite->Command;
            break;
        case matesearch :
            CheckItem = Mate->Command;
            break;
        case matching :

```

```
        CheckItem = Matching->Command;
        break;
    }
    CheckMenuItem(TChessMenu->Handle, CheckItem, MF_CHECKED);
}
```

(29) 函数名称: void TMainForm::UnCheckLevelMenu(LEVELTYPE level)

说明: 自定义子菜单项去勾函数。改变选中子菜单项“容易”、“每次移动时间”、“总时间”、“匹配”、“无限制”、“回合搜索”、“将死搜索”之一时,在该项前去掉勾。

函数定义如下:

```
void TMainForm::UnCheckLevelMenu(LEVELTYPE level)
{
    int CheckItem = Easy->Command;

    switch (level)
    {
        case normal :
            CheckItem = MoveTime->Command;
            break;
        case fullgametime :
            CheckItem = TotalTime->Command;
            break;
        case plysearch :
            CheckItem = Ply1->Command;
            break;
        case easygame :
            CheckItem = Easy->Command;
            break;
        case infinite :
            CheckItem = Infinite->Command;
            break;
        case matesearch :
            CheckItem = Mate->Command;
            break;
        case matching :
            CheckItem = Matching->Command;
            break;
    }
}
```

```

    CheckMenuItem(TChessMenu->Handle, CheckItem, MF_UNCHECKED);
}
//-----

```

(30) 函数名称: void __fastcall TMainForm::EasyClick(TObject *Sender)

说明: 主菜单项“容易”事件处理函数。选中子菜单项“容易”时, 在该项前打勾。
对弈时在棋盘上对下一步要吃掉的棋子用正方形框住。

函数定义如下:

```

void __fastcall TMainForm::EasyClick(TObject *Sender)
{
    if (Level != easygame)
    {
        UnCheckLevelMenu(Level);
        CheckMenuItem(TChessMenu->Handle, Easy->Command, MF_CHECKED);
    }

    Level = easygame;
    AverageTime = 5.;
    MaxLevel = MAXPLY;
    PrintCurLevel();
}
//-----

```

(31) 函数名称: void __fastcall TMainForm::DoneClick(TObject *Sender)

说明: 主菜单项“完成”事件处理函数。测试棋盘中的棋子摆放是否正确。如果不正确, 则弹出错误对话框, 直至正确才退出编辑状态。

函数定义如下:

```

void __fastcall TMainForm::DoneClick(TObject *Sender)
{
    SQUARETYPE sq;
    int TotalCount[2] = { 0, 0 };
    int KingCount[2] = { 0, 0 };
    int AssistCount[2] = { 0, 0 };
    int BishopCount[2] = { 0, 0 };
    int KnightCount[2] = { 0, 0 };
    int RookCount[2] = { 0, 0 };
    int GunnerCount[2] = { 0, 0 };
    int PawnCount[2] = { 0, 0 };
    bool Done = false;
    bool RKingNumDone, BKingNumDone, RAssistNumDone, BAssistNumDone;
}

```

```

bool RBishopNumDone, BBishopNumDone, RKnightNumDone, BKnightNumDone;
bool RRookNumDone, BRoookNumDone, RGunnerNumDone, BGunnerNumDone;
bool RPawnNumDone, BPawnNumDone;

RKingNumDone = BKingNumDone = RAssistNumDone = BAssistNumDone = true;
RBishopNumDone = BBishopNumDone = RKnightNumDone = BKnightNumDone = true;
RRookNumDone = BRoookNumDone = RGunnerNumDone = BGunnerNumDone = true;
RPawnNumDone = BPawnNumDone = true;

bool RKingPosDone, BKingPosDone, RAssistPosDone, BAssistPosDone;
bool RBishopPosDone, BBishopPosDone, RPawnPosDone, BPawnPosDone;
RKingPosDone = BKingPosDone = RAssistPosDone = BAssistPosDone = true;
RBishopPosDone = BBishopPosDone = RPawnPosDone = BPawnPosDone = true;

AnsiString ErrorNumText = "";
AnsiString ErrorPosText = "";

if (Modified)
{
    for (sq = 0; sq < 0x99; sq++)
        if (((sq>>4)<10) && ((sq % 16)<9) && ((sq | 0xff)>=0))
        {
            if (ChessBoard[sq].piece != empty)
            {
                TotalCount[ChessBoard[sq].color]++;
                if (ChessBoard[sq].piece == king)
                    KingCount[ChessBoard[sq].color]++;
                if (ChessBoard[sq].piece == assist)
                    AssistCount[ChessBoard[sq].color]++;
                if (ChessBoard[sq].piece == bishop)
                    BishopCount[ChessBoard[sq].color]++;
                if (ChessBoard[sq].piece == knight)
                    KnightCount[ChessBoard[sq].color]++;
                if (ChessBoard[sq].piece == rook)
                    RookCount[ChessBoard[sq].color]++;
                if (ChessBoard[sq].piece == gunner)
                    GunnerCount[ChessBoard[sq].color]++;
                if (ChessBoard[sq].piece == pawn)
                    PawnCount[ChessBoard[sq].color]++;
            }
        }
}

```



```
    }  
  }  
  
  if (KingCount[red] != 1)  
  {  
    ErrorNumText = "帅";  
    RKingNumDone = false;  
  }  
  if (KingCount[black] != 1)  
  {  
    ErrorNumText = ErrorNumText + "将";  
    BKingNumDone = false;  
  }  
  if (AssistCount[red] > 2)  
  {  
    ErrorNumText = ErrorNumText + "仕";  
    RAssistNumDone = false;  
  }  
  if (AssistCount[black] > 2)  
  {  
    ErrorNumText = ErrorNumText + "士";  
    BAssistNumDone = false;  
  }  
  if (BishopCount[red] > 2)  
  {  
    ErrorNumText = ErrorNumText + "相";  
    RBishopNumDone = false;  
  }  
  if (BishopCount[black] > 2)  
  {  
    ErrorNumText = ErrorNumText + "象";  
    BBishopNumDone = false;  
  }  
  if ((KnightCount[red] > 2) || (KnightCount[black] > 2))  
  {  
    ErrorNumText = ErrorNumText + "马";  
    RKnightNumDone = BKnightNumDone = false;  
  }  
  if ((RookCount[red] > 2) || (RookCount[black] > 2))
```

体字代

```

        {
            ErrorNumText = ErrorNumText + "车";
            RRoNumDone = BRoNumDone = false;
        }
    if (GunnerCount[red] > 2)
    {
        ErrorNumText = ErrorNumText + "炮";
        RGunnerNumDone = false;
    }
    if (GunnerCount[black] > 2)
    {
        ErrorNumText = ErrorNumText + "炮"; //黑方的炮实为异体字，此处用正
        BGunnerNumDone = false;
    }
    if (PawnCount[red] > 5)
    {
        ErrorNumText = ErrorNumText + "兵";
        RPawnNumDone = false;
    }
    if (PawnCount[black] > 5)
    {
        ErrorNumText = ErrorNumText + "卒";
        BPawnNumDone = false;
    }

    ErrorNumText = ErrorNumText + "的数目不正确";

    for (sq = 0; sq < 0x99; sq++)
    if (((sq >> 4) < 10) && ((sq % 16) < 9) && ((sq | 0xff) >= 0))
    {
        if (ChessBoard[sq].color == red)
            if (ChessBoard[sq].piece == king)
                if (((sq >> 4) > 2) || ((sq % 16) > 5) || ((sq % 16) < 3))
                    RKingPosDone = false;

        if (ChessBoard[sq].color == black)
            if (ChessBoard[sq].piece == king)
                if (((sq >> 4) < 7) || ((sq % 16) > 5) || ((sq % 16) < 3))
                    BKingPosDone = false;
    }

```

```

if (ChessBoard[sq].color == red)
    if (ChessBoard[sq].piece == assist)
    {
        if (((sq >> 4) > 2) || ((sq % 16) > 5) || ((sq % 16) < 3))
            RAssistPosDone = false;
        if ((sq == 0x13) || (sq == 0x15) || (sq == 4) || (sq == 0x24))
            RAssistPosDone = false;
    }

if (ChessBoard[sq].color == black)
    if (ChessBoard[sq].piece == assist)
    {
        if (((sq >> 4) < 7) || ((sq % 16) > 5) || ((sq % 16) < 3))
            BAssistPosDone = false;
        if ((sq == 0x83) || (sq == 0x55) || (sq == 0x94) || (sq == 0x74))
            BAssistPosDone = false;
    }

if (ChessBoard[sq].color == red)
    if (ChessBoard[sq].piece == bishop)
        if ((sq != 2) && (sq != 6) && (sq != 0x20) && (sq != 0x24)
            && (sq != 0x28) && (sq != 0x42) && (sq != 0x46))
            RBishopPosDone = false;

if (ChessBoard[sq].color == black)
    if (ChessBoard[sq].piece == bishop)
        if ((sq != 0x52) && (sq != 0x56) && (sq != 0x70) && (sq != 0x74)
            && (sq != 0x78) && (sq != 0x92) && (sq != 0x96))
            BBishopPosDone = false;

if (ChessBoard[sq].color == red)
    if (ChessBoard[sq].piece == pawn)
    {
        if ((sq < 0x50) && (sq >= 0x30 ))
            if ((sq == 0x31) || (sq == 0x33) || (sq == 0x35) || (sq == 0x37)
                || (sq == 0x41) || (sq == 0x43) || (sq == 0x45) || (sq ==
0x47))

                RPawnPosDone = false;
            if ((sq >> 4) < 3)

```

```

        RPawnPosDone = false;
    }

    if (ChessBoard[sq].color == black)
        if (ChessBoard[sq].piece == pawn)
        {
            if ((sq >= 0x50) && (sq < 0x70))
                if ((sq == 0x51) || (sq == 0x53) || (sq == 0x55) || (sq == 0x57)
                    || (sq == 0x61) || (sq == 0x63) || (sq == 0x65) || (sq ==
0x67))

                    BPawnPosDone = false;
                    if ((sq >> 4) >= 7)
                        BPawnPosDone = false;
                }
        }

    if (RKingPosDone == false)
        ErrorPosText = ErrorPosText + "帅";
    if (BKingPosDone == false)
        ErrorPosText = ErrorPosText + "将";
    if (RAssistPosDone == false)
        ErrorPosText = ErrorPosText + "仕";
    if (BAssistPosDone == false)
        ErrorPosText = ErrorPosText + "士";
    if (RBishopPosDone == false)
        ErrorPosText = ErrorPosText + "相";
    if (BBishopPosDone == false)
        ErrorPosText = ErrorPosText + "象";
    if (RPawnPosDone == false)
        ErrorPosText = ErrorPosText + "兵";
    if (BPawnPosDone == false)
        ErrorPosText = ErrorPosText + "卒";

    ErrorPosText = ErrorPosText + "的位置错误";

    if ((TotalCount[red] <= 16) && (TotalCount[black] <= 16) &&
        RKingNumDone && BKingNumDone && RAssistNumDone && BAssistNumDone
        && RBishopNumDone && BBishopNumDone && RKnightNumDone &&
BKnightNumDone

```

```

        && RRookNumDone && BRookNumDone && RGunnerNumDone &&
BGunnerNumDone
        && RPawnNumDone && BPawnNumDone)
    {
        ResetNewPos();
        if (Attacks(Player, PieceTable[Opponent][0].isquare))
            ErrorPosText = "将(或帅)" + ErrorPosText;

        if (!Attacks(Player, PieceTable[Opponent][0].isquare)&&
            RKingPosDone && BKingPosDone && RAssistPosDone && BAssistPosDone
            && RBishopPosDone && BBishopPosDone && RPawnPosDone &&
BPawnPosDone)
            Done = true;
        else
            Error(ErrorPosText.c_str());
    }
    else
        Error(ErrorNumText.c_str());

}
else
    NormalSetup();

if (Done)
    NormalSetup();

}
//-----

```

(32) 函数名称: void __fastcall TMainForm::Hint1Click(TObject *Sender)

说明: 主菜单项“提示”事件处理函数。把程序计算得出的最好走法在信息框中显示出来。

函数定义如下:

```

void __fastcall TMainForm::Hint1Click(TObject *Sender)
{
    SearchHintMove();
    DisplayHint();
}
//-----

```

(33) 函数名称: void __fastcall TMainForm::PassClick(TObject *Sender)

说明: 主菜单项“对弈”事件处理函数。每点击一次, 程序走一步, 每走完一次, 程序轮流改变对手。

函数定义如下:

```
void __fastcall TMainForm::PassClick(TObject *Sender)
{
    if (Level == easygame)
        HideAttacks();
    CurPlayer = Opponent;
    ComputerColor = Player;
    ComputersTurn();
}
```

//-----

(34) 函数名称: void __fastcall TMainForm::RedTurnClick(TObject *Sender)

说明: 主菜单项“轮流”子菜单项“红方”事件处理函数。选中子菜单项“红方”时, 在该项前打勾。同时设置为当前棋手。

函数定义如下:

```
void __fastcall TMainForm::RedTurnClick(TObject *Sender)
{
    CheckMenuItem(TChessEditMenu->Handle, BlackTurn->Command, MF_UNCHECKED);
    CheckMenuItem(TChessEditMenu->Handle, RedTurn->Command, MF_CHECKED);
    CurPlayer = red;
    ComputerColor = black;
    if (CurPlayer != Player)
    {
        Opponent = ComputerColor;
        Player = ProgramColor = CurPlayer;
    }
}
```

//-----

(35) 函数名称: void __fastcall TMainForm::BlackTurnClick(TObject *Sender)

说明: 主菜单项“轮流”子菜单项“黑方”事件处理函数。选中子菜单项“黑方”时, 在该项前打勾。同时设置为当前棋手。

函数定义如下:

```
void __fastcall TMainForm::BlackTurnClick(TObject *Sender)
```

```

{
    CheckMenuItem(TChessEditMenu->Handle, BlackTurn->Command, MF_CHECKED);
    CheckMenuItem(TChessEditMenu->Handle, RedTurn->Command, MF_UNCHECKED);
    CurPlayer = black;
    ComputerColor = red;
    if (CurPlayer != Player)
    {
        Opponent = ComputerColor;
        Player = ProgramColor = CurPlayer;
    }
}

```

//-----

(36) 函数名称: void __fastcall TMainForm::ColorsClick(TObject *Sender)

说明: 主菜单项“颜色”事件处理函数。弹出颜色对话框, 设置棋盘背景颜色。
Invalidate 函数请求重画。

函数定义如下:

```

void __fastcall TMainForm::ColorsClick(TObject *Sender)
{
    ColorDlg->ShowModal();
    hColorBrush = CreateSolidBrush(RGB(ColorSquareColors[0],
ColorSquareColors[1], ColorSquareColors[2]));
    Invalidate();
}
//-----

```

(37) 函数名称: void __fastcall TMainForm::PieceValuesClick(TObject *Sender)

说明: 主菜单项“棋子值”事件处理函数。模态弹出设置棋子值对话框, 设置每个
棋子(士、象、马、车、炮、兵)的估价值。

函数定义如下:

```

void __fastcall TMainForm::PieceValuesClick(TObject *Sender)
{
    PieceValueDlg->ShowModal();
}
//-----

```

(38) 函数名称: void __fastcall TMainForm::BestPathClick(TObject *Sender)

说明： 主菜单项“最好路线”事件处理函数。翻转子菜单项“最好路线”时，改变该项前的勾号。勾住“最好路线”时，显示对弈时的最好路线。去掉勾时，则不显示对弈时的最好路线。

函数定义如下：

```
void __fastcall TMainForm::BestPathClick(TObject *Sender)
{
    ShowBestPath = !ShowBestPath;

    if (ShowBestPath)
        CheckMenuItem(TChessMenu->Handle, BestPath->Command, MF_CHECKED);
    else
    {
        CheckMenuItem(TChessMenu->Handle, BestPath->Command, MF_UNCHECKED);
        InfoForm->BestPath->Caption="";
    }

}

//-----
```

(39) 函数名称：void __fastcall TMainForm::MouseDownClick(TObject *Sender)

说明： 主菜单项“拖动”事件处理函数。选中该子菜单项时，在该项前打勾号。当移动棋子时，采用动画方式。

函数定义如下：

```
void __fastcall TMainForm::MouseDownClick(TObject *Sender)
{
    POINT Point;

    if (!Dragging && GotStartSquare)
    {
        DrawNormalBitmap(MoveStartSquare);
        GetCursorPos(&Point);
        DragStart(MoveStartSquare, Point);
    }

    Dragging = true;
    CheckMenuItem(TChessMenu->Handle, MouseDrag->Command, MF_CHECKED);
    CheckMenuItem(TChessMenu->Handle, MouseSelect->Command, MF_UNCHECKED);

}
```



```
//-----
```

(40) 函数名称: void __fastcall TMainForm::MouseSelectClick(TObject *Sender)

说明: 主菜单项“选择”事件处理函数。选中该子菜单项时, 在该项前打勾号。移动棋子时, 采用棋子反相方式。

函数定义如下:

```
void __fastcall TMainForm::MouseSelectClick(TObject *Sender)
{
    DragEnd(true);
    Dragging = false;
    CheckMenuItem(TChessMenu->Handle, MouseDrag->Command, MF_UNCHECKED);
    CheckMenuItem(TChessMenu->Handle, MouseSelect->Command, MF_CHECKED);
    if (GotStartSquare)
        DrawInvertedBitmap(MoveStartSquare);
}
//-----
```

(41) 函数名称: void __fastcall TMainForm::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)

说明: 主窗体鼠标移动事件处理函数。选中棋子时, 在当前鼠标位置显示棋子(动画)。

函数定义如下:

```
void __fastcall TMainForm::FormMouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y)
{
    if (!GotStartSquare || !Dragging)
        return;
    Drag(Point(X, Y));
}
//-----
```

(42) 函数名称: void __fastcall TMainForm::SoundClick(TObject *Sender)

说明: 主菜单项“声音”事件处理函数。翻转子菜单项“声音”时, 改变该项前的勾号。只有系统中配有多媒体喇叭时, 才有可能发出声音。

函数定义如下:

```
void __fastcall TMainForm::SoundClick(TObject *Sender)
{
    SoundOn = !SoundOn;
```

```

    if (SoundOn)
        Sound->Checked = true;
    else
        Sound->Checked = false;

}
//-----

```

(43) 函数名称: void __fastcall TMainForm::Timer1Timer(TObject *Sender)

说明: 主窗体定时器事件处理函数。每当程序发出 WM_TIMER 消息时, 在信息窗体中显示下棋时间。

函数定义如下:

```

void __fastcall TMainForm::Timer1Timer(TObject *Sender)
{
    DisplayTime();
}
//-----

```

(44) 函数名称: void __fastcall TMainForm::FormDestroy(TObject *Sender)

说明: 主窗体窗体关闭事件处理函数。释放系统资源。对于非可视化指针, 用 new 创建, 还需用 delete 撤消。

函数定义如下:

```

void __fastcall TMainForm::FormDestroy(TObject *Sender)
{
    delete FileName;
    delete InfoForm;
    delete EditForm;
    DeleteObject(hColorBrush);
    for (int i = 0; i < 7; i++)
    {
        DeleteObject(MaskArray[i]);
        for (int j = 0; j < 2; j++)
            DeleteObject(PieceBmpArray[i][j]);
    }
    Timer1->Enabled=false;
}
//-----

```

(45) 函数名称: void __fastcall TMainForm::MoveTimeClick(TObject *Sender)

说明： 主菜单项“每次移动时间”事件处理函数。选中该子菜单项时，在该项前打勾号。模态弹出“设置每次移动秒数”对话框，设置每次移动程序“想问题”的时间。如果输入没有两位小数的浮点数值，弹出错误对话框。

函数定义如下：

```
void __fastcall TMainForm::MoveTimeClick(TObject *Sender)
{
    char *SecsPerMove = new char[40];
    COLORTYPE color;
    int temp;
    sprintf(SecsPerMove, "%.2f", AverageTime);

    if (InputSecPerMoveDlg->ShowModal() == mrOk)
    {
        SecsPerMove = InputSecPerMoveDlg->Edit1->Text.c_str();
        double NewMoveTime = atof(SecsPerMove);

        if (NewMoveTime > 0.)
        {
            if (Level != normal)
            {
                UnCheckLevelMenu(Level);
                CheckMenuItem(TChessMenu->Handle, MoveTime->Command, MF_CHECKED);
            }
            Level = normal;
            AverageTime = NewMoveTime;
            PrintCurLevel();
            /*
            在 C++Builder 中枚举常量加一个常数，不能成为后一个常量。
            只得采用整数间接来达到目的。
            */
            for (temp=0; temp<=1; temp++)
            {
                if (temp==0)
                    color = red;
                if (temp==1)
                    color = black;
                ChessTime[color].totaltime = (MoveNumber / 2) * AverageTime;
            }
        }
    }
}
```

```

        MaxLevel = MAXPLY;
    }
    else
        Error("时间不正确。请重输。");
    }
    delete SecsPerMove;

}
//-----

```

(46) 函数名称: void __fastcall TMainForm::TotalTimeClick(TObject *Sender)

说明: 主菜单项“总时间”事件处理函数。选中该子菜单项时, 在该项前打勾号。

模态弹出“设置每次游戏分钟数”对话框, 设置每次游戏程序的时间。如果输入没有两位小数的浮点数值, 弹出错误对话框。

函数定义如下:

```

void __fastcall TMainForm::TotalTimeClick(TObject *Sender)
{
    char *TotalMoveTime = new char[40];
    int temp;
    COLORTYPE color;
    sprintf(TotalMoveTime, "%.2f", AverageTime);

    if (InputMinPerGameDlg->ShowModal() == mrOk)
    {
        TotalMoveTime = InputMinPerGameDlg->Edit1->Text.c_str();
        double NewMoveTime = atof(TotalMoveTime);
        if (NewMoveTime > 0.)
        {
            if (Level != fullgametime)
            {
                UnCheckLevelMenu(Level);
                CheckMenuItem(TChessMenu->Handle, TotalTime->Command, MF_CHECKED);
            }
            Level = fullgametime;
            AverageTime = NewMoveTime;
            PrintCurLevel();
        }
    }
}

```

在 C++Builder 中枚举常量加一个常数, 不能成为后一个常量。

只得采用整数间接来达到目的。

```

*/
    for (temp=0;temp<=1;temp++)
    { { if (temp==0)
        color = red;
        if (temp==1)
            color= black;
        ChesTime[color].totaltime = (MoveNumber / 2) * AverageTime;
    }
    }
    MaxLevel = MAXPLY;
}
else
    Error("时间不正确。请重输。");
}
delete TotalMoveTime;

}
//-----

```

(47) 函数名称: void __fastcall TMainForm::MatchingClick(TObject *Sender)

说明: 主菜单项“匹配”事件处理函数。选中该子菜单项时, 在该项前打勾号。同时设置信息窗体中的级别文本。

函数定义如下:

```

void __fastcall TMainForm::MatchingClick(TObject *Sender)
{
    if (Level != matching)
    {
        UnCheckLevelMenu(Level);
        CheckMenuItem(TChessMenu->Handle, Matching->Command, MF_CHECKED);
    }
    Level = matching;
    PrintCurLevel();
}
//-----

```

(48) 函数名称: void __fastcall TMainForm::Exit1Click(TObject *Sender)

说明: 主菜单项“退出”事件处理函数。

函数定义如下:

```

void __fastcall TMainForm::Exit1Click(TObject *Sender)

```

```
{
    MainForm->Close();
}
```

(49) 函数名称: void TMainForm::Exit(TMessage &Msg)

说明: 自定义消息处理函数。响应 IDM_EXIT 消息请求关闭窗口。

函数定义如下:

```
void TMainForm::Exit(TMessage &Msg)
{
    MainForm->Exit1Click(Exit1);
}
//-----
```

(50) 函数名称: void TMainForm::EndGame()

说明: 自定义结束游戏函数。动态创建对话框。注意, 别忘记撤消创建的资源。

函数定义如下:

```
void TMainForm::EndGame()
{
    TEndChessGameDlg *EndChessGameDlg = new TEndChessGameDlg(Application,
(AnsiString) EndGameMessage);
    EndChessGameDlg->ShowModal();
    delete EndChessGameDlg;
    return;
}
//-----
```

(51) 函数名称: void __fastcall TMainForm::FormCloseQuery(TObject *Sender, bool &CanClose)

说明: 关闭应用程序时, 为避免数据丢失, 应对一些重要的数据进行保存。解决问题的关键是 FormCloseQuery。当窗口关闭时, 调用 VCL 函数 TForm::Close。Close 并不是自己关闭窗体。首先它申请 FormCloseQuery 句柄允许。程序退出前, 必须经过 FormCloseQuery 句柄函数允许。

当选择【游戏】|【退出】命令或单击程序标题栏的关闭图标时, FormCloseQuery 将执行。如果 FormCloseQuery 设置 CanClose 为 false, 窗体将不关闭。设置 CanClose 为 true 则允许程序关闭。在 FormCloseQuery 返回前, 调用 QuitProgram()保存 chess.ini 文件。

函数定义如下:

```
void __fastcall TMainForm::FormCloseQuery(TObject *Sender, bool &CanClose)
{
    if (ComputerThinking || AutoPlay)
```

```

{
    ::PostMessage(MainForm->Handle, WM_COMMAND, IDM_STOP, 0L);
    ::PostMessage(MainForm->Handle, WM_COMMAND, IDM_EXIT, 0L);
    CanClose=false;
}

QuitProgram();
CanClose = true;
}
//-----

```

4.1.3 交互菜单设计

菜单的设计要服从程序整体功能的需要，往往是一个菜单项对应一项功能。

1. 主菜单 TChessMenu 的设计

设计主菜单必须主窗体上先做一个可视化菜单组件，然后就已有的菜单组件进一步制作其菜单项。以下是先做一个可视化菜单组件的步骤：

- (1) 单击 Standard 组件选项卡中的 MainMenu 组件。
- (2) 用鼠标单击主窗体，如图 4.9 所示。
- (3) 选择 View | Object Inspector 命令打开对象观察器，把其中的 Properties 选项卡中的 Name 属性值更改为“TChessMenu”。

这样一个可视化菜单组件就做好了。

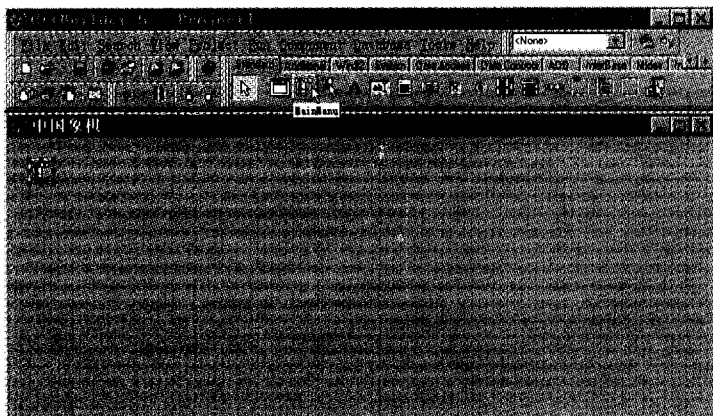


图 4.9 设计可视菜单组件

下面讲述主菜单 TChessMenu 中菜单项的设计方法。

- (1) 双击主菜单 TChessMenu 组件，弹出菜单设计器，如图 4.10 所示。
- (2) 菜单设计器中有一个给定的空菜单项，将其 Caption 属性设为“游戏”，如图 4.11 所示。

- (3) 接下来, 按下回车键, 在其下方出现的空子菜单项, 将其 Caption 属性设为“新建”, 如图 4.12 所示。
- (4) 同理继续建立其他子菜单项,
- (5) 如果在空子菜单项的 Caption 栏中输入连字符“—”, 将产生分隔条, 如图 4.13 所示。
- (6) 在“游戏”右侧出现的空子菜单项, 将其 Caption 属性设为“编辑”。如图 4.14 所示。要说明的是, 在 C++Builder 5.0 中设置菜单属性 AutoHotkeys = maAutoMatic 时, 程序运行时在每个菜单项后面自动加上快捷键。如果不喜欢自动加上快捷键, 应设置属性 AutoHotkeys = maManual, 如图 4.15、图 4.16 所示。

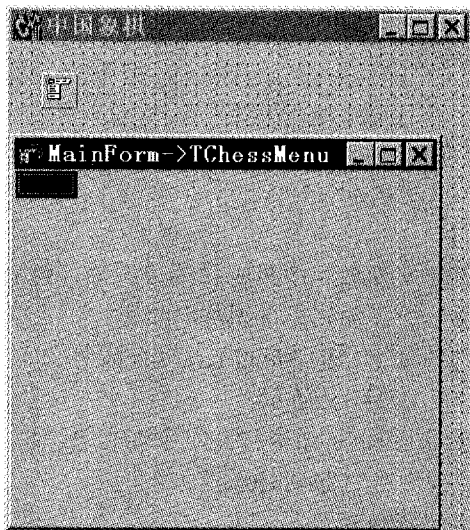


图 4.10 弹出的 TChessMenu 菜单设计器

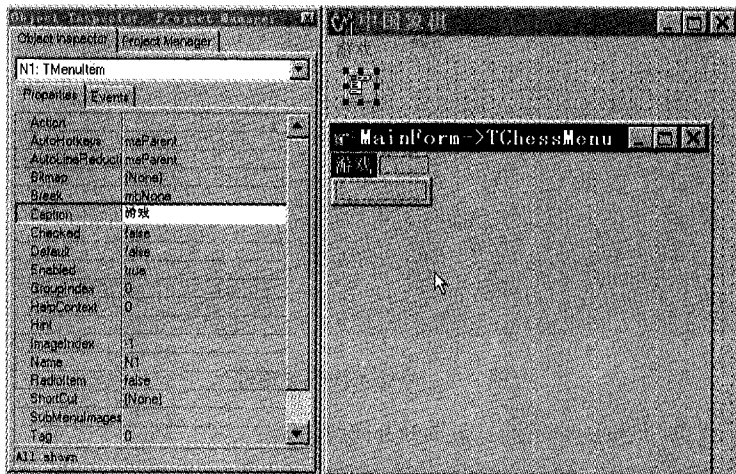


图 4.11 设置 Caption 属性为“游戏”

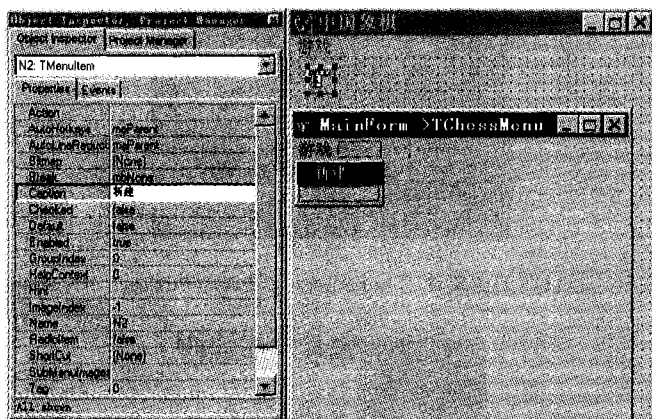


图 4.12 设置 Caption 属性为“新建”

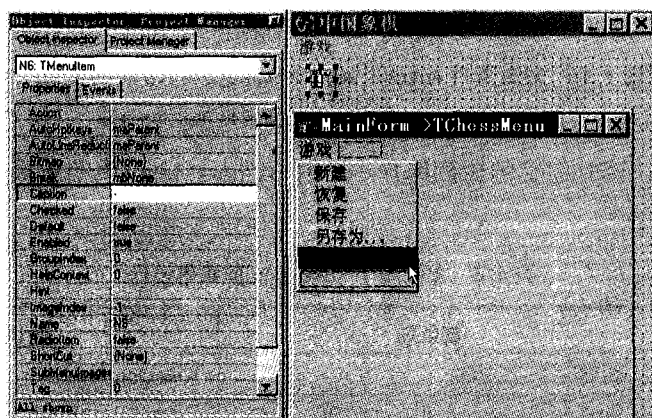


图 4.13 设置 Caption 属性为“-”

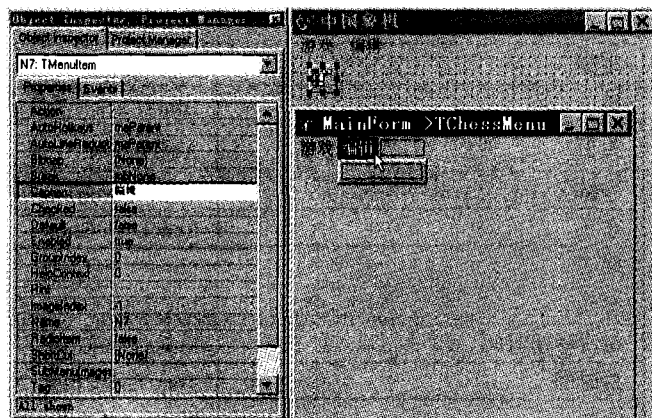


图 4.14 设置 Caption 属性为“编辑”

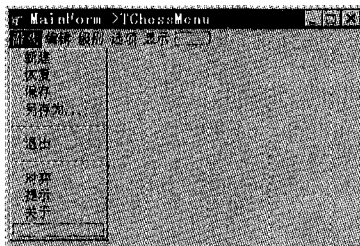


图 4.15 设计时的主菜单 TChessMenu 属性 AutoHotkeys = maManual

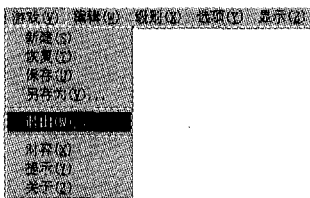


图 4.16 设置属性 AutoHotkeys = maAutoMatic 时的菜单,

程序运行时在每个菜单项后面自动加上快捷键

根据上述步骤，建立完整的轮到用户走子时 TChessMenu 主菜单项的设计如表 4.3 所示。

表 4.3 主菜单 TChessMenu 的菜单设计

菜 单	菜单项	回调函数
游戏	新建	NewGame1Click
	恢复	RestoreGame1Click
	保存	SaveGame1Click
	另存为...	SaveGameAsClick
	退出	Exit1Click
	对弈	PassClick
	提示	Hint1Click
	关于	AboutClick
编辑	悔棋	UndoMoveClick
	重下	RedoUndoClick
	摆子	MovePieceClick
级别	容易	EasyClick
	每次移动时间...	MoveTimeClick
	总时间	TotalTimeClick
	匹配	MatchingClick
	无限制	InfiniteClick
	回合搜索	Ply1Click
	将死搜索	MateClick
选项	两人对弈	TwoPlayerClick
	演示模式	DemoClick

续表

菜 单	菜单项	回调函数
显示	棋子值...	PieceValuesClick
	拖动	MouseDownClick
	选择	MouseSelectClick
	声音	SoundClick
	颜色...	ColorsClick
	反转棋盘	ReverseClick
	最好路线	BestLineClick

2. 主菜单 TChessEditMenu 的设计

设计主菜单必须主窗体上先做一个可视化菜单组件, 然后就已有的菜单组件进一步制作其菜单项。以下是先做一个可视化菜单组件的步骤(参见主菜单项 TChessMenu 的设计):

- (1) 单击 IDE 可视化组件——Standard 组件选项卡中的 MainMenu 组件。
- (2) 在主窗体中单击。
- (3) 选择 View | Object Inspector 命令, 打开对象观察器, 把其中的 Properties 选项卡中的 Name 属性值更改为“TChessEditMenu”, 就可以产生一个可视化菜单组件。

下面讲述 TChessEditMenu 菜单项的设计方法。

- (1) 双击主菜单 TChessEditMenu 组件, 弹出菜单设计器。
- (2) 菜单设计器中有一个给定的空菜单项, 将其 Caption 属性设为“完成”。
- (3) 接下来, 按下回车键, 在其右侧出现的空子菜单项, 将其 Caption 属性设为“清除”。
- (4) 同理继续建立子菜单项“轮流”, 单击鼠标右键, 在弹出菜单中选中 Create Submenu, 在其右侧出现的空级联菜单中, 按照上面所说, 建立“红方”、“黑方”两个级联菜单。
- (5) 最后, 按同样道理建立“忽略”子菜单项。

轮到用户编辑棋盘中的棋子时 TChessEditMenu 主菜单项(如图 4.17 所示)的设计如表 4.4 所示:

表 4.4 主菜单 TChessEditMenu 的菜单设计

菜 单	菜单项	回调函数
完成		DoneClick
清除		ClearClick
轮流	红方	RedTurnClick
	黑方	BlackTurnClick
忽略		CancelClick

3. 主菜单 TChessThinkMenu 的设计

轮到机器走子时主菜单 TChessThinkMenu 只包括“停止”菜单项(见图 4.18), 主要属性设置如下:

```
Name=Stop  
AutoHotkeys = maManual  
Caption = “停止”  
Visible = true
```

当用户单击时, 通过 PeekMessage 函数捕获其 ID 号 Stop->Command。

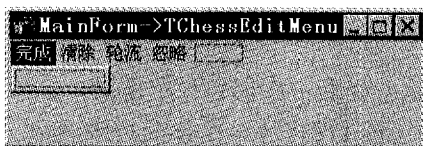


图 4.17 设计时的主菜单 TChessEditMenu 的属性 AutoHotkeys = maManual

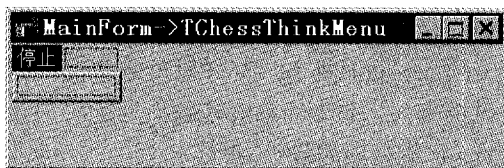


图 4.18 设计时的主菜单 TChessThinkMenu 的属性 AutoHotkeys = maManual

4.1.4 运行时修改菜单

在 CHESS 项目中如果用户需要改变程序的思考时间, 可以通过单击“停止”菜单来响应用户的干预, 动态地进行改变。这在程序运行期间通过改变窗体的 Menu 属性来实现。

CHESS 项目中有 3 个主菜单 TChessMenu、TChessThinkMenu、TChessEditMenu, 在 MainForm1.cpp 的成员函数 void TMainForm::NormalSetup()中使用如下的代码可动态地改变主菜单。

```
MainForm->Menu=TChessMenu;
```

这一行的代码表示名为 MainForm 主窗体的菜单变成名为 TChessMenu 的主菜单。此时轮到用户走子。

在 MainForm1.cpp 的成员函数 void TMainForm::ComputersTurn()中使用如下代码可动态地改变主菜单:

```
MainForm->Menu=TChessThinkMenu;
```

这一行代码使名为 MainForm 主窗体的菜单变成名为 TChessThinkMenu 的主菜单。此时轮到程序思考。

在 MainForm1.cpp 的成员函数 void __fastcall TMainForm::MovePieceClick(TObject *Sender)中使用以下代码可动态地改变主菜单:

```
MainForm->Menu=TChessEditMenu;
```

这一行代码使名为 MainForm 主窗体的菜单变成名为 TChessEditMenu 的主菜单。此时可进行棋盘编辑。

4.1.5 时钟组件

定时器在 C++Builder 程序中运行时并不出现, 与其相应的属性可以用与其等价的 Windows API 函数来实现。如:

```
Enabled = False 对应 KillTimer(Handle,1);  
Enabled = True 对应 SetTimer(Handle,1,1000,0);
```

设置定时器的目的主要是利用时钟事件来刷新当前的时间, 显示对弈双方的用时。具体作法是:

- (1) 单击 System 组件选项卡中的 Timer 组件,
- (2) 在主窗体 MainForm 中单击。

于是一个定时器就出现在主窗体中了, 默认的名字是 Timer1。

1. Timer 的主要属性

要求 Timer1 每隔一秒发送 WM_TIMER 消息。操作如下:

- (1) 单击对象观察器中的 Properties 选项卡。
- (2) 在相应的属性中输入值。

其属性如下:

```
Interval =1000  
Enabled = False  
OnTimer = Timer1Timer
```

一开始时钟事件是禁止的, 只有当 Enabled 变为 True 时才每隔一秒系统就触发一次时钟事件。

2. Timer 的方法

要求每收到一个 WM_TIMER 消息就显示棋钟数字。操作如下:

- (1) 单击对象观察器中的 Events 选项卡。
- (2) 双击 OnTimer 属性在代码窗体输入如下代码:

```
void __fastcall TMainForm::Timer1Timer(TObject *Sender)  
{  
    DisplayTime();  
}
```

```
}
```

这样每隔一秒就刷新一次用户的用时时间。

4.2 扩展 CHESS 项目

CHESS 项目具有以下功能:

- 对弈功能;
- 棋局保存打开功能;
- 对弈级别设置功能;
- 棋盘颜色变换功能;
- 变换先后手功能;
- 棋局编辑功能等。

前一节只是介绍了 CHESS 项目的基础框架,只能完成一些简单的功能,即项目只包括了主窗体单元(主窗体文件、主窗体函数定义文件、主窗体包含文件),而窗体函数定义的具体操作该如何实现,比如棋盘的实现、使用自己的光标、生成棋子动画、响应用户消息、排除异常错误、显示棋盘编辑子窗体、显示对弈信息子窗体、显示棋钟、打开保存棋盘对话框、改变棋盘颜色、改变棋子估值、定义对局时间、定义走步时间、报告错误、结束对弈等等,所有这些功能需要通过向 CHESS 项目添加新的单元文件或函数定义文件才能做到。具体步骤如下:

- (1) 双击 CHESS.bpr 图标或通过 IDE 环境单击 File | Open Project 命令在弹出的对话框中选中 CHESS.bpr 文件,打开 CHESS 项目;
- (2) 选择 File | New 命令打开 New Items 对话框,在其中选中 Cpp File 项,单击 OK 按钮,打开 CPP 代码编辑器窗口;
- (3) 选择 File | Save As 命令打开 Save Unit As 对话框,在【文件名】文本框中输入 ChessErr.cpp,目的是作为报告错误动态对话框函数定义文件;
- (4) 重复第 2、3 步,建立文件名为 EndGameDlog.cpp(结束游戏动态对话框)、Evaluate.cpp(局面评估函数定义)、GlobalFun.cpp(自定义全局函数定义)、MoveGener.cpp(合法走子函数定义)、TreeSearch.cpp(棋路树搜索函数定义)等的文件;
- (5) 选择 File | New 命令打开 New Items 对话框在其中选中 Header File 项,单击 OK 按钮,打开包含文件代码编辑器窗口;
- (6) 选择 File | Save As 命令打开 Save File As 对话框,在【文件名】文本框中输入 CDefines.h 作为项目自定义数据类型包含文件;
- (7) 重复第 5、6 步,建立文件名为 Chess.h(自定义消息标识包含文件)、ChessErr.h(报告错误动态对话框包含文件)、Global.h(全局自定义函数声明包含文件)等的包含文件;
- (8) 选择 File | New Form 命令新建一个窗体;

- (9) 选择 File | Save As 命令打开 Save Unit As 对话框, 在【文件名】文本框中输入文件名 AboutDlog.cpp(静态象棋说明对话框);
- (10) 重复第 8、9 步, 建立文件名为 ColorDlog.cpp(改变棋盘颜色对话框)、EditForm2.cpp(棋盘编辑器窗体)、InfoForm3.cpp(对弈信息子窗体)、InputMinPerGame.cpp(定义对局时间对话框)、InputPlyDepth.cpp(定义对弈深度对话框)、InputSecPerMove.cpp(定义走步时间对话框)、PieceValueDlog.cpp(改变棋子估值对话框)等的文件;
- (11) 选择 File | Save 命令保存 CHESS 项目。

经过以上几步, C++Builder 会自动把它们添加到 CHESS 项目中, 于是完整的框架经扩充就浮现出来了(附带说明的是, 窗体文件及项目资源文件已隐含地自动建立起来了)。CHESS 项目由原先的 6 个文件增加到一共包括 16 个 CPP 源程序, 9 个窗体文件, 14 个包含文件。但是这些函数定义文件、包含文件基本上是空的, 其代码需要在 IDE 自动管理的基础上由用户添加而成, 以后分章节详细论述; 窗体文件的属性是默认值, 但可以改变, 以后章节将详细论述; 资源文件的创建见第 3 章 3.9 节。

4.2.1 CHESS 项目文件

CHESS 项目可进行实战对弈, 也可摆成各种定式局面, 对弈的同时由程序给出各种提示信息。根据 CHESS 项目的任务, 接下来是确定 CHESS 项目中每个文件的功能及作用。16 个 CPP 源程序的功能如表 4.5 所示, 9 个窗体文件作用说明如表 4.6 所示, 14 个包含文件作用如表 4.7 所示, 另加 3 个资源文件, 如表 4.8 所示。

一个项目文件: Chess.bpr。

表 4.5 源程序文件

CPP 源程序名	说 明
AboutDlog.cpp	关于对话框函数定义文件
Chess.cpp	CHESS 项目主调函数文件
ChessErr.cpp	错误对话框函数定义文件
ColorDlog.cpp	颜色对话框函数定义文件
EditForm2.cpp	编辑窗体函数定义文件
EndGameDlog.cpp	结束游戏对话框函数定义文件
Evaluate.cpp	局面评估函数定义文件
GlobalFun.cpp	全局函数定义文件
InfoForm3.cpp	信息窗体函数定义文件
InputMinPerGame.cpp	输入每局对弈时间对话框函数定义文件
InputPlyDepth.cpp	输入搜索深度对话框函数定义文件
InputSecPerMove.cpp	输入每步对弈时间对话框函数定义文件
MainForm1.cpp	主窗体函数定义文件

续表

CPP 源程序名	说 明
MoveGener.cpp	生成合法棋步函数定义文件
PieceValueDlog.cpp	输入每个棋子估值对话框函数定义文件
TreeSearch.cpp	棋路搜索树函数定义文件

表 4.6 窗体文件

窗体文件名	说 明
AboutDlog.dfm	关于对话框窗体文件
ColorDlog.dfm	颜色对话框窗体文件
EditForm2.dfm	编辑窗体文件
InfoForm3.dfm	信息窗体文件
InputMinPerGame.dfm	输入每局对弈时间对话框窗体文件
InputPlyDepth.dfm	输入搜索深度对话框窗体文件
InputSecPerMove.dfm	输入每步对弈时间对话框窗体文件
MainForm1.dfm	主窗体文件
PieceValueDlog.dfm	棋子值对话框窗体文件

表 4.7 包含文件

包含文件名	说 明
AboutDlog.h	关于对话框包含文件
CDefines.h	全局数据类型定义文件
Chess.h	自定义消息 ID 号文件
ChessErr.h	对弈错误提示对话框包含文件
ColorDlog.h	颜色对话框包含文件
EditForm2.h	编辑窗体包含文件
EndGameDlog.h	结束游戏对话框包含文件
Global.h	全局函数声明包含文件
InfoForm3.h	信息窗体包含文件
InputMinPerGame.h	输入每局对弈时间对话框包含文件
InputPlyDepth.h	输入搜索深度包含文件
InputPlyPerMove.h	输入每步对弈时间对话框包含文件
MainForm1.h	主窗体包含文件
PieceValueDlog.h	输入棋子值对话框包含文件

三个资源文件：Chess.rc、Chess.res、Myrc.res，其实也可合并成一个。

表 4.8 资源文件

资源文件名	说 明
Chess.rc	文本资源文件
Chess.res	由 Chess.rc 编译成的资源文件(参见第 3 章 3.9 节)
MyRc.res	自定义光标文件(参见第 6 章 6.3 节)

以上文件中的函数关系是：程序一运行将显示对弈主窗体(由属性或资源决定)，当主窗体发生某些事件时，将调用相应的回调函数(窗体的方法)，而每个回调函数的具体动作又由 API 函数和自定义函数决定。函数的调用关系形成一个阶梯层次。

4.2.2 功能菜单

功能菜单是用户与程序交互操作的主要途径，菜单越多，其功能就越强大，程序也就越复杂，CHESS 项目的菜单简述如下：

1. 游戏菜单

游戏菜单主要提供开始、结束游戏及进行文件操作等功能，如图 4.19 所示，其功能如表 4.9 所示。



图 4.19 游戏菜单

表 4.9 游戏菜单

菜单名称	功 能
新建	主窗体进入初始对弈状态。同时清除信息窗体中的提示信息
恢复	调用公用对话框打开组件函数，读入二进制棋盘文件
保存	调用公用对话框保存组件函数，写入二进制棋盘文件
另存为...	调用公用对话框保存组件函数，换名写入二进制棋盘文件
退出	请求关闭窗口
对弈	每单击一次，程序走一步，每走完一次，程序轮流改变对手
提示	把程序计算得出的最好走法在信息框中显示出来
关于	调用自定义静态对话框函数，显示版权信息

2. 编辑菜单

编辑菜单主要提供对棋盘的操作(插入、删除)，如图 4.20 所示，其功能如表 4.6 所示。

表 4.10 编辑菜单

菜单名称	菜单功能
悔棋	至少要动过一次棋子，该菜单项才由灰色变为黑色(激活状态)
重下	至少要悔过一次棋子，该菜单项才由灰色变为黑色(激活状态)
摆子	进入编辑状态，单击鼠标左键，可以使编辑窗体中的棋子透明显示，第二次在主窗体单击时，可以添加棋子，再单击时，擦除棋子

3. 级别菜单

级别菜单通过设置搜索的时间或回合数，从而改变对弈的水平，如图 4.21 所示，其功能如表 4.11 所示。



图 4.20 编辑菜单

图 4.21 级别菜单

表 4.11 级别菜单

菜单名称	菜单功能
容易	在该项前打勾。对弈时在棋盘上对下一步要吃掉的棋子用正方形框住
每次移动时间	在该项前打勾。弹出“设置每次移动秒数”对话框，设置每次移动程序“想问题”的时间。如果输入没有两位小数的浮点数值，弹出错误对话框
总时间	在该项前打勾。弹出“设置每次游戏分钟数”对话框，设置每次游戏程序的时间。如果输入没有两位小数的浮点数值，弹出错误对话框
匹配	在该项前打勾。同时设置信息窗体中的级别文本。由用户设定机器的对弈时间
无限制	改变信息窗体中信息提示为无限制。程序无限制地加大深度搜索最好的一步棋
回合搜索	改变信息窗体中信息提示为回合搜索。程序在未达到设定的深度时一直搜索可能最好的一步棋
将死搜索	改变信息窗体中信息提示为将死搜索。程序无限制地加大深度直到搜索可能将死的一步棋

4. 选项菜单

选项菜单如图 4.22 所示，其功能如表 4.12 所示。

表 4.12 选项菜单

菜单名称	菜单功能
两人对弈	翻转本菜单项“两人对弈”为“单独对弈”
演示模式	轮流成为红方、黑方，即自己与自己对弈
棋子值	设置棋子值对话框，设置每个棋子(士、象、马、车、炮、兵)的估价值
移动	具有两个子菜单。①拖动：在该项前打勾号。当移动棋子时，采用动画方式 ②选择：在该项前打勾号。移动棋子时，采用棋子反相方式
声音	翻转子菜单项“声音”时，改变该项前的勾号。只有系统中配有多媒体喇叭时，才有可能发出声音

5. 显示菜单

显示菜单如图 4.23 所示，其功能如表 4.13 所示。



图 4.22 选项菜单



图 4.23 显示菜单

表 4.13 显示菜单

菜单名称	菜单功能
颜色	弹出颜色对话框，设置棋盘背景颜色
反转棋盘	对换红方和黑方的棋盘
最好路线	翻转子菜单项“最好路线”时，改变该项前的勾号

6. 编辑主菜单

编辑主菜单如图 4.24 所示，其功能如表 4.14 所示。



图 4.24 编辑主菜单



图 4.25 停止主菜单

表 4.14 编辑主菜单

菜单名称	菜单功能
完成	测试棋盘中的棋子摆放是否正确。如果不正确，则弹出错误对话框，直至正确才退出编辑状态
清除	把主窗体中棋盘上棋子全部擦除
轮流	在该项前打勾，同时设置当前玩家
忽略	把主窗体中棋盘上编辑过的棋子全部作废

7. 停止主菜单

停止主菜单的功能是响应用户的干预，停止棋路树搜索，回到用户输入信息的状态。停止主菜单如图 4.25 所示。

4.3 编译、运行 CHESS 项目

当程序设计过程一结束，就可以编译、运行程序了，C++Builder 5.0 提供了几个编译菜单项和运行项，它们具有不同的编译作用，如表 4.15 所示。

表 4.15 编译、运行菜单项

菜 单 项	功 能
Compile Unit	对上次编译后修改的源代码进行编译
Make Project	编译所有文件并生成可执行文件
Build Project	编译项目中的所有文件
Information for Project	显示编译信息和编译状态
Make All Projects	编译项目组中的所有项目文件
Build All Projects	重新编译项目组中所有修改过的项目文件
Run	编译、运行项目文件

表 4.15 中的前 6 项为 Project 主菜单下的子菜单，Run 菜单项是 Run 主菜单下的子菜单。除非用户对全局对象进行了修改，一般不要采用 Build Project 或 Build All Projects 命令。在实际编程过程中，最多时候选择 Make Project，合理地选择编译菜单项有助于节约时间，有利于减小调试范围。

程序编译过程中，难免会出现各种各样的错误，这时就需要调试程序，程序的调试既枯燥又费时间，但调试程序是开发应用程序避免不了的一个环节，有人说，20%的时间进行程序设计，80%的时间进行程序的调试。即使是大名鼎鼎的 Microsoft 公司也不得不频繁发布 Windows 补丁，以不断完善他们的产品。下一章将重点讲述 C++Builder 提供的调试功能。

第 5 章 调试 CHESS 项目

一般说来，无论在编程时如何仔细，程序总会有错误，尤其是像 CHESS 项目这样比较大的程序，各个变量、函数、模块之间或多或少存在一些问题。本章参照调试 CHESS 项目的一些常见手段和方法，讲述程序的调试，可以使用户熟悉调试程序常用的工具，也有利于在以后的应用中能够经常地使用它们。

5.1 程序错误的种类

程序的错误一般分为语法错误、链接错误、运行错误和逻辑错误 4 类。

1. 语法错误

语法错误产生于程序的设计阶段，是指用户的程序代码违反了 C++Builder 的语法规则时发生的错误。C++Builder 能够进行合法性检查，编译时往往会有警告或错误提示信息，这类错误比较容易处理。例如，将编辑组件属性 Edit1->Text="5.00" 改为 Edit.Text="5.00"，重新编译，在代码编辑窗口的下面将有错误提示信息出现。单击错误提示信息中的错误提示，代码编辑器就会迅速定位于 Edit.Text="0." 处，将错误改正，重新编译，程序编译通过。

2. 链接错误

单个模块通过了编译，但是在把几个模块集成测试、链接库文件时却出现错误，这往往是引用函数时出现问题，仔细检查也不难排除错误。

3. 运行错误

程序通过编译后，在运行阶段出现的错误称为运行错误。例如，运算时出现了被零除，代码试图打开一个不存在的文件，使用 C++Builder 内部集成调试器，或者通过单步执行，或者通过观察窗口监视某个变量的取值，对这类错误也容易解决。比如：CHESS 项目中如果在 C 盘没有建立 BcbChess 目录或该目录下没有 BkBrush.bmp 背景位图文件的话，操作系统检测出此状态，程序将中止运行。

4. 逻辑错误

逻辑错误产生在运行阶段，程序中的语句是合法的，而且确实执行了，但是执行的不是需要的操作，它可能不产生错误信息，常表现为程序的运行结果与预期的效果不一致。例如，在 C++Builder 4.0 以前的版本中，在 OWL 中枚举变量可以在循环中加 1 成为后一

个变量, 可在 C++Builder 4.0 中编译成执行文件运行时却进入死循环(如图 5.1 所示), 而在 C++Builder 5.0 中又能正常运行。

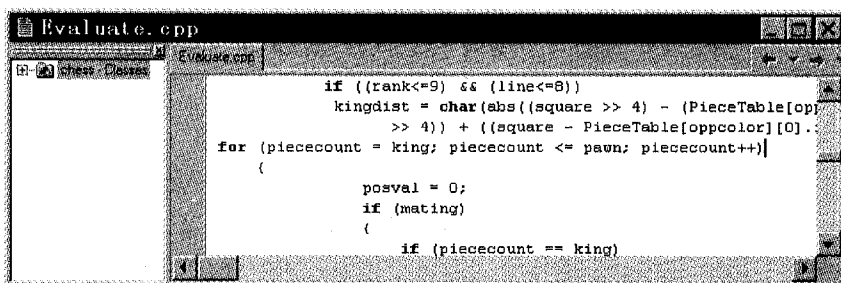


图 5.1 枚举变量 piececount 加 1 进入死循环

这类错误不易被发现, 因此, C++Builder 提供的调试工具主要是针对此类错误。

5.2 C++Builder 集成环境中的调试选项

C++Builder 集成环境中的命令为用户提供了多种调试工具和手段, 熟悉并会使用这些工具和手段将有利提高程序开发效率。首先必须熟悉 Run 菜单下的命令(简称 Run 命令)中提供的大部分调试功能, 如表 5.1 所示。

表 5.1 Run 命令

命令名称	为项目文件传递指定的参数
Step Over	单步执行程序, 不进入函数内部
Trace Into	跟踪执行程序, 进入函数内部
Trace to Next Source Line	将执行点移至程序源代码的下一行
Run to Cursor	执行到代码编辑器中当前光标所在的行
Run Until Return	全速执行例程的代码, 直到返回为止
Show Execution Point	返回到代码编辑器中执行点光标的位置
Program Pause	暂停运行程序
Program Reset	终止运行程序后重新运行程序
Evaluate/Modify	显示 Evaluate/Modify 对话框
Inspect	显示 Inspect 对话框, 输入要查阅的对象名
Add Watch	打开 Watch Properties 对话框
Add BreakPoints	设置和修改断点

5.3 主要调试手段

观察表 5.1 中 Run 命令可以看出, C++Builder 的主要调试手段有设置断点、观察变量和单步执行等, 而调试手段中最重要的概念是断点。所谓断点, 就是在程序代码的某一行上设置一个标记, 程序执行到这里就暂停, 由调试器接管对程序的控制。

5.3.1 使用 Evaluate/Modify 窗口

图 5.2 所示的是观察 CHESS 项目中结构变量 ZeroMove 的取值。选择 Run | Evaluate/Modify 命令弹出 Evaluate/Modify 窗口，在 Expression 下拉列表框中输入表达式，选择 Watch 按钮可以观察部分表达式的取值。要修改表达式时，可以在 New Value 框中输入一个新的值，然后单击 Modify 按钮。要计算表达式的值，可以单击 Evaluate 按钮，那么符合 C++语法规则的表达式的值就将会出现在 Result 列表框中。通过对变量和表达式的观察，可以判断程序的运行是否正常。

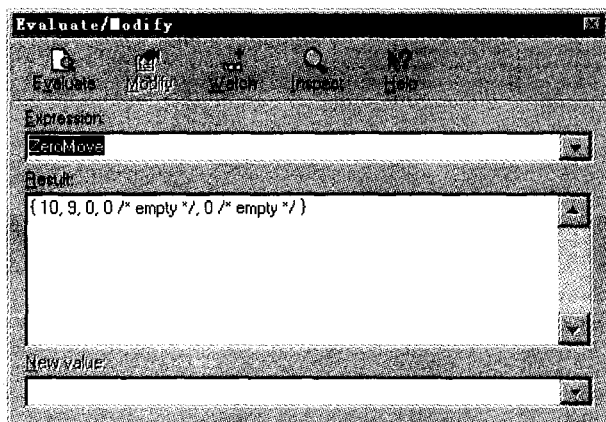


图 5.2 Evaluate/Modify 窗口

5.3.2 断点

在调试程序过程中，常常需要在程序运行到某点时暂时中断执行，这就需要设置断点来实现这一操作，它常用于判断错误出现的大致范围。

断点的设置方法如图 5.3 所示，是在 CHESS 项目中局面评估函数的循环上设置断点。将光标定位于代码编辑器中要设置断点的语句上，右击，弹出代码编辑用的快捷菜单，选择 Debug | Toggle Breakpoint 命令，或者单击要设为断点的行的最左端，此时，选中的语句变为红色并在语句的左侧有红色小圆点出现，表明已设置好断点。

断点设置好后，还可以设置它的禁止(Disabled)和允许(Enabled)状态。要清除断点标志，只需单击语句左侧的红色小圆点，或再按一次 F5 快捷键即可。

如果设置了很多断点，就有必要对这些断点进行管理(如图 5.4 所示)，选择 View | Debug Windows | Breakpoints 命令，弹出 Breakpoints List 窗口，在窗口中右击，弹出快捷命令，选择相应命令可以对断点进行编辑。选择 Enabled 命令使断点有效；选择 Delete 命令能删除断点；选择 View source 观察断点的所在行；选择 Edit Source 命令编辑断点所在行。如果希望程序满足某个条件时才中断程序的运行，选择 Properties 命令。选择 Properties 命令后，弹出 Source Breakpoint Properties 对话框，就可以为断点设置条件。断点条件有

两类，一类是条件表达式(Condition)，表达式为真时，程序将中断运行，表达式为假时，程序将继续执行；一类是计数(Pass Count)，当通过断点的次数达到计数数值时，程序将中断执行。

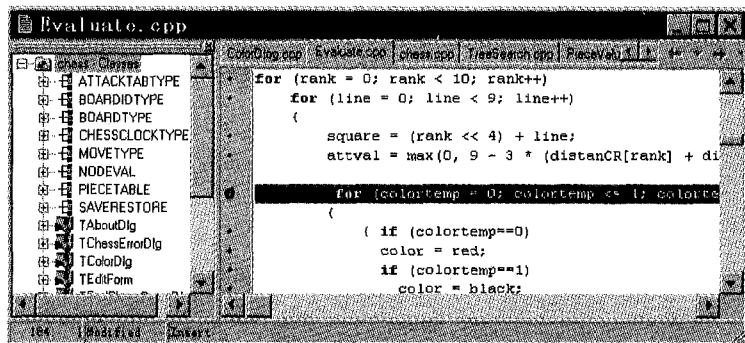


图 5.3 断点的设置

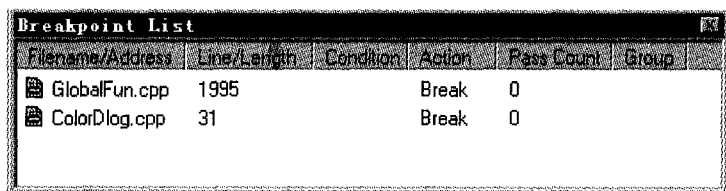


图 5.4 断点列表

5.3.3 观察变量

设置观察变量的方法是选择 Run | Add Watch 命令，或者在代码编辑器中把光标移到某个表达式上，然后按 Ctrl+F5 键，或者选择 View | Debug Windows | Watches 命令，或者右击，在快捷菜单中选择 Add Watch 命令，弹出 Watch Properties 对话框和 Watch List 窗口，前者用于输入和编辑所要添加的观察变量，后者列出所有的观察变量。Watch Properties 对话框如图 5.5 所示，在 Expression 下拉列表框中可以输入要观察的变量名。Repeat count 文本框用于查看数组，例如，要观察程序中某一数组 A[130]后 10 个数，在 Expression 下拉列表框中输入“A[121]”，在 Repeat count 文本框中输入“10”，在 Watch List 窗口中将列出这 10 个观察元素。Digits 文本框指定浮点数变量的有效位数。选中 Enabled 复选框可使观察变量有效，表示将该表达式加到观察窗口，对话框下部的选项用于设置观察变量的显示格式，选择 Hexadecimal 单选按钮使观察变量以十六进制格式显示。

在调试器中设置了许多观察变量后，就需要管理观察变量，Watch List 窗口就起到这个作用。图 5.6 是观察 CHESS 项目结构变量 ZeroMove 的窗口。在 Watch List 窗口中右击，弹出 Watch List 的快捷菜单，Disable Watch 命令用于禁止某个观察表达式，Enable Watch 命令用于激活某个观察表达式，Delete Watch 删除某个观察表达式。通过快捷菜单，可以对观察变量进行编辑。

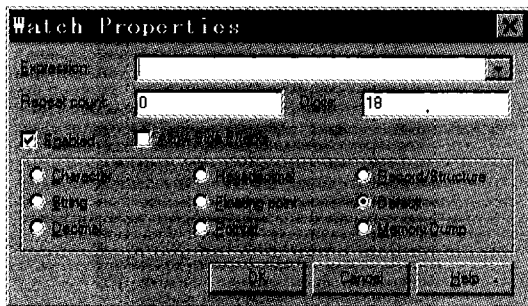


图 5.5 观察变量属性

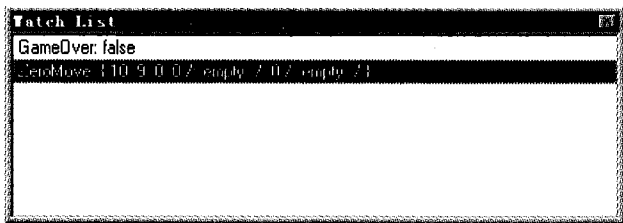


图 5.6 Watch List 窗口

5.3.4 观察类和对象

有时用户还需要观察复合型的类和对象的变化。例如，在观察变量的同时，还想观察棋子值对话框“士”编辑框的值，选择的方法是选择 Run | Inspect 命令，弹出 Inspect 对话框，如图 5.7 所示。

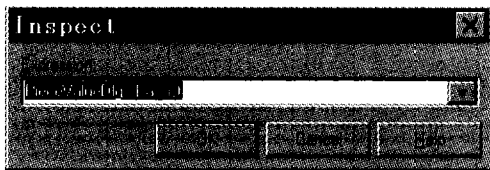


图 5.7 Inspect 对话框

在 Inspect 对话框的 Expression 下拉列表框中可以输入想要观察的类和对象，也可以输入不同数据类型的数据变量。输入一个要查看的表达式，然后单击 OK 按钮，就弹出调试观察器(Debug Inspector)，如图 5.8 所示。

调试观察器共有 3 个选项卡：Data 选项卡、Methods 选项卡和 Properties 选项卡。如果观察简单类型的数据变量，在调试观察器中将没有 Methods 选项卡和 Properties 选项卡。要观察对象和类的属性、方法和数据，只需要选择相应的选项卡，然后双击具体的属性、方法和数据的值列即可。例如，要观察某程序中的编辑组件 Eassist 的 MaxLength 的属性，打开 Data 选项卡，双击 MaxLength 属性的值列，获知 MaxLength 属性为“0”。

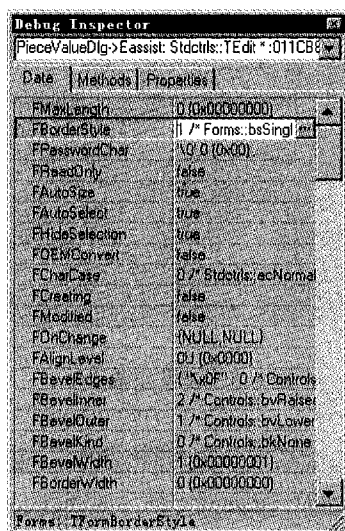


图 5.8 调试观察器

5.4 辅助调试工具

调试工具有简单与复杂之分，以下介绍的工具有一定难度，只作一般介绍。

5.4.1 CPU 窗口调式工具

CPU 窗口主要帮助程序员观察应用程序的底层运行情况，它包括 5 个相应的面板。在对程序调试的暂停状态可以打开 CPU 窗口(如图 5.9 所示)，打开的方法是选择 View | Debug Windows | CPU 命令，也可直接按 ALT+F2 键，即可以显示 CPU 窗口。

从图中左上顺时针开始，这 5 个窗格依次为：

- 反汇编窗格(Disassembly pane)，主要作用是将程序代码分解成汇编指令，包括指令地址、汇编指令及其操作数；
- 寄存器窗格(Registers pane)，包括 8 个 32 位通用寄存器、6 个 16 位寄存器和 1 个 32 位程序计数器，主要作用是显示当前 CPU 寄存器中的值，该值用十六进制表示；
- 标志窗格(Flags pane)，主要作用是显示当前 CPU 中 15 个标志位的值；
- 机器堆栈窗格(Machine Stack pane)，主要作用是显示当前程序所用堆栈中的数据，包括内存地址、地址当前的值和相应 ASCII 值；
- 内存转储窗格(Memory Dump pane)，主要作用是显示程序可利用内存空间的原始值，该窗格包括 3 个部分：内存地址、存储值和相应的 ASCII 码。

使用 CPU 窗口调式工具需要对 CPU 内部结构有一定的了解，尤其是有汇编语言的编程经验。

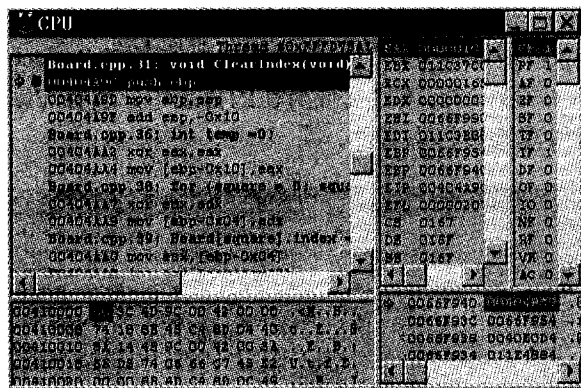


图 5.9 CPU 窗口

5.4.2 CodeGuard 调试工具

CodeGuard 是 C++Builder 中新增加的一个工具, CodeGuard 可以在运行期捕捉那些由于没有违反语法而没有被编译器发现的错误, 主要检查对内存空间和文件流不正确使用, 以及非法调用函数(例如参数不匹配, 函数返回值不匹配)产生的各种错误。CodeGuard 需要用到两个库文件, 一个是静态库文件 CG32.LIB, 一个是动态链接库文件 CG32.DLL, 这两个库文件包含了对函数描述, 包括函数参数的类型以及函数返回值类型。

要使 CodeGuard 在执行编译时发生效用, 可选择 Project | Option 命令, 在 Project | Options(如图 5.10 所示)对话框中打开 CodeGuard 选项卡, 选中 CodeGuard Validation 复选框。

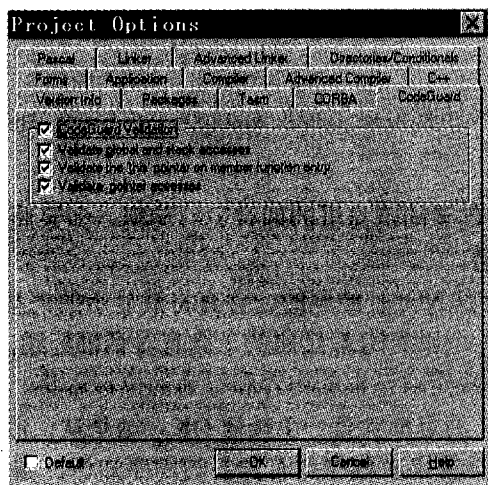


图 5.10 在 Project Options 对话框中选中 CodeGuard Validation 复选框

要使 CodeGuard 在程序运行时发生效用,可选择 Tools | CodeGuard Configuration 命令,弹出 CodeGuard Configuration 对话框,如图 5.11 所示,选中 Enable 复选框。

当 CodeGuard 产生效用后, CodeGuard 将会把它所能找到的错误记录到一个日志文件中, 文件的扩展名为.cgi, 该日志文件的文件名为项目的名称 Project.cgi。其中记载了所发现的错误。该日志文件保存在当前项目文件所在的目录中, 选择 View | Debug Windows | CodeGuard Log 命令可以观察该日志文件。

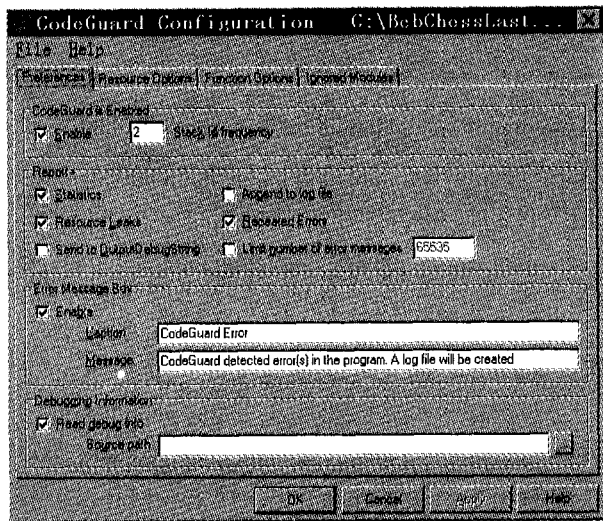


图 5.11 CodeGuard Configuration 对话框

5.5 MessageBox 的妙用

MessageBox 的功能是创建、显示、和操作一个消息框。

函数原型是:

```
int MessageBox(HWND hWnd, LPCTSTR lpText, LPCTSTR lpCaption, UINT uType);
```

参数:

- hWnd: 标识将被创建的消息框的拥有窗口句柄。如果此参数为 NULL, 则消息框没有拥有窗口句柄。
- lpText: 指向一个以 NULL 结尾的、含有将被显示的消息的字符串的指针。
- lpCaption: 指向一个以 NULL 结尾的、含有对话框标题的字符串的指针。
- uType: 指定一个决定对话框的内容和行为的位标志集。标志的含义举例如下:
MB_OK: 消息框含有一个按钮 OK。这是默认值。

使用前几节讲的方法往往要求程序员有较多的程序知识, 难度较大, 而且对于大型的 Windows 程序, 首先要缩小范围, 这时直观的 MessageBox 就可以派上用场了。一开始要找到程序的入口点, 然后加入 MessageBox 函数, 看能否正常运行, 如果没有问题, 撤消 MessageBox 函数, 再往下一句加入 MessageBox 函数, 这样一步一步地跟踪, 当 MessageBox

函数出现异常时,再结合其他手段就可以找出症结之所在。必要时,可以把一些参数传给 `MessageBox` 函数显示出来,就更直观了。

作者以前使用 `C++Builder 4.0` 调试程序时,曾发现采用枚举变量作为循环条件的程序一启动后就不再往下执行,仔细检查也没有发现可疑的问题,因为同样的等价程序在使用 `C++ OWL` 和 `Visual C++` 都没有出现过问题。于是采用 `MessageBox` 来帮忙,终于解决了问题。

此次使用 `C++Builder 5.0` 调试象棋程序时遇到时间老是出现负数的问题,走几步后程序就不能正常工作,于是作者不得不使用 `MessageBox` 一步一步地检查 `CHESS` 项目,发现原来是读取系统时间的函数 `GetLocalTime()`或 `GetSystemTime()`或 `Now().DecodeTime()`中的毫秒项出现错误。

5.6 巧用小项目文件

作者在本次的象棋程序中发现时间老是出现负数,采用 `MessageBox` 跟踪到错误后,发现读取系统时间的函数中的毫秒项出现错误。考虑到有时一个问题是由几种因素引起的,必须把问题孤立起来,才能得出确切的结论。于是,利用已有的函数建立了 `TestTime` 项目文件,结果证实了是函数中毫秒项有问题。方法如下:

- (1) 仿照建立 `CHESS` 项目的方法,选择 `File | New Application` 命令,程序将自动出现 `Form1`;
- (2) 在对象观察器的 `Properties` 选项卡中改变属性 `Caption` 为 `TimeForm`;
- (3) 单击 `System` 组件选项卡中的 `Timer` 定时器组件,再在 `Form1` 中单击;
- (4) 单击 `Additional` 组件选项卡中的 `StaticText` 静态文本框(显示时间字符串)组件,再在 `Form1` 中单击。结果如图 5.12 所示。

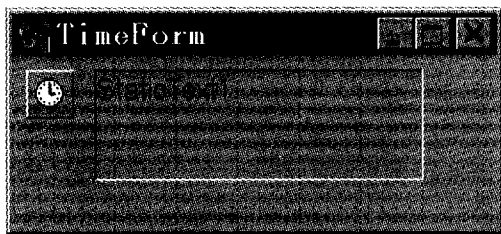


图 5.12 Form1 的外观

- (5) 双击 `Form1` 中的定时器,出现 `Unit1.cpp` 代码编辑器;
- (6) 在 `TForm1::Timer1Timer` 函数中手工加入 `DisplayTime()` 函数以显示时间;
- (7) 选择 `File | New` 命令,打开 `New Items` 对话框,在其中选中 `Cpp File` 程序,将自动出现 `File1.cpp` 代码编辑窗口,由用户输入必需的代码具体定义 `DisplayTime()` 函数(见第 9 步);

- (8) 选择 File | Save Project As...命令保存项目文件, 程序将自动弹出 Save Unit1 As 对话框, 在【文件名】文本框中输入文件名“TimeForm.cpp”保存单元文件; 接着弹出 Save File1 As 对话框, 在【文件名】文本框中输入文件名“Time.cpp”保存 C++自定义文件; 最后弹出 Save Project1 As 对话框, 在【文件名】文本框中输入文件名“TestTime.bpr”保存项目文件。结果如图 5.13 所示。

- (9) 具体定义主窗体中定时器回调函数, 其定义放在单元文件 Time.cpp 中, 函数说明如下:

- 读取系统时间函数, 毫秒保存在 systime.wMilliseconds 中:

```
void GetTime(int *hour, int *minute, int
*second, int * sec100)
{
    SYSTEMTIME systime;
    GetLocalTime(&systime);
    *hour = systime.wHour;
    *minute = systime.wMinute;
    *second = systime.wSecond;
    *sec100 = systime.wMilliseconds * 10;
}
```

- 显示系统时间函数:

```
void DisplayTime()
{
    int hour1, min1, sec1, s100;
    static bool Colon = true;
    double CurTime = 0.0 ;
    char buf[80];
    GetTime(&hour1, &min1, &sec1, &s100);
    CurTime += ((s100 ) * 0.01+
//s100 乘以 0.01 本应该是秒, 可运行时 1 秒变为 5 秒。
//为了避免出现时间错误, 作者只好乘以零, 干脆忽略该项。
    (sec1 ) +
    (min1 ) * 60.0 +
    (hour1) * 3600.0);
    min1 = int(CurTime / 60.0);
```

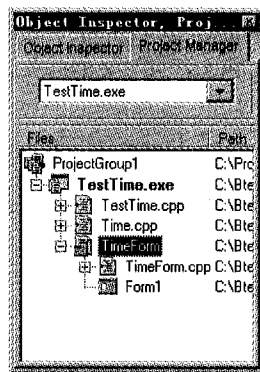


图 5.13 TestTime 项目

```
sprintf(buf, "%2.2d%s%02d", min1, (Colon) ? ":" : " ", int(CurTime -
min1 * 60.0));
```

```
Form1->StaticText1->Caption = buf;
```

```
Colon = !Colon;
```

```
}
```

(10) 编译运行程序(如图 5.14 所示), 本应该是每秒数字加 1, 实际上却是每秒加 5。

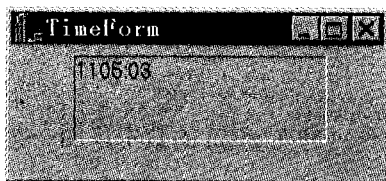


图 5.14 TestTime 程序的运行状态

提示: 把 DisplayTime()函数中的 s100 项乘以 0 忽略该项后, 数字时间显示正常。

以上例子的完整程序源代码在所附光盘中目录 BCBTEST 内。

由于作者水平有限, 不敢判定是否是 GetLocalTime()或 GetSystemTime()或 Now().DecodeTime()函数出现了错误, 但可以肯定的是 GetLocalTime()或 GetSystemTime()或 Now().DecodeTime()函数的毫秒项有问题, 应用时请注意。

5.7 CHESS 项目综合调试实例

对于 CHESS 项目这样较大型的程序出现的错误, 要采取化整为零的方法, 各个击破, 即先分别测试, 再集成测试, 确保问题小而单纯。

5.7.1 综合各种小技巧调试

这次在调试程序时, 刚开始运行 CHESS 项目时棋盘上只有 4 个棋子。于是作者重新考虑底层的每个自定义函数的可靠性, 经过全面考虑, 问题集中在下面语句。

```
if (!(square & 0xa9))
```

本来是希望通过这一语句把在棋盘外的位置排除在计算之外, 但实际情况试过才知道。具体步骤如下:

- (1) 选择 File | New 命令打开 New Items 对话框, 在其中选中 Console Wizard, 弹出 Console Wizard 对话框;
- (2) 选中 C++单选按钮和 Console Application 复选框后, 单击 OK 按钮;

出现如下代码:

```
#pragma hdrstop
```

```
//-----
```

```
#pragma argsused
```

```
int main(int argc, char* argv[])
```

```
{
```

```
    return 0;
```

```
}
```

在上述代码中加入包含文件,并在 main 函数中添加一些代码,变成如下代码:

```
#include <conio.h>
```

```
#include <stdio.h>
```

```
#pragma hdrstop
```

```
//-----
```

```
int square,i=0;
```

```
#pragma argsused
```

```
int main(int argc, char* argv[])
```

```
{
```

```
for (square = 0; square < 0x99; square++)
```

```
    if (!(square & 0xa9))
```

```
    {
```

```
        i= square;
```

```
    }
```

```
    return 0;
```

```
}
```

由于这个程序相当简单,估计 square 取值也是很少的,于是采取如下步骤:

- (1) 单步执行,按 F8 键;
- (2) 程序执行进入循环体中时,选择 Run | Evaluate/Modify 命令,弹出 Evaluate/Modify 窗口;
- (3) 在 Expression 下拉列表框中输入“I”,单击 Evaluate 按钮,在 Result 列表框中出现一个值,或单击 Watch 按钮出现 Watch List 窗口;
- (4) 记下每个 i 的十六进制取值。

以下是记下的 i 十六进制取值:

0X0, 0X2, 0X4, 0X6, 0X10, 0X12, 0X14, 0X16, 0X40, 0X42, 0X44, 0X46, 0X50, 0X52, 0X54, 0X56。

很明显, i 的取值没有 90 个,因此,问题就找到了。

(5) 修改 main()函数成如下代码:

```
int main(int argc, char* argv[])
{
    for (square = 0; square < 0x99; square++)
        if (((square>>4)<10) && ((square % 16)<9) && ((square | 0xff)>=0))
        {
            i= square;
        }
    return 0;
}
```

(6) 编译、运行程序。

结果 i 取 0X0~0X98 共 90 个值, 说明只对整个棋盘内(10×9)的 90 个棋子位置进行逐个计算, 而对棋盘外面的位置一概忽略。

5.7.2 采用输出文件的方式调试

解决了上述问题, 再次运行 CHESS 项目却发现棋盘上虽然一个棋子不少, 却没法移动棋子, 可见问题还没完全解决。于是又得采取对一切质疑的态度, 重新考虑每个自定义函数的可靠性, 经过全面考虑, 问题集中在 CalcPieceTable()计算棋子表函数上, 因为这个函数中用到 square ^=0x98 这个语句, 当初写它时候, 只记得书本上曾讲过这种用法, 其实对其含义并不十分了解。因为 CalcPieceTable()函数是对棋盘中的每个棋子进行扫描计算, 其关键是看变量 square 是否取足 90 个棋格位置值, 于是取出这一段函数制作以下 DOS 程序。步骤如下:

- (1) 选择 File | New 命令打开 New Items 对话框, 在其中选中 Console Wizard 选项打开 Console Wizard 对话框;
- (2) 选中 C++ 单选按钮和 Console Application 复选框后, 单击 OK 按钮。

出现如下代码:

```
#pragma hdrstop

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    return 0;
}.
```

在上述代码中加入包含文件, 并在 main 函数中添加一些代码, 变成如下代码:

```
#include <conio.h>
#include <stdio.h>

#pragma hdrstop

//-----

FILE *out;

#pragma argsused
int main(int argc, char* argv[])
{
    int square;

    out = fopen("h.dat", "wt");

    square = 0;
    do
    {
        square ^=0x98;
        if (!(square & 4))
        {
            if (square >= 0x70)
                square = (square + 0x11) & 0x73;
            else
                square += 0x10;
        }
        fprintf(out,"square %x\n",square);
    } while (square);

    fclose(out);
    getch();
    return 0;
}
```

由于这个程序稍微复杂一点，对其结果不能预见，担心输出的结果太多，故采取把结果输出到 h.dat 文件中的方法。运行程序，从 h.dat 文件中知 square 只取 0X0, 0X21, 0X42, 0X63 4 个值，说明原来打算对整个棋盘(10×9)90 个棋子位置进行逐个计算已不可能。找到了问题的症结，改起来就很容易了，把 main()函数改为如下代码：

```
int main(int argc, char* argv[])
{
    int square,i;

    out = fopen("h.dat", "wt");

    square = 0;
    do
    {
        square ^=0x77;
        if (!(square & 4))
        {
            if (square >= 0x70)
                square = (square + 0x11) & 0x73;
            else
                square += 0x10;
        }
        fprintf(out,"square %x\n",square);
    } while (square);
    for (i=0;i<=9;i++)
    {
        square=8+0x10*i;
        fprintf(out,"square %x\n",square);
    }

    for ( i=0;i<=7;i++)
    {
        square=0x80+i;
        fprintf(out,"square %x\n",square);
    }

    for ( i=0;i<=7;i++)
    {
        square=0x90+i;
        fprintf(out,"square %x\n",square);
    }
}
```

```
    fclose(out);  
    getch();  
    return 0;  
}
```

编译、运行程序，结果 square 取 0X0~0X98 共 90 个值，说明打算对整个棋盘(10×9)90 个棋子位置进行逐个计算正好满足需要。

第 6 章 美化 CHESS 项目——图形与图像

图形和图像能使应用程序增色不少，高质量的图形和图像已经成为许多应用程序的一个重要组成部分。在 C++Builder 中为了增加图形和图像，可以在编程时插入一些事先准备好的图片、也可以在设计时用图形和图像组件创建图片或者在程序运行时动态地绘制它。C++Builder 提供了大量的组件、类和方法，为进行图形应用程序编程提供了简洁而有效的方法。但要进行高性能的程序设计，必须要理解 GDI，同时更深入地了解 VCL 图形组件。

6.1 Windows 图形编程知识

Win32 应用程序接口(API)指的是 Windows 的图形输出函数，其主要特点之一是设备无关性，GDI 为应用程序生成的发往显示器、打印机和其他设备的图形输出提供一系列函数和相关的数据结构，为其编写的程序可以在多种设备上输出而不必考虑设备的型号、厂家等具体细节。应用程序通过创建设备描述表与图形设备接口(GDI)交互。

6.1.1 设备描述表

设备描述表的属性描述了所选定的绘图目标(画笔和刷子)、所选定的字库和它的颜色、目标在设备上的绘制(或映射)的方式、设备上可使用的输出区域(剪接区域)和其他重要信息，包含这些属性的数据结构称为设备描述表(DC)。设备描述表(如表 6.1 所示)的建立使用了大多数参数的默认值，代替了在每次调用 GDI 函数时传送大量的参数。

表 6.1 设备描述表属性、默认值和相关的 GDI 函数

属 性	默 认 值	GDI 函数
背景颜色	白色	SetBkColor
背景模式	OPAQUE	SetBkMode
位图	无默认值	CreateBitmap、CreateBitmapIndirect、CreateCompatibleBitmap、SelectObject SelectObject
画刷	WHITE_BRUSH	CreateBrushIndirect、CreateDIBPatternBrush、CreateHatchBrush、CreatePatternBrush、CreateSolidBrush、Select Object
画刷原点	(0, 0)	SetBrushOrg UnrealizeObject
剪接区域	显示器表面	ExcludeClipRect、IntersectClipRect、OffsectClpiRgn SelectClipRgn
调色板	DEFAULT_PALETTE	CreatePalette、RealizePalette、SelectPalette

续表

属 性	默 认 值	GDI 函数
当前画笔位置	(0, 0)	MoveToEx
绘图模式	R2_COPYPEN	SetROP2
字库	SYSTEM_FONT	CreateFont、CreateFontIndirect、SelectObject
字符间距	0	SetTextCharacterExtra
映射模式	MM_TEXT	SetMapMode
画笔	BLACK_PEN	CreatePen、CreatePenIndirect、SelectObject
多边形填充模式	ALTERNATE	SetPoly FillMode
变倍模式	BLACKONWHITE	SetStretchBltMode
文字颜色	黑色	SetTextColor
观察点步长	(1, 1)	SetViewportExt
观察点原点	(0, 0)	SetViewportOrg
窗口步长	(1, 1)	SetWindowExt
窗口原点	(0, 0)	SetWindowOrg

6.1.2 图形设备接口

Windows 使用句柄确定项, 每个设备描述表用一个设备描述句柄(HDC)来确定。这些对象都可通过相应类型的句柄(HPEN、HBRUSH、HBITMAP、HFONT、HPALETTE 等)或普通的 GDI 对象句柄(HGDIOBJ)进行引用。更新默认值的过程如下:

- 选择一个新的对象进入设备描述表并保存旧对象的句柄。
- 使用新对象进行必要的处理。
- 使用保存的句柄将旧的对象选进设备描述表。

一个设备描述表在再次拥有其所有的原始对象之前不能被释放或撤消。程序通过从 Windows 请求库存的对象, 或显式地创建对象本身来获取新的 GDI 对象。对后一种情况, 每个对象在被选进设备描述表之前创建, 并且只有在实际被另一个对象取代后才能撤消, 操作系统提供的库存对象不需要程序撤消。除了调色板, 每个 GDI 对象都使用 SelectObject()函数选进或选出设备描述表, 而函数的返回值为替代的对象提供一个句柄。使用后, 要释放设备描述表。如果没有释放设备描述表, 程序将会遇到内存泻漏、资源不足的问题。最严重时, 将导致系统崩溃。

在 C++Builder 中封装了大部分 GDI 对象, 从而更加易于使用 GDI 函数编程。如 TFont 封装了字体对象、TPen 封装了画笔对象、TBrush 封装了画刷对象等。在 C++Builder 中封装的 TFont、TPen、TBrush 等对象分别提供了一个 Handle 属性, 对这些对象不能实现的功能, 可以通过 Windows GDI 函数访问 Handle 属性来实现。C++Builder 对区域、调色板、路径等 GDI 对象没有封装, 对这些对象的操作必须使用 GDI 函数。

6.2 图形功能

在 C++Builder 中, 绘图操作经常是在对象的画布(Canvas)上, 而不是直接在对象上绘制。TCanvas 类变量常用作窗体或其他组件的属性, 而不单独使用。它是 C++Builder 中几乎所有图形组件都有的一个属性, 同时它本身也是对象。画布属性提供了一个用户作图区域, 可以实现各种绘图功能。TCanvas 类是整个图形和图像绘制的基础, 它封装了应用程序在图形设计所需要的大多数 GDI 对象和绘图函数。虽然 C++Builder 用 TCanvas 类封装了 Windows 的设备描述表, 用户仍可使用 GDI 函数直接在 Canvas 上绘图。TCanvas 的属性 Handle 就是设备描述表的句柄。

在 C++Builder 5.0 中 TCanvas 类的内容比以前版本的内容更丰富, 其主要变化是增加了新的绘图成员函数以及在多线程应用程序中线程对画布控制的相关成员函数。

下面是 TCanvas 类的定义。

```
class PASCALIMPLEMENTATION TCanvas : public Classes::TPersistent
{
    typedef Classes::TPersistent inherited;

private: //私有定义
    HDC FHandle;
    TCanvasState State;
    TFont* FFont;
    TPen* FPen;
    TBrush* FBrush;
    Windows::TPoint FPenPos;
    int FCopyMode;
    Classes::TNotifyEvent FOnChange;
    Classes::TNotifyEvent FOnChanging;
    _RTL_CRITICAL_SECTION FLock;
    int FLockCount;
    int FTextFlags;
    void __fastcall CreateBrush(void);
    void __fastcall CreateFont(void);
    void __fastcall CreatePen(void);
    void __fastcall BrushChanged(System::TObject* ABrush);
    void __fastcall DeselectHandles(void);
    TCanvasOrientation __fastcall GetCanvasOrientation(void);
    Windows::TRect __fastcall GetClipRect();
    HDC __fastcall GetHandle(void);
```

```

Windows::TPoint __fastcall GetPenPos();
TColor __fastcall GetPixel(int X, int Y);
void __fastcall FontChanged(System::TObject* AFont);
void __fastcall PenChanged(System::TObject* APen);
void __fastcall SetBrush(TBrush* Value);
void __fastcall SetFont(TFont* Value);
void __fastcall SetHandle(HDC Value);
void __fastcall SetPen(TPen* Value);
void __fastcall SetPenPos(const Windows::TPoint &Value);
void __fastcall SetPixel(int X, int Y, TColor Value);

protected: //受保护的函数
    virtual void __fastcall Changed(void);
    virtual void __fastcall Changing(void);
    virtual void __fastcall CreateHandle(void);
    void __fastcall RequiredState(TCanvasState ReqState);

public: //公有定义, 这部分最重要。
    __fastcall TCanvas(void);
    __fastcall virtual ~TCanvas(void);
/*以下是 TCanvas 的方法*/
    void __fastcall Arc(int X1, int Y1, int X2, int Y2, int X3, int Y3, int
X4, int Y4); //画圆弧
    void __fastcall BrushCopy(const Windows::TRect &Dest, TBitmap* Bitmap,
const Windows::TRect &Source, TColor Color);
    void __fastcall Chord(int X1, int Y1, int X2, int Y2, int X3, int Y3,
int X4, int Y4); /*画圆弧, 并用一直线连接圆弧的端点*/
    void __fastcall CopyRect(const Windows::TRect &Dest, TCanvas* Canvas,
const Windows::TRect &Source); //复制另一个 TCanvas 对象的部分图像
    void __fastcall Draw(int X, int Y, TGraphic* Graphic); //在画布上输出
//图像
    void __fastcall DrawFocusRect(const Windows::TRect &Rect);
    void __fastcall Ellipse(int X1, int Y1, int X2, int Y2)/* overload */;
//画椭圆
    void __fastcall Ellipse(const Windows::TRect &Rect)/* overload */;
    void __fastcall FillRect(const Windows::TRect &Rect); //填充矩形
    void __fastcall FloodFill(int X, int Y, TColor Color, TFillStyle
FillStyle); /*填充一个封闭矩形*/
    void __fastcall FrameRect(const Windows::TRect &Rect); //绘制矩形边框

```



```

void __fastcall LineTo(int X, int Y); //画直线
void __fastcall Lock(void);
void __fastcall MoveTo(int X, int Y); //移动画笔位置
void __fastcall Pie(int X1, int Y1, int X2, int Y2, int X3, int Y3, int
X4, int Y4); /*画饼状区域*/
void __fastcall Polygon(const Windows::TPoint * Points, const int
Points_Size); /*画多边形*/
void __fastcall Polyline(const Windows::TPoint * Points, const int
Points_Size); /*连接画布上的一系列点*/
void __fastcall PolyBezier(const Windows::TPoint * Points, const int
Points_Size);
void __fastcall PolyBezierTo(const Windows::TPoint * Points, const int
Points_Size);
void __fastcall Rectangle(int X1, int Y1, int X2, int Y2)/* overload */;
//画矩形
void __fastcall Rectangle(const Windows::TRect &Rect)/* overload */;
void __fastcall Refresh(void);
void __fastcall RoundRect(int X1, int Y1, int X2, int Y2, int X3, int
Y3); //画圆角矩形
void __fastcall StretchDraw(const Windows::TRect &Rect, TGraphic*
Graphic); /*使图像适应画布大小*/
tagSIZE __fastcall TextExtent(const AnsiString Text);
int __fastcall TextHeight(const AnsiString Text); //确定字符串的高度
void __fastcall TextOut(int X, int Y, const AnsiString Text); //在画
布上输出字符串
void __fastcall TextRect(const Windows::TRect &Rect, int X, int Y, const
AnsiString Text); /*在限定的矩形区域输出字符串*/
int __fastcall TextWidth(const AnsiString Text); //确定字符串的宽度

bool __fastcall TryLock(void);
void __fastcall Unlock(void);
__property Windows::TRect ClipRect = {read=GetClipRect}; /*定义绘画区
域*/
__property HDC Handle = {read=GetHandle, write=SetHandle, nodefautl};
/*设备描述表句柄*/
__property int LockCount = {read=FLockCount, nodefautl};
__property TCanvasOrientation CanvasOrientation =
{read=GetCanvasOrientation, nodefautl};

```

```

__property Windows::TPoint PenPos = {read=GetPenPos, write=SetPenPos};
/*画笔的当前位置*/
__property TColor Pixels[int X][int Y] = {read=GetPixel, write=SetPixel};
/*可以读取剪裁矩形中指定位置的像素的颜色, 也可以修改该像素的颜色*/
__property int TextFlags = {read=FTextFlags, write=FTextFlags, node
fault};
__property Classes::TNotifyEvent OnChange = {read=FOnChange, write=Fon
Change};
__property Classes::TNotifyEvent OnChanging = {read=FOnChanging, write=
FOnChanging};

__published:
__property TBrush* Brush = {read=FBrush, write=SetBrush}; //封装了画刷
//对象
__property int CopyMode = {read=FCopyMode, write=FCopyMode,
default=13369376}; /*画笔的复制特性*/
__property TFont* Font = {read=FFont, write=SetFont}; //封装了字体对象
__property TPen* Pen = {read=FPen, write=SetPen}; //封装了画笔对象
};

```

6.2.1 画布(TCanvas)的属性

下面介绍画布对象的几种常用属性。

1. TFont 对象

TFont 对象描述了画布绘制文本时所使用字体的属性, 它包含有如下几个重要的属性。

- **Name** 属性: 定义了字体的名称, 如 Arial、Times New Roman 或 Westminster。可以使用系统中任何已经安装的字体, 也可以创建自定义的字体。
- **Height** 属性: 确定字体的高度(以像素为单位), 不包括字符顶部的间距。
- **Size** 属性: 确定字体的大小, 与 Height 有关。字体的大小是磅值。当 Size 特性为一个磅值, Height 的特性就变为负数。如果 Height 的特性为一个正数, 字体的 Size 特性就变为负数。Size 与 Height 属性的关系如下:

```
Font->Size = - Font->Height * 72 / Font->PixelsPerInch;
```

- **Style** 属性: 字体的样式。它是枚举类型集合, 可以使多种样式同时有效。这些字体样式包括: fsBold(黑体); fsItalic(斜体); fsUnderline(下划线); fsStrikeOut(删除线)。
- **Color** 属性: 确定字体的颜色。TFont 类的 Color 属性是 TColor 类型。

很多类和部件都具有颜色属性,即 TColor 类型。TColor 在文件 graphics.hpp 中定义如下:

```
enum TColor{clMin=-0x7FFFFFFF-1,clMax=0x7FFFFFFF};
```

● 使用颜色常量定义

在 TColor 的枚举中,这些常量是 32 位二进制常量。在 graphics.hpp 中也定义了 TColor 的一些常用常量。这些常量可以直接映射成与系统调色板中最相近的颜色(如颜色常量 clRed 映射成为红色),也可以映射成在 Windows 控制面板中颜色部分定义的相应系统视频颜色(如颜色常量 ClHighlightText 映射成为 Windows 相应的高亮度文本颜色)。

直接映射成与系统调色板中最相近颜色的颜色常量如表 6.2 所示。

表 6.2 与调色板最相近颜色常量

颜色取值	含 义	颜色取值	含 义
clAqua	浅绿色	clBlue	蓝色
ClBlack	黑色	clDkGray	深灰色
ClFuchsia	紫红色	clGray	灰色
ClGreen	绿色	clLime	酸橙色
ClLtGray	浅灰色	clMaroon	栗色
clNavy	深蓝色	clOlive	橄榄色
ClPurple	紫色	clRed	红色
ClSilver	银色	clTeal	水鸭色
ClWhite	白色	clYellow	黄色

映射成 Windows 控制面板中相应系统视频颜色的颜色常量如表 6.3 所示。

表 6.3 系统视频颜色常量

颜色取值	含 义	颜色取值	含 义
ClActiveCaption	活动窗口标题栏的颜色	clBackground	Windows 桌面的背景颜色
ClInactiveCaption	非活动窗口标题栏的颜色	clMenu	菜单的背景颜色
ClWindow	窗口的背景颜色	clWindowFrame	窗口结构的颜色
ClMenuText	菜单文本颜色	clWindowText	窗口内文本颜色
ClCaptionText	活动窗口标题栏文本颜色	clActiveBorder	活动窗口边界的颜色
ClInactiveBorder	非活动窗口边界的颜色	clAppWorkSpace	应用程序工作间的颜色
ClHighlight	被选定文本的背景颜色	clHightlightText	被选定文本的颜色
ClBtnFace	按钮外观颜色	clBtnShadow	按钮按下变暗的颜色
ClGrayText	变暗文本的颜色	clBtnText	按钮文本颜色
ClInactiveCaptionText	非活动窗口标题栏的文本颜色	clBtnHighlight	按钮高亮度时的颜色

续表

颜色取值	含义	颜色取值	含义
cl3DdkShadow	三维显示元件的黑色阴影	cl3Dlight	三维显示元件的明亮颜色
ClInfoText	工具条的文本颜色	clInfoBk	工具条的背景颜色

● 二进制码定义

如果用 4 字节十六进制代码而不是用 Graphics.hpp 中定义的 TColor 颜色常量来定义颜色, 则其后 3 个字节代表 RGB 中蓝色、绿色、红色 3 种颜色各自的强度。00FF0000 表示纯蓝, 0000FF00 表示纯绿, 000000FF 表示纯红, 00000000 表示黑色, 00FFFFFF 表示纯白色。

如果最高位字节是 00, 则获取的颜色是与系统调色板中最相近的颜色相匹配。如果最高位字节是 01, 则获取的颜色是与当前使用的调色板中最相近的颜色相匹配。如果最高位字节是 02, 则获取的颜色是与当前设备描述表里逻辑调色板中最相近的颜色相匹配。

可采用颜色常量来描述字体的颜色, 如 ClWhite(白色)、ClBlack(黑色)和 ClRed(红色)等。

有时候需要使用自定义的颜色, 如(TColor)0x00c700 把一个十六进制值类型转换为 TColor。

2. TBrush 对象

TBrush 对象封装了标准 Windows 画刷对象, 描述了画布在填充图形和背景时使用的颜色和图案。使用画刷的 Color 属性可以设置画刷的颜色, 默认的画刷颜色是 clWhite。使用画刷的 Style 属性可以确定填充图案的式样。填充图案的式样一共有 8 种, 如表 6.4 所示。

表 6.4 画刷的 Style 属性值含义

属性值	说明
bsSolid	实心填充
bsClear	不填充, 透明
bsBDiagonal	左斜线填充
bsFDiagonal	右斜线填充
bsCross	十字线填充
BsDiagCross	交叉线填充
bsHorizontal	水平线填充
bsVertical	垂直线填充

3. TPen 对象

TPen 对象描述了画布的笔(Pen)的属性, 它用于控制画布上所画线或形状的外轮廓线。它的属性有: 颜色(Color), 宽度(Width), 模式(Mode)以及式样(Style)。

- Color 属性: 控制画笔的颜色。

- **Width** 属性：确定线的宽度(以像素为单位)。默认宽度为 1，如果宽度小于 1，将按 1 处理。
- **Mode** 属性：确定画笔的颜色如何与画布颜色相互作用，对应于 Windows 中光栅操作。在表 6.5 中列举了 Mode 的取值及其作用效果。

表 6.5 Mode 的取值及其作用效果

Mode 的值	作用效果
pmBlack	黑色
PmCopy	由画笔颜色确定
PmMask	画笔颜色和画布背景色的取并的颜色
PmMaskNotPen	画笔颜色取反和画布背景色的共同色
PmMaskPenNot	画布背景色取反和画笔颜色的共同色
PmMerge	画笔颜色和画布背景色不相同部分的组合
PmMergeNotPen	画笔颜色取反和画布背景色的组合
PmMergePenNot	画布背景色取反和画笔颜色的组合
PmNop	不改变画布颜色
PmNot	画布背景色取反
PmNotCopy	画笔颜色取反
PmNotMask	pmMask 的取反
PmNotMerge	pmMerge 的取反
PmNotXor	pmXor 的取反
PmWhite	白色
PmXor	画笔颜色与画布背景色取或的组合

- **Style** 属性：确定画笔线的式样。画笔线的式样如表 6.6 所示。

表 6.6 Style 的取值及其意义

Style 的值	说 明
PsClear	不画线(用于删除当前画笔所画图形上的线)
PsDash	由下划线组成的线段
PsDashDot	点画线
ps DashDotDot	双点划线
PsDot	点线
PsInsideFrame	实线(当宽度大于 1 时可能使用抖动色)
PsSolid	实线

4. 画布对象的其他属性

画布对象除了上面介绍的几个属性外，还有 PenPos、Pixels 和 CopyMode 等。

- PenPos 属性

PenPos 描述画笔的当前位置，它的类型为 TPoint。它不是一个只读属性，可以读取或设置画笔的当前位置。读取画笔当前的位置所在的点，是 LineTo 方法所画线的起点。设置画笔的当前位置，相当于调用 MoveTo 方法。

- Pixels 属性与 ClipRect 属性

Pixels 用于确定当前像素点的颜色，可以读取或设置某一像素点的颜色。

ClipRect 属性：确定画布上一个矩形绘图区域。在 ClipRect 区域之外的图形将被剪切，且不出现在画布上。

- CopyMode 属性

确定画布上图像的相互作用方式，如表 6.7 所示。它可用于用 CopyRect 方法从另外一个画布上复制图像时，也可以用于 TBitmap 对象在画布上画它们自己。使用 CopyMode 处理图像时能影响图像的绘制效果。例如合并图像，与多个图像组合时使位图的各个部分产生透明效果。默认模式是 cmSrcCopy。

表 6.7 CopyMode 属性的取值及拷贝效果：

值	效 果
cmBlackness	黑色
cmDstInvert	反转目的位图
cmMergeCopy	源位图与目的画笔样式进行与操作
cmMergePaint	反转的源位图与目的位图进行或操作
cmNotSrcCopy	复制反转的源位图到目的位图中
cmNotSrcErase	将或操作后的目的位图和源位图进行反转
cmPatCopy	将目的画笔样式与目的位图进行异或操作
cmPatInvert	将目的位图与目的画笔样式进行异或操作
cmPatPaint	先将反转的源位图与目的画笔样式进行或处理，再将该操作的结果与目的位图进行或操作
cmSrcAnd	目的位图与源位图进行与操作
cmSrcCopy	将源位图复制到目的位图中
cmSrcErase	反转目的位图并将结果与源位图进行或操作
cmSrcInvert	将目的位图与源位图进行异或操作
cmSrcPaint	将目的位图与源位图进行或操作
cmWhiteness	白色

6.2.2 画布对象的常用方法

画布对象 TCanvas 是 C++Builder 提供的类的一个标准属性，如同 TForm、TGraphicControl 以及它们的子类一样，提供了一个可以在其上面绘制、着色对象或操纵的表面。由于是其内置对象，仅作为一个特性而存在，因此不能单独存在，要依附于某个组件，如定义一个窗体，那么这个窗体就含有 TCanvas 对象，同时自然就能使用 TCanvas 内置方法。它的方法较多，主要包括画线、画圆、椭圆、矩形和圆角矩形等基本图形的绘制

方法,画填充图形的方法以及文字在图形中的输出方法等。表 6.8 包括了画布对象的常用方法。

表 6.8 画布对象的常用方法

方 法	说 明
Arc	使用当前画笔在画布上画弧
BrushCopy	在透明背景下显示位图
Chord	在椭圆上画弦
CopyRect	复制一个剪取矩形图像
Draw	在画布上复制内存中的图形
Ellipse	画椭圆,并用当前画刷填充
FillRect	用当前画笔填充矩形
FloodFill	用当前画笔填充画布任意封闭区域
FrameRect	用画刷画笔矩形的边(一个像素宽度)
LineTo	从 PenPos 到(X, Y)画直线
MoveTo	设置当前画图位置
Pie	画饼形图
PolyBezier	画 Bezier 曲线
Polygon	画多边形。用画刷值填充图形
PolyLine	画折线段,不自动闭合
Rectangle	画以点(X1, Y1)和(X2, Y2)为对角顶点的矩形,并用当前画刷填充
Refresh	取消选定的画笔、画刷和字体,但不改变它们的属性值
RoundRect	画圆角矩形
StretchDraw	画自动拉伸的图形
TextExtent	返回 AnsiString 类字符串的高度和宽度
TextHeight	返回 AnsiString 类字符串的高度
TextOut	从指定位置开始输出字符串,并将 PenPos 更新到字符串尾
TextRect	在一个矩形框内输出字符串,超出矩形框的字符串不显示
TextWidth	返回 AnsiString 类字符串的宽度

使用 TCanvas 画布作图非常简单,如画一个矩形,可以编写如下代码:

```
Canvas->Rectangle(10,10,10,10);
```

在一个名为 Form2 的窗体上画一个矩形,可以编写如下代码:

```
Form2->Canvas->Rectangle(10,10,10,10);
```

书写一行文字,可以编写如下代码:

```
Form2->Canvas->TextOut(10,12,"Hello!");
```

6.2.3 实现 CHESS 项目画布

下面举例说明使用位图填充窗体背景。使用 LoadFromFile 函数装入如图 6.1 所示的位图文件，使用 FillRect 函数进行填充形成如图 6.2 所示的背景。使用 VCL 函数比使用原始的 API 函数要方便得多。如果用户对此位图不满意的话，可以换成自己喜爱的位图。

```
Graphics::TBitmap *BLKBrushBmp = new Graphics::TBitmap();  
BLKBrushBmp->LoadFromFile("c:\\BcbChess\\BKBrush.bmp");  
EditForm->Canvas->Brush->Bitmap=BLKBrushBmp;  
EditForm->Canvas->FillRect(EditForm->GetClientRect());
```

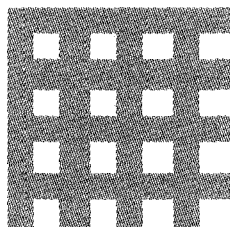


图 6.1 填充背景的位图(8X8 像素)

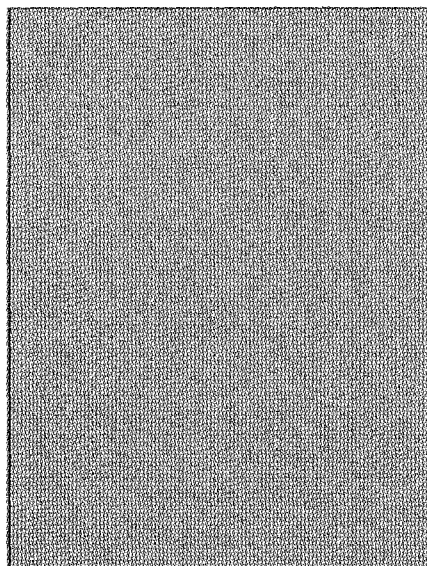


图 6.2 利用 Canvas->Brush->Bitmap 设置窗体背景

6.3 图像功能

C++Builder 中提供了一系列处理图像的对象。利用这些对象，不需要考虑复杂的图像文件格式，就能对图像进行各种处理。常用的图像类和组件包括 TGraphic 类、TPicture 类、TImage 组件、TShape 组件和 TPaintBox 组件等。本节介绍 C++Builder 的图像功能，对图像的有关属性和方法进行了介绍。

6.3.1 TGraphic 类

TGraphic 类在文件 graphics.hpp 里定义。TGraphic 类是存储和显示可视化图像 Tbitmap

(位图类), TIcon(图标类)和 TMetafile(图元文件类)的对象的抽象基类。由于是抽象基类, TGraphic 类不能被实例化, 许多继承的图像对象重载了 TGraphic 类的方法。TGraphic 类提供了与 TPicrure 对象和 TGraphic 类剪贴板协同工作的方法。当图像的类型(图标类、位图类、图元文件类)已知时, 则用相应的对象(TBitmap,TIcon,TMetafile)存储图像。否则, 可用 TPicture 类存储任意类型的 TGraphic 类图像。TGraphic 对象的常用属性、方法如表 6.9 和表 6.10 所示(其派生类也可以用):

表 6.9 TGraphic 对象的常用属性

方 法	含 义
LoadFromFile	将一个兼容类型的图形装入图形对象中
SaveToFile	将一个图形对象存入磁盘中

表 6.10 TGraphic 对象的常用方法

属 性	含 义
Empty	是只读布尔属性, 指明图形对象是否实际含有图形
Height	确定图形对象的高度, 以像素为单位
Width	确定图形对象的宽度, 以像素为单位

TBitmap 类是对 Windows 位图(HBITMAP)的封装, Palette 属性封装了其调色板(HPALETTE)。TBitmap 类包含位图的内部映像, 画图时自动管理调色板。每个位图包含一个 BITMAPINFO 数据结构和一个定义位图像素的矩阵两个部分。其图像易于创建、操作(调整比例、滚动、旋转、描绘)、在内存存储以及作为文件存储在磁盘中。TBitmap 类复制的很快, 因为它复制句柄而不是复制文件。

TMetafile 类是对 Win32 增强图元文件的封装。TMetafile 类包含增强型图元图像(EMF 文件格式)。图元文件存储容量小, 存储同样的图像用位图要几百 KB, 而用图元文件存储则只要 10KB~20KB。

TIcon 类是对 Windows 图标类的封装。TIcon 类包含图标文件(ICO 文件格式)。与 TBitmap 类和 TMetafile 类不同, TIcon 类创建的图像不能进行拉伸。

6.3.2 TPicture 类

TPicture 类包含了位图、图标、图元文件图像和用户自定义图像, 是这些图形的容器。通常通过 TImage 对象的 Picture 属性可以访问 TPicture 类, 其方法 LoadFromFile 和 SaveToFile 具有多态性, 能够处理在图像属性里定义的任意类型图像。而其他一些类则无此功能, 如 TIcon 类只能处理图标(ICO)文件。TPicture 类的属性有: Icon、Bitmap、Metafile、Graphic、Height 和 Width。

6.3.3 TImage 组件

TImage 组件通常用来在窗体中显示图像。在设计阶段以虚框显示于窗体表面, 作为其他图形组件的不可见的容器。其 Picture 属性定义实际显示的位图、图标、图元文件以及其他图像。它能存取 JPEG 图像文件、位图文件、图标文件、图元文件和增强图元文件。如果要在运行时装入图像要使用 Picture 属性的 LoadFromFile 方法。表 6.11 列举了 TImage 组件属性及含义。

表 6.11 TImage 组件属性及含义

属 性	含 义
AutoSize	确定图像是否调整自身大小以适应它所包含图像的尺寸
Canvas	定义一个 TCanvas 对象, 可以在 TImage 组件上绘图
Center	指定图像是否居于组件中央
Picture	确定显示的图像
Stretch	确定图像是否自动调整自身的大小, 对于 .ICO 文件, 该属性没有影响
Transparent	确定背景是否透明

6.3.4 TShape 组件

TShape 组件用于在窗体上显示一个简单的几何图形, 显示的几何图形由 Shape 属性确定, 其取值如表 6.12 所示。

表 6.12 Shape 属性取值及说明

Shape 取值	说 明
stRectangle	矩形
stSquare	正方形
stRoundRect	圆角正方形
StRoundSquare	圆角矩形
StEllipse	椭圆
stCircle	圆

另外, Tshape 组件还分别提供了 Pen 属性和 Brush 属性, 前者用于绘制几何图形的轮廓, 后者用于几何图形的填充模式。

6.3.5 TPaintBox 组件

TPaintBox 组件在窗体上提供了一个可以绘图的矩形区域。提供了一个 Canvas 属性, 利用此属性可以来绘图, 但 TPaintBox 组件无法记住 Canvas 中的内容, 为了解决这一问题, 需要重载 OnPaint 事件, 这样在组件或窗体无效时图形就会重画一遍。

6.3.6 创建和使用自定义光标

如果 C++Builder 中自带的光标, 没有一个是自己想要的, 需要把自己定义的光标加入到应用程序中去。

1. 创建自定义光标



图 6.3 是作者刚开始使用图像编辑器制作的等待光标, 只有 4 种颜色, 表明此时轮到程序思考。

图 6.3 自定义等待光标

C++Builder 的图像编辑器在新建一个位图时, 弹出 256 种颜色的调色板, 新建一个图标时, 弹出 16 种颜色的调色板, 新建一个光标时, 只弹出 4 种颜色的调色板。为了做出一个很漂亮的彩色光标, 作者使用图像编辑器制作了一个 256 色的光标, 具体做法如下:

- (1) 先做出一个 256 色的 32×32 像素的位图;
- (2) 接着打开一个 256 色的光标(可以从网上下载或利用现成的);
- (3) 通过复制位图再粘贴到已打开的光标里;
- (4) 修改光标名称, 保存光标资源。



于是就可以得到一个色彩众多的光标。图 6.4 是后期制作的等待光标, 表明此时轮到程序思考。

图 6.4 自定义等待光标

命名以上光标为“IDC_WCUR”的光标。保存资源文件为 MyRc.res。

无论用什么工具创建光标, 都要把它保存在资源文件(.res)内而不是光标文件(.cur)内。资源文件有两个好处: 第一, 它可以包含不同类型的多种资源, 比如, 一个资源文件可包含多个光标; 第二, C++Builder 可以方便地把一个资源文件连接到最后的可执行文件, 所以不必建立自己的光标文件。

2. 使用自定义光标

在#include 声明下, 写入下列代码定义用于标识新光标的常量。

```
const int crWaitCursor = 1;
```

在第 1 个 Pragma 资源声明下, 加入包含定制光标的资源声明。

```
#pragma resource "*.dfm"
#pragma resource "myrc.res"
```

加入下列代码以载入定制光标，将其分配给 MainForm:

```
Screen->Cursors[crWaitCursor]=LoadCursor(HInstance, "IDC_WCUR");  
Screen->Cursor =TCursor(crWaitCursor);
```

当一个应用程序开始时，C++Builder 自动创建一个指向 TScreen 的全局指针，名为 Screen。通过 Cursors 的简单数组来管理光标资源。

当应用程序开始时，Cursors 数组定义 18 个标准光标，它们的下标为-17~0，所以自定义光标从 1 加入。

一旦自定义光标被加入 Cursors 数组，它可以分配给任何组件或窗体中的 Cursor 参数。光标在移出组件或窗体之外后会变为自定义光标的形状。应用程序结束后，自定义光标便自动从内存中清除，因为它是 Cursors 数组的一部分。

完整代码参见 MainForm1.cpp 源文件。由于该文件太大，此处只列举了相关的部分：

```
#pragma resource "Chess.res"  
#pragma resource "MyRc.res"  
void __fastcall TMainForm::FormShow(TObject *Sender)  
{  
    .....  
    Screen->Cursors[crWaitCursor]=LoadCursor(HInstance, "IDC_WCUR");  
    //把自定义光标装入 Screen 对象  
    .....  
    Screen->Cursor = TCursor(crArrow); //分配 Windows 内置箭头光标给应用程序  
}
```

以上代码表明光标资源是在主窗体一显示就载入。

```
void TMainForm::ComputersTurn()  
{  
    WhoseTurn = computer;  
    Screen->Cursor =TCursor(crWaitCursor); //分配自定义等待光标给应用程序  
    .....  
}
```

以上是轮到计算机计算时的回调函数。程序处于机器控制状态，光标发生改变。

6.3.7 实现 CHESS 项目图像

如图 6.5 所示的是利用位图按钮 TBitBtn 组件的 Glyph 属性，在对象观察器中单击 TBitBtn 组件 Glyph 属性，弹出 Picture Editor 对话框，选择所要的 Check.bmp 图标即可。

图 6.6 是利用 TImage 组件的 Picture 属性，在对象观察器中单击 TImage 组件 Picture 属性，弹出 Picture Editor 对话框，选择所要的 CHESS.ico 图标即可。

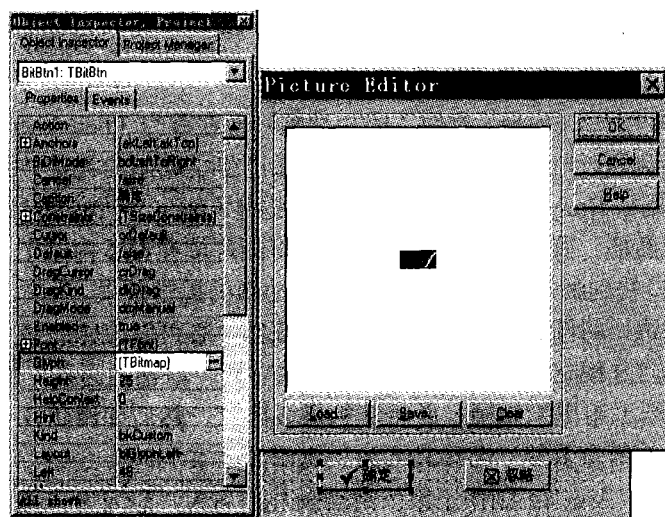


图 6.5 位图按钮的制作

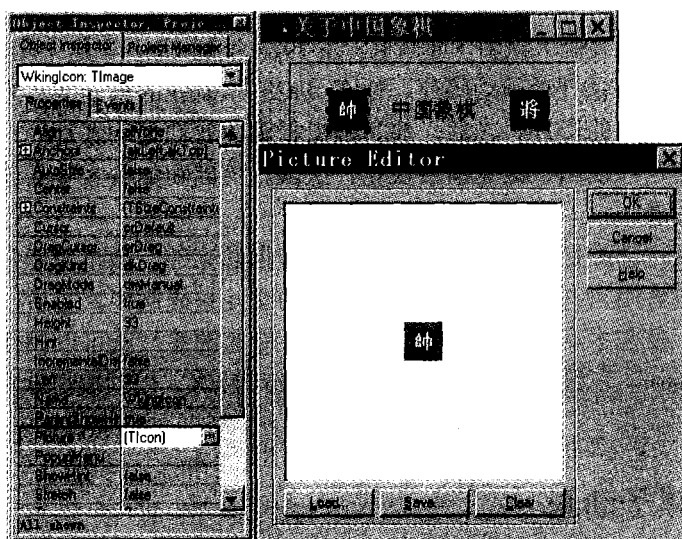


图 6.6 装入图标

6.4 生成动画

通常生成动画是将一系列画面连续显示而达到动画效果的。一般由软件本身提供的各类工具产生关键帧，安排显示的次序和效果，再组合成所需的动画即可完成。这次在象棋程序中应用的技术却不是关键帧技术，而是对象技术。

6.4.1 动画简史

1831 年由法国人 J.A.Plateau 发明了图片动画,他在称为 Phenakistoscope 的机器中放置了一个图画的转盘和一些用于观看的窗口,结果产生了动画的视觉效果。

1834 年,英国人 Horner 发展了上述思想,发明了称为“西洋镜”的机器,这种设备上有一个可旋转的圆桶,圆桶边缘有一些槽口,而画片放在设备内壁上。当圆桶旋转的时候,观众可通过槽口看到画片上的画面,从而产生动画的视觉效果。后来,法国人 E.Reynaud 对西洋镜进行了改进,将圆桶边上的槽口用置于圆桶中心可以旋转的镜子代替。

1906 年美国 J.S.Blackton 制作完成第一部名字叫“滑稽的面孔”的影片。1908 年法国人 E.Cohl 在白纸上用黑笔绘画,并将其拍摄成照片,然后使用负片在黑色背景上显示白色图形动画。他的最重要的一部作品名为“傀儡的故事”。

1909 年美国 W.McCay 制作了第一部动画片,名为“恐龙专家格尔梯”。虽然这部片子放映时间很短,但 McCay 仍然用了大约一万幅图画。在 1913 年~1917 年之间,美国制作完成了几部动画系列片,其中最著名的是由 P.Sullivan 制作的“小猫菲利斯”。

1915 年,美国人 E.Hurd 提出了图元动画(Cel animation),这个名字来源于制作这种动画片使用的透明的赛璐珞纸的英文名称。

商业动画片之父是美国人沃特·迪斯尼,在 1928 年到 1938 年这十年间,他制作了“米老鼠”和“唐老鸭”等系列动画动画片。他在 1928 年制作的第一个“米老鼠”短片是第一部具有同步音响效果的影片。之后,他又推出长篇商业动画动画片“白雪公主和七个小矮人”。与此同时,在俄国、捷克斯洛伐克、法国、英国和加拿大等其他国家也有一些动画影片制作的开拓者。

采用计算机制作动画是 20 世纪 60 年代中期发展起来的,最早的影片系采用程序设计语言编程制作,或者用只有计算机专家才能理解的交互系统制作。之后,对用户友善的交互系统不断发展,使艺术家们不需要太多的计算机专家的参与就可以制作影片。如二维动画制作软件 Animator、三维动画制作软件 3D Studio MAX、视频编辑软件 Adobe Premiere 等。

6.4.2 生成棋子动画

大多数计算机动画是运用两种主要方法:基于精灵模式的动画(对象动画)和基于帧模式的动画(帧动画)。基于帧模式的动画是用一连串静止帧组合起来给人一种连续运动的感觉。它实际上是模仿了录像和动画的工作方式。基于精灵模式的动画是在一个不变的背景上移动一个物体。一些高质量的动画程序采用以上两种方法的结合。

所有的动画方法都根据一条基本原理:在内存中处理图像,并把像素描到屏幕上。在内存中处理图像会提高处理速度,这是因为各种操作在系统内存中比在显示内存中执行起来要快得多。这种技术还减少了屏幕闪烁,因为中间画面不会被显示到屏幕上。

基于精灵模式的动画执行时,首先通过把已保存的背景区域复制到背景位图上来完成动画所在的背景区域要被填补的工作。复制只在动画的位置上进行,而且只复制足够多的

像素以擦除动画就可以了。背景位图被填补之后，动画的坐标就可以移动了。计算好新位置，存储了一部分背景图像后，把动画位图复制到背景位图的新坐标处。保存的背景区域将会被下一次调用来从背景中擦除动画。然后，用掩码位图在背景上透明地绘制动画。一旦动画在背景上绘好之后，就把这复合图像复制到屏幕。

选取一块动画所在的背景区域意思是说把欲画的动画的背景区域的像素涂黑。这里需要使用掩码位图和 Canvas->CopyMode 取值 cmSrcAnd 复制方式。对掩码位图和背景像素做“与”操作，和掩码位图中的白色像素相“与”的背景像素保持不变，和掩码位图中的黑色像素相“与”的背景像素会被消除并变为黑色。这一步在背景图像上产生一片动画的阴影。

将 Canvas->CopyMode 取值为 cmSrcPaint 是将动画位图和背景位图进行“或”操作，通过黑色阴影像素和动画像素的“或”操作可以完全地把动画复制到图像上，而动画的黑色像素与背景的非阴影像素相“或”却不使背景像素产生丝毫改变。

例如：棋子是一个简单的矩形，因此必须用掩码位图技术绘制它从而产生透明感。掩码图技术需要棋子的背景必须为黑色，掩码位图与棋子有着相同的形状。背景位图、棋子位图和棋子掩码位图如图 6.7 所示。

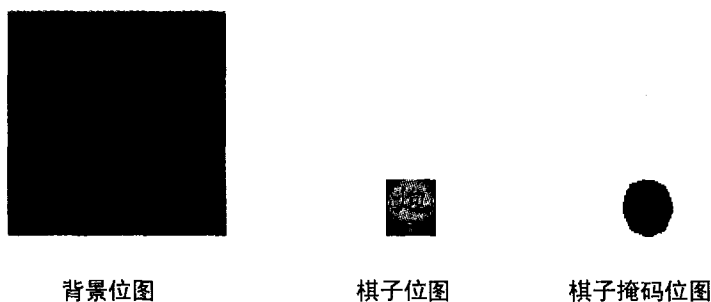


图 6.7 背景位图、棋子位图与棋子掩码位图

掩码位图中的白色像素在图像被显示时是透明的，掩码位图中的黑色像素在图像被显示时填充以棋子的像素。若动画对象中含有黑色区域，那么在掩码位图中该区域也必须是黑色的。否则，像素会成为透明的。

透明地绘制棋子，分两步：在背景上棋子所在的地方选出一块区域，使用掩码位图和“与”(AND)复制方式，然后用复制方式绘制棋子，相当于对图像作“或”(OR)运算。

6.4.3 棋子动画函数实现

CHESS 项目中有 3 种情况可以调用动画函数：当主窗体中的鼠标在棋子上按下时，选中棋子鼠标移动时，选中菜单“对弈”或“将死搜索”或“演示模式”时，主窗体的这 3 种事件将分别调用 UpdateScreen、DragStart 和 DragEnd3 个函数，它们再调用 SaveBackground、RestoreBackground 和 ShowPiece 从而实现动画过程。以下函数实际上是用户手工编写的自定义函数，位于 GlobalFun.cpp 文件中。

1. 函数名称: static void SaveBackground(POINT& p)

说明: 保存背景图像。这一保存的背景区域将会被下一次调用, 用来从背景中擦除棋子。

函数定义如下:

```
static void SaveBackground(POINT& p)
{
    HDC hDC;
    HBITMAP hBitmap;
    BITMAP Bitmap;

    LastPoint = p;
    hDC = GetDC(MainForm->Handle);

    hBitmap = PieceBmpArray[0][red]; // 位图句柄
    GetObject(hBitmap, sizeof(BITMAP), (LPSTR) &Bitmap);

    hSaveDC = CreateCompatibleDC(hDC);
    hOldBmp = (HBITMAP) SelectObject(hSaveDC,
        CreateCompatibleBitmap(hDC, Bitmap.bmWidth, Bitmap.bmHeight));

    BitBlt(hSaveDC, 0, 0, Bitmap.bmWidth, Bitmap.bmHeight,
        hDC, p.x, p.y, SRCCOPY);

    ReleaseDC(MainForm->Handle, hDC);
}
```

2. 函数名称: static void RestoreBackground()

说明: 把背景图像恢复为原始的形态。

函数定义如下:

```
static void RestoreBackground()
{
    HDC hDC;
    HBITMAP hBitmap;
    BITMAP Bitmap;

    hDC = GetDC(MainForm->Handle);
```



```
hBitmap = PieceBmpArray[0][red]; // 位图句柄
GetObject(hBitmap, sizeof(BITMAP), (LPSTR) &Bitmap);

BitBlt(hDC, LastPoint.x, LastPoint.y, Bitmap.bmWidth, Bitmap.bmHeight,
    hSaveDC, 0, 0, SRCCOPY);

DeleteObject(SelectObject(hSaveDC, hOldBmp));
DeleteDC(hSaveDC);
ReleaseDC(MainForm->Handle, hDC);
}
```

3. 函数名称: static void ShowPiece(POINT& p)

说明: 绘制透明棋子。

函数定义如下:

```
static void ShowPiece(POINT& p)
{
    HBITMAP hOldBmp;
    BITMAP Bitmap;
    HDC hMemoryDC, hDC;

    hDC = GetDC(MainForm->Handle);
    hMemoryDC = CreateCompatibleDC(hDC);
    GetObject(hDragPiece, sizeof(BITMAP), (LPSTR) &Bitmap);

    hOldBmp = (HBITMAP)SelectObject(hMemoryDC, hDragMask);
    BitBlt(hDC, p.x, p.y, Bitmap.bmWidth, Bitmap.bmHeight,
        hMemoryDC, 0, 0, SRCAND);

    SelectObject(hMemoryDC, hDragPiece);
    BitBlt(hDC, p.x, p.y, Bitmap.bmWidth, Bitmap.bmHeight,
        hMemoryDC, 0, 0, SRCINVERT);

    SelectObject(hMemoryDC, hOldBmp);
    DeleteDC(hMemoryDC);
    ReleaseDC(MainForm->Handle, hDC);
}
```

4. 函数名称: static void UpdateScreen(POINT& p)

说明: 执行动画——填补、移动和绘制。

函数定义如下:

```
static void UpdateScreen(POINT& p)
{
    HDC hDC, hUpdateDC, hNewSaveDC;
    HBITMAP hUOldBmp, hNewOldBmp;
    BITMAP Bitmap;
    RECT rect, rect1, rect2;
    int Width, Height;

    GetObject(hDragPiece, sizeof(BITMAP), (LPSTR) &Bitmap);

    rect1.left = p.x;
    rect1.top = p.y;
    rect1.right = p.x + Bitmap.bmWidth;
    rect1.bottom = p.y + Bitmap.bmHeight;
    rect2.left = LastPoint.x;
    rect2.top = LastPoint.y;
    rect2.right = LastPoint.x + Bitmap.bmWidth;
    rect2.bottom = LastPoint.y + Bitmap.bmHeight;

    if (!Intersect(rect1, rect2))
    {
        RestoreBackground();
        SaveBackground(p);
        ShowPiece(p);
        return;
    }

    hDC = GetDC(MainForm->Handle);

    hNewSaveDC = CreateCompatibleDC(hDC);
    hNewOldBmp = (HBITMAP)SelectObject(hNewSaveDC,
        CreateCompatibleBitmap(hDC, Bitmap.bmWidth, Bitmap.bmHeight));

    rect.left = min(rect1.left, rect2.left);
    rect.top = min(rect1.top, rect2.top);
```

```
rect.bottom = max(rect1.bottom, rect2.bottom);
rect.right = max(rect1.right, rect2.right);

Width = rect.right - rect.left;
Height = rect.bottom - rect.top;

hUpdateDC = CreateCompatibleDC(hDC);
hOldBmp = (HBITMAP)SelectObject(hUpdateDC,
                                CreateCompatibleBitmap(hDC, Width, Height));

/* 获取新的区域*/
BitBlt(hUpdateDC, 0, 0, Width, Height, hDC, rect.left, rect.top, SRCCOPY);

/* 恢复老区域*/
BitBlt(hUpdateDC, abs(rect.left - rect2.left), abs(rect.top - rect2.top),
        Bitmap.bmWidth, Bitmap.bmHeight, hSaveDC, 0, 0, SRCCOPY);

/* 存储新的背景*/
BitBlt(hNewSaveDC, 0, 0, Bitmap.bmWidth, Bitmap.bmHeight,
        hUpdateDC, abs(rect.left - rect1.left), abs(rect.top - rect1.top),
        SRCCOPY);

/* 传输掩码位图 */
DeleteObject(SelectObject(hSaveDC, hDragMask));

BitBlt(hUpdateDC, abs(rect.left - rect1.left), abs(rect.top - rect1.top),
        Bitmap.bmWidth, Bitmap.bmHeight, hSaveDC, 0, 0, SRCAND);

/* 传输棋子*/
SelectObject(hSaveDC, hDragPiece);
BitBlt(hUpdateDC, abs(rect.left - rect1.left), abs(rect.top - rect1.top),
        Bitmap.bmWidth, Bitmap.bmHeight, hSaveDC, 0, 0, SRCINVERT);

/* 棋子在背景上绘好后, 把复合图像复制到屏幕*/
BitBlt(hDC, rect.left, rect.top, Width, Height, hUpdateDC, 0, 0, SRCCOPY);

SelectObject(hSaveDC, hOldBmp);
DeleteDC(hSaveDC);
hSaveDC = hNewSaveDC;
```

```
hOldBmp = hNewOldBmp;
LastPoint = p;
DeleteObject(SelectObject(hUpdateDC, hUOldBmp));
DeleteDC(hUpdateDC);
ReleaseDC(MainForm->Handle, hDC);
}
```

5. 函数名称: void DragStart(SQUARETYPE square, POINT& point)

说明: 开始移动棋子时保存现场。

函数定义如下:

```
void DragStart(SQUARETYPE square, POINT& point)
{
    point.x -= 18;
    point.y -= 18;

    StartPoint = GetSquareXY(square);

    hDragPiece=PieceBmpArray[ChessBoard[square].
    piece-1][ChessBoard[square].color];
    hDragMask = MaskArray[ChessBoard[square].piece-1];

    Screen->Cursor=TCursor(crNone); //动态分配光标给应用程序
    SetCapture(MainForm->Handle); //捕获光标, 禁止输入
    ClearSquare(square);
    SaveBackground(point);
    ShowPiece(point);
    IsDragging = true;
}
```

6. 函数名称: void DragEnd(bool ValidMove)

说明: 移动结束后, 恢复现场, 同时允许在窗口输入命令。

函数定义如下:

```
void DragEnd(bool ValidMove)
{
    if (!IsDragging)
        return;
    if (ValidMove)
        RestoreBackground();
}
```

```
else
{
    LineDDA(CurPoint.x, CurPoint.y, StartPoint.x, StartPoint.y,
        LINEDDAPROC(ReturnHome), NULL);
    RestoreBackground();
}
Screen->Cursor=crDefault;
ReleaseCapture();
IsDragging = false;
}
```

第 7 章 跟踪 CHESS 项目——消息处理

所有的 C++Builder 类都有内置的消息处理机制。它们接收某些消息和分派消息，依据接收的消息调用一系列指定的方法。对某一消息如果没有指定的方法，就进行默认的处理。但是，有时仍然需要自己处理 C++Builder 未处理的消息或创建自己的消息。

7.1 Windows 消息与象棋窗体事件

消息是 Windows 操作系统送往应用程序的事件。Windows 消息数据结构中最重要的是一个标识消息的整型值。许多消息定义在 MESSAGES.HPP 中。传统的 Windows 程序有个窗口过程作为系统消息的回调函数。在窗口过程中，通常有一个庞大的 switch 语句，且对每个有待处理的消息有一个 case 语句。

附加的信息作为两个参数 wParam 和 lParam 传递给窗口过程，分别用于“字类型参数”和“long 类型参数”。通常，每个参数都含有不止一个信息，因此需要用 LOWORD 和 HIWORD 宏取出相应的信息。例如，调用 HIWORD(lParam)得到该参数的高字。

C++Builder 采用的是事件处理系统。事件由 VCL 组件和系统产生，而且通过事件处理程序传递应用程序窗体和其他对象。调用这些事件处理过程，其参数用来通知应用程序做所需的信息。在 C++Builder 中，几乎每个 Windows 消息都由特定的 C++Builder 事件封装起来。如 Windows 的窗体重画消息 WM_PAINT 在 C++Builder 中被 OnPaint 事件封装，用户只需在 OnPaint 事件处理过程中加入代码，而不必关心消息的具体处理过程。

由于 C++Builder 采用了事件处理系统，程序员在大多数情况下不需要理会 Windows 消息处理的细节，只要将注意力放在程序的事件处理函数中即可。表 7.1 是 Windows 消息与象棋窗体事件对照表，从表 7.1 可以看出，内置的象棋窗体事件并不足够多，我们在处理 CHESS 项目发现需要处理有些消息时，在将要处理的主窗体对象中并不存在相应的事件，这就需要在应用程序中添加消息映射进而处理相应的消息。

表 7.1 Windows 消息与象棋窗体事件对照表

Windows 消息	象棋窗体事件
WM_CREATE	OnCreate
WM_DESTROY	OnClose
WM_LBUTTONDOWN,WM_RBUTTONDOWN	OnMouseDown
WM_LBUTTONUP,WM_RBUTTONUP	OnMouseUp
WM_MOUSEMOVE	OnMouseMove
WM_PAINT	OnPaint
WM_TIMER	OnTimer

要重点说明的是，对于单击鼠标和选择菜单的消息 WM_COMMAND，在 C++Builder 中变为 OnClick 事件。比如，单击主窗体的名为“Stop”的菜单，不需要再应用 PostMessage 或 SendMessage 发送消息，它已设置 MSG 消息结构中的 wParam 参数为 Stop->Command，如果要获取此参数，可采用 PeekMessage 或 GetMessage 函数即可。

7.2 自定义象棋消息

定义自己的消息的时候，需要创建新消息处理器。包括定义自己的消息和声明一个新的消息处理方法。

7.2.1 声明一个消息标识符

消息标识符是个整型常量。Windows 为自己保留小于整数 1024 的消息，因此，在声明消息时，应从大于整数 1024 开始。

常量 WM_APP 是自定义消息的起始值。当自定义消息时，应从 WM_APP 为基础进行增加。如：

```
#define EM_ERROR          24344
#define WM_MYMESSAGE      (WM_APP+300)
```

CHESS 项目中定义了如下消息常量：

```
#define IDM_ERROR          10010    //程序出错时发送的常量
#define IDM_FILENEW        10020    //重新开始对弈时发送此常量
#define IDM_EXIT           10041    //结束游戏时发送此常量
```

7.2.2 声明一个消息结构

如果想给消息参数命名，只要声明一个消息结构类型就可以了。消息结构就是传递给消息处理方法的参数类型。例如语句：

```
TMessage &Msg;
```

声明变量 Msg 为指向消息结构 TMessage 的指针。

C++Builder 5.0 默认的消息结构是 TMessage，在其安装目录下的 Messages.hpp 文件中有说明。下面是 TMessage 消息结构：

```
struct TMessage
{
    unsigned Msg;
    union
```

```
{
    struct
    {
        Word WParamLo;
        Word WParamHi;
        Word LParamLo;
        Word LParamHi;
        Word ResultLo;
        Word ResultHi;
    };
    struct
    {
        int WParam;
        int LParam;
        int Result;
    };
};
};
```

老版本的 Windows 消息结构默认为 MSG(如 C++ OWL、Visual C++)，定义如下：

```
typedef struct tagMSG {    // msg
    HWND  hwnd;
    UINT  message;
    WPARAM wParam;
    LPARAM lParam;
    DWORD time;
    POINT pt;
} MSG;
```

理解消息结构可以通过提取相应的参数灵巧地设计出高水平的程序。

CHESS 项目在映射消息时主要用到 TMessage 消息结构，但在应用 Windows API 函数时还得使用 MSG 消息结构。如 PeekMessage、GetMessage 函数都是使用 MSG 消息结构的。

7.2.3 定义消息处理句柄

消息处理句柄即消息处理函数，有两种情况需要程序员声明消息处理函数：一是组件中不含有用于处理该消息的事件处理句柄，即程序员不能创建该消息的事件处理过程；另外一个自定义消息的消息处理函数。

在修改组件的消息处理之前，必须确保确实是想做的。C++Builder 把大多数 Windows 消息转换为组件开发员和组件用户都能处理的事件。应尽可能修改事件处理，而不是修改消息处理。

要修改消息处理，需要重载消息处理方法。做法是：

- (1) 在组件中声明一个新方法。
- (2) 用三个宏将该方法映射到消息。宏的形式如下：

```
BEGIN_MESSAGE_MAP  
    MESSAGE_HANDLE(param1, param2, param3)  
END_MESSAGE_MAP
```

其中 param1 是 Windows 定义的消息索引，param2 是消息结构的类型，param3 是消息方法的名字。

可以在 BEGIN_MESSAGE_MAP 和 END_MESSAGE_MAP 两个宏之间放置任意个想放置的 MESSAGE_HANDLE 宏。

在消息处理方法的内部，组件可以访问消息结构的所有参数。由于传递给消息处理器的是个指针，因此在需要时，处理器可以修改参数值。消息的返回值是被修改的最频繁的参数，该值是 SendMessage 的返回值。

7.2.4 发送和传递消息

当应用程序通过事件处理系统来处理 Windows 消息的时候，发送和传递消息是通过事件的触发来完成的，当应用程序直接处理 Windows 消息的时候，发送和传递消息是通过操作系统来完成的。当然通过触发事件来发送和传递消息归根结底也是通过操作系统来完成的，当应用程序处理用户自定义消息的时候，VCL 提供了 Perform 方法来完成消息的发送和传递。Perform 函数也可以发送和传递 Windows 消息，Perform 的函数原型为：

```
int __fastcall Perform(Cardinal Msg,int WParam,int LParam)
```

Perform 避开 Windows 的消息系统，直接将消息发送到指定的窗口。也可使用 PostMessage 或 SendMessage 函数。

- **PostMessage:** 该函数将一个消息放入(寄送)到与指定的窗口创建的线程相联系的消息队列里不等待线程处理消息就返回。
- **SendMessage:** 该函数将指定的消息发送到一个或多个窗口。此函数为指定的窗口调用窗口程序，直到窗口程序处理完消息再返回。而 PostMessage 不同，将一个消息寄送到一个线程的消息队列后立即返回。

7.3 CHESS 项目消息处理实例

为了扩展 CHESS 窗体事件调用，增强程序的功能，必须在 CHESS 项目中添加消息处理器。包括在主窗体包含文件中加以声明，在主窗体函数定义文件中加以定义，具体如下：

- (1) 选择 File | Open 命令打开 Open 对话框, 利用该对话框打开主窗体 MainForm1.h 包含文件;
- (2) MainForm1.h 包含文件的类 TMainForm 的 __published 部分由 IDE 自动管理, 不需要改动, 而 private 及 public 部分可由用户修改。

下面是在 MainForm1.h 文件中用手工为类 TMainForm 的自定义消息 IDM_ERROR、IDM_FILENEW、IDM_EXIT 进行声明的处理器。

```
class TMainForm : public TForm
{
private:
MESSAGE void ELError(TMessage &Msg);
MESSAGE void FileNew(TMessage &Msg);
MESSAGE void Exit(TMessage &Msg);
public:
BEGIN_MESSAGE_MAP
MESSAGE_HANDLER(IDM_ERROR, TMessage, ELError);
MESSAGE_HANDLER(IDM_FILENEW, TMessage, FileNew);
MESSAGE_HANDLER(IDM_EXIT, TMessage, Exit);
END_MESSAGE_MAP(TForm);
}
```

也就是说, IDM_ERROR 消息将触发 ELError 函数, IDM_FILENEW 消息将触发 FileNew 函数, IDM_EXIT 消息将触发 Exit 函数。

- (3) 选择 File | Open 命令打开 Open 对话框, 利用该对话框打开主窗体函数定义文件 MainForm1.cpp; 在其中具体实现每个函数的功能, 见下面的定义:

● ELError 函数

```
void TMainForm::ELError(TMessage &Msg)
{
    TChessErrorDlg *ChessErrorDlg= new TChessErrorDlg(Application,
    (AnsiString)buffer);
    ChessErrorDlg->ShowModal();
    delete ChessErrorDlg;
}
```

该函数将弹出象棋错误对话框。

● FileNew 函数

```
void TMainForm::FileNew(TMessage &Msg)
{
    MainForm->NewChessGame1Click(NewChessGame1);
}
```

该函数将重新建立一个棋盘。

- Exit 函数

```
void TMainForm::Exit(TMessage &Msg )  
{  
    MainForm->Exit1Click(Exit1);  
}
```

该函数将结束象棋游戏。

通过以上创建消息处理器，程序运行时，就能处理用户自己的消息了。要说明的是，以上消息和消息处理函数是相互对应的。

第 8 章 拦截 CHESS 项目错误——异常处理

C++Builder 5.0 支持 C++异常处理、基于 C 的结构异常处理以及 VCL 异常处理，但只有 VCL 异常被内置于许多类中，可以在出现错误时自动发送异常，若不处理异常，VCL 将以默认方式处理。

8.1 异常的分类

异常是最近几年加入到 C++中的新功能。异常是打断进程正常执行的一个事件。异常不同于中断，中断是为了响应外部事件而产生的，异常通常是由指令流中的操作产生的。异常可以产生于系统硬件或者由应用程序产生。

C++Builder 支持三种异常处理，这三种异常处理分别是：结构化异常处理、C++异常处理和 VCL 异常处理，现分别介绍如下。

8.1.1 结构化异常处理

结构化异常处理是 Windows 操作系统发生异常时使用的一种非标准的机制。它将可能产生的异常代码封装在 `try` 模块中，而响应异常的代码封装在 `__except` 模块中，或者使用 `__finally` 模块。结构化异常通过 `RaiseException()` 函数发送。

下面的例子说明 `try`、`__finally` 的使用。

```
#include <stdio.h>
#include <windows.h>

int main()
{
    float x,y,z;

    try
    {
        y=2.0;
        z=0.0;
    }
    try
```

```
{
    puts("一个异常。");
    x=y/z;
}

__except(EXCEPTION_EXECUTE_HANDLER)
{
    puts("捕获一个异常。");
}

__finally
{
    puts("终止块还是要执行。");
}

try
{
    {
        y=2.0;
        z=5.0;
        try
        {
            puts("没有异常。");
            x=y/z;
        }
        __except(EXCEPTION_EXECUTE_HANDLER)
        {
            puts("没有捕获到异常。");
        }
    }
    __finally
    {
        puts("没有捕获到异常，但终止块还是要执行!");
    }
    return x;
}
```

8.1.2 C++异常处理

在 C++异常处理中要求用到三个关键字 `try`、`catch`、`throw`。因此 C++异常处理分为三大块:

- `try` 块用于标志运行时间可能遇到特殊状况的函数代码准备捕捉异常。

- `catch` 块在 `try` 块后面，可有多，用来对应处理遇到异常事件的函数代码。它不能获取操作系统异常，而结构化异常 `except` 模块不能获取 C++ 的异常。
- `throw` 语句用于将程序代码运行时监视到的异常事件转移到另外异常处理函数代码中。

以下例子显示 C++ 异常处理的应用。

```
#include <condefs.h>
#include <iostream.h>

int main(int argc, char argv[])
{
    int *space;
    try
    {
        cout<<"正常。"<<endl;
        space=new int[512];
        if (space==NULL)
            throw "内存分配失败! ";
        else
            delete space;
    }
    catch(char *exception)
    {
        cout<<"异常"<<endl;
    }
    return 0;
}
```

以上程序必须命名为 `.cpp` 后缀，通过命令行使用 `bcc32.exe` 可成功编译并运行，而不使用 IDE。

8.1.3 VCL 异常处理

VCL 是用 Object Pascal 编写的，它不能很好地使用 C++ 异常处理。VCL 异常与 C++ 异常类似，而与 Windows 异常不一样，VCL 异常是不可恢复的；在产生异常后，不能再直接执行程序。VCL 异常不能是一个基本的类型，而 C++ 的异常可以是一个基本的类型。VCL 异常处理可以使用集成环境进行编译和运行，VCL 内置了一般的异常处理，这就使得 VCL 能够捕捉那些发生在消息处理期间被处理的异常。如果应用程序不能捕捉这些异常并且处理异常，VCL 将处理这些异常，以下介绍 VCL 异常处理。

VCL 默认的错误处理允许应用程序在异常发生时仍然能够继续运行,但是由于应用程序同时也为其成员函数分配了内存和其他系统资源,因此使用 C++语言中的 `try/catch` 块主动捕捉并且处理异常,就可以处理任何所需的清除任务。

下面是一个简单的用来发送异常的方法:

```
void __fastcall TForm1::IntExeceptBtnClick(TObject *Sender)
{
    try
    {
        throw 23;
    }
    catch (int ErrorCode)
    {
        ShowMessage(ErrorCode);
    }
}
```

上面的 `throw` 语句产生一个要被发送的异常, `catch` 语句用来获取错误。当然, `ErrorCode` 变量在产生异常时被设置为整型值 23。

要说明的是,这里的异常不是由错误产生,而是由使用关键字 `throw` 所产生。

8.2 VCL 异常类

C++Builder 的 VCL 库之所以能够捕捉和处理绝大部分的出错情况,关键在于 C++Builder 内置了许多异常类用来处理可能发生的各种异常。程序员也可以方便地创建自己的异常类用来处理程序中可能发生的特殊出错事件。下面所列的是 C++Builder 异常处理的基类,它的说明在 `Sysutils.hpp` 中:

```
class DELPHICLASS Exception;
class PASCALIMPLEMENTATION Exception : public System::TObject
{
    typedef System::TObject inherited;

private:
    AnsiString FMessage;
    int FHelpContext;

public:
    __fastcall Exception(const AnsiString Msg);
```

```

__fastcall Exception(const AnsiString Msg, const System::TVarRec * Args,
const int Args_Size);
__fastcall Exception(int Ident)/* overload */;
__fastcall Exception(System::PResStringRec ResStringRec)/* overload
*/;
__fastcall Exception(int Ident, const System::TVarRec * Args, const int
Args_Size)/* overload */;
__fastcall Exception(System::PResStringRec ResStringRec, const
System::TVarRec * Args, const int Args_Size
)/* overload */;
__fastcall Exception(const AnsiString Msg, int AHelpContext);
__fastcall Exception(const AnsiString Msg, const System::TVarRec * Args,
const int Args_Size, int AHelpContext
);
__fastcall Exception(int Ident, int AHelpContext)/* overload */;
__fastcall Exception(System::PResStringRec ResStringRec, int
AHelpContext)/* overload */;
__fastcall Exception(System::PResStringRec ResStringRec, const
System::TVarRec * Args, const int Args_Size
, int AHelpContext)/* overload */;
__fastcall Exception(int Ident, const System::TVarRec * Args, const int
Args_Size, int AHelpContext
)/* overload */;
__property int HelpContext = {read=FHelpContext, write=FHelpContext,
nodefault};
__property AnsiString Message = {read=FMessage, write=FMessage};
public:
#pragma option push -w-inl
/* TObject.Destroy */ inline __fastcall virtual ~Exception(void) { }
#pragma option pop
};

```

VCL 异常类都包含 **Message** 属性，它可以对异常进行文本描述，程序员可以利用这一属性将这一消息传给几个不同的构造函数，并且通过 **Message** 属性来获取这个消息，为用户显示异常信息，如表 8.1 所示。

表 8.1 异常类

异 常 类	含 义
EAbort	不显示错误消息对话框，终止事件序列
EAccessViolation	检查无效的内存访问错误
EBitsError	阻止无效地访问一个逻辑数组
EComponentError	无效地重命名或注册组件错误
EConvertError	指出字符串或其他对象转换错误
EDatabaseError	数据库访问错误
EDBEditError	与指定模式不兼容的错误
EDivByZero	整数被零除的错误
EInOutError	指出文件 I/O 错误
EIntOverflow	指出整数计算结果超过分配的寄存器的大小
EInvalidCast	不合法类型转换产生的异常
EInvalidGraphic	试图操作未被识别的图形文件格式产生的异常
EInvalidOperation	非法操作组件产生的异常
EInvalidPointer	非法使用指针产生的异常
EMenuError	菜单项包含的异常
EOleCtrlError	检测 ActiveX 组件产生的异常
EOleError	OLE 自动化异常
EPrinterError	打印异常
EPropertyError	设置属性失败产生的异常
ERangeError	整数赋值超过类型规定的范围
ERegistryException	注册表异常
EStackOverflow	由于堆栈溢出产生的异常
EZeroDivide	浮点数零除产生的异常

8.3 发送异常

下面代码中的 `throw` 语句将创建一个 `Exception` 的实例对象，并调用 `Exception` 的构造函数，所有 `Exception` 异常类的派生类都有 `Message` 属性，该属性通过构造函数获取值，因此通过该属性可以获取用户定制显示消息。

```
int StrToInt(AnsiString S)
{
    int i = atoi(S.c_str());
    if (i == 0)
        throw EConvertError("Cannot convert" + S + "to an int");
    return i;
}
```

这个函数使用了 `atoi` 函数，以便把字符串转换为整数。如果转换成功，`atoi` 函数返回转换后的字符串。如果出现问题，`atoi` 函数返回 0。当出现错误时，`StrToInt` 程序就发送一个 `EConvertError` 异常，并且返回一个说明出现什么错误的字符。

8.4 捕获异常

程序员可以将异常类作为一个参数传递给 `catch` 块，语法如下：

```
catch(const exception_class&exception_variable)
```

例如：

```
try
{
    x= StrToInt(Edit1->Text);
}
catch
{
    ErrorDlg("数值输入错误。请重输！");
}
```

可以看到在 `catch` 块中捕获一个异常类的引用就相当于捕获这个类，现在遇到这一类型的异常都会被 `catch` 块捕获，其他类型的异常按默认处理。

8.5 定制异常

与 C++Builder 的事件不能封装所有的消息一样，并非所有具体应用程序中有可能发生的异常都能被捕捉并处理，如果我们不满足 C++Builder 自动处理异常的方式，这时创建自定义的异常类也是一种方法。为了创建自定义的异常类，首先应该说明一个新的类：

```
class ESpellingError:public Exception //声明一个自定义的异常子类
{
public:
    __fastcall ESpellingError(const AnsiString Message)
        :Exception(Message){}
};
```

这段代码表示 `ESpellingError` 是 `Exception` 类的子类。

以下为了检查单词 `Hello` 是否拼写正确：

```
void __fastcall TForm1::EWriteClick(TObject *Sender)
```

```
{  
AnsiString S = Edit1->Text;  
if (S == "Helo")  
{  
    S = "Hello 写错! "+ "当前是 Helo!";  
    throw ESpellingError(S + "\r 请重输!");  
}  
}
```

8.6 运行象棋异常处理

CHESS 项目加入了异常处理，从而增强了程序的健壮性，下面举例说明。

如图 8.1 所示的是在红色编辑框中输入一个字母，当输入焦点退出时发生整数转换异常，程序捕获该异常，弹出错误对话框。

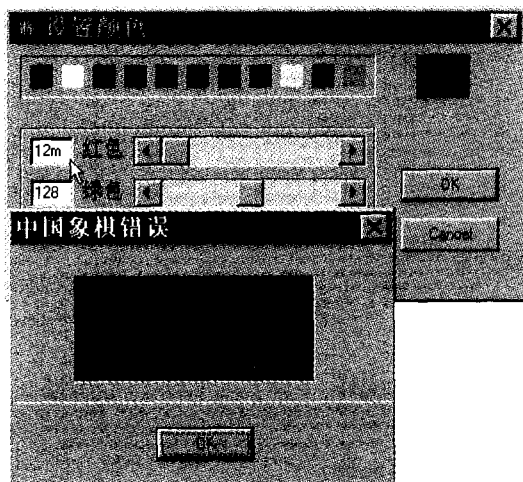


图 8.1 整数转换异常

为了捕获 CHESS 项目运行时用户在改变颜色对话框的文本框中输入非数字或数字值不在 0~255 之间的错误，需要加入异常处理程序。当发出一个异常时，try 代码块被保护起来，运行库搜索调用堆栈，catch (EConvertError &error)捕获整数转换错误。当捕捉到了该异常后，显示出错消息，并把输入聚焦到编辑控件中。在没有将其完全恢复之前将继续弹出错误对话框。其代码实现制作步骤如下：

- (1) 选择 File | Open Project 命令，打开 Open Project 对话框，利用该对话框打开 Chess.bpr 项目；
- (2) 选择 File | Open 命令，打开 Open 对话框，利用该对话框打开 ColorDlog.h 包含文件，在 TcolorDlg 类的私有部分 private 加入如下函数声明：

```
bool GetColorValue(TObject *Sender);
```

(3) 选择 View | Project Manager 命令, 打开 Project Manager 对话框, 利用该对话框打开 ColorDlg.cpp 源文件;

(4) 手工添加如下代码:

```
bool TColorDlg::GetColorValue(TObject *Sender)
```

```
{  
    int NewVal;  
    bool Ok;  
    BYTE *CurVal;  
    TScrollBar *Scroller;  
    TEdit *Sender1 = (TEdit *)Sender;  
    Ok = true;  
    try  
    {  
        NewVal = StrToInt(Sender1->Text);  
    }  
    catch (EConvertError &error)  
    {  
        Ok=false;  
    }  
}
```

```
if (Sender1==EditRed)
```

```
{  
    CurVal = &SqColors[0];  
    Scroller = ScrollBar_Red;  
}
```

```
else if (Sender1==EditGreen)
```

```
{  
    CurVal = &SqColors[1];  
    Scroller = ScrollBar_Green;  
}
```

```
else if (Sender1==EditBlue)
```

```
{  
    CurVal = &SqColors[2];  
    Scroller = ScrollBar_Blue;  
}
```

```
if (Ok)
```

```

    {
        Ok = (NewVal > 255) ? false : ( (NewVal < 0 ) ? false : true );
    }
else
    NewVal = *CurVal;

if (!Ok)
    {
        Error("请输入一个介于 0~255 之间的有效数.");
        Sender1->SetFocus();

    }
else if (NewVal != *CurVal)
    {
        *CurVal = (BYTE)NewVal;
        Scroller->Position=*CurVal;
    }

ColorSelection->Color = TColor( RGB(
        ScrollBar_Red->Position,
        ScrollBar_Green->Position,
        ScrollBar_Blue->Position));
    return Ok;
}

```

- (5) 选择 View | Object Inspector 命令，打开 Object Inspector 对话框颜色编辑组件的“Events”选项卡双击 OnExit 的回调函数 EditExit，打开代码编辑器修改 ColorDlog.cpp 源文件，加入如下代码：

```

void __fastcall TColorDlg::EditExit(TObject *Sender)
{
    if( GetColorValue(EditRed) &&
        GetColorValue(EditGreen) && GetColorValue(EditBlue))
    {
        *
        ScrollBar_Red->Position = StrToInt(EditRed->Text);
        ScrollBar_Green->Position=StrToInt(EditGreen->Text);
        ScrollBar_Blue->Position=StrToInt(EditBlue->Text);
    }
}

```

该函数表示当鼠标退出编辑框时将调用 `GetColorValue()` 函数，判断用户在编辑框中输入的数字是否在 0~255 之间，如果不是，程序发出一个异常，显示出错消息，并把输入聚焦到编辑控件中。直到用户输入正确为止，才关闭弹出的错误对话框，并相应改变滚动条的位置。

完整的源代码请见所附光盘中的 `ColorDlog.cpp` 和 `ColorDlog.h` 文件。

第 9 章 人工智能与中国象棋

采用人工智能的方法研究中国象棋，是 20 世纪 80 年代起步的。当时开局、残局阶段采用棋谱信息库，将象棋专家知识储存于计算机中备用，需要时搜索取出，中局阶段采用棋路树搜索法，并设计了局面评估函数。经过 20 年的发展，对中国象棋的研究更加深入了，但要想更快地发展，需要付出汗水和掌握广博深入的知识。

9.1 人工智能简介

人工智能是在计算机科学、控制论、信息论、神经生理学、心理学、哲学和语言学等多种学科相互渗透的基础上发展起来的。它用计算机程序来做某些工作，这些工作如果让人来做的话，就需要智能。因此，也可称之为机器智能(Machine Intelligence)，人工智能的英文名称是 Artificial Intelligence，缩写为 AI。

9.1.1 人工智能的研究方法

人工智能研究的最佳方法一般可分为 3 大类。

第 1 类是符号主义。它们基于纽威尔(Newell)和西蒙(Simon)的物理符号系统的假说和有限合理性原理。尽管不是所有人都赞同这一假说，但几乎大多数被称为“经典的人工智能”均在其指导之下。这类方法中，突出的方法是将数理逻辑操作应用于说明性知识库。最早由麦卡锡(John McCarthy)的“采纳意见者”备忘录提出，这种风格的人工智能运用说明语句来表达问题域的“知识”，这些语句基于或实质上等同于一阶逻辑中的语句。采用逻辑推理可推导这种知识的结果。这种方法有许多变形，包括那些强调对逻辑语言中定义域的形式公理化的角色的变形。当遇到“真正的问题”，这一方法需要掌握问题域的足够知识，通常就称作基于知识的方法。其代表性的成果是 1957 年研制的逻辑理论机的数学定理证明程序 LT(逻辑理论家)。

在大多数符号处理方法中，对需求行为的分析和为完成这一行为所做的机器合成要经过几个阶段。最高阶段是知识阶段，机器所需知识在这里说明。接下来是符号阶段，知识在这里以符号组织表示(例如表可用列表处理语言 LISP 来描述)，同时在这里说明这些组织的操作。接着，在更低级的阶段里实施符号处理。多数符号处理采用自上而下的设计方法，从知识阶段向下到符号和实施阶段。符号主义者走过的是一条启发式算法→专家系统→知识工程的发展道路。

第 2 类是行为主义。它们通常采用自下而上的方式，从最低阶段向上进行。突出的方法是“动物进化”。偏爱这种方式的人们指出，人的智能经过了在地球上十亿年或更长时

间的进化过程。他们认为,为了制造出真正的智能机器,必须沿着这些进化的步骤走。因此,必须集中研究复制信号处理的能力和简单动物如昆虫的支配系统,沿着进化的阶梯向上进行。这一方案不仅能在短期内创造实用的人造物,又能为更高级智能的建立打好坚实的基础。于是布鲁克成功研制了一个由 150 个传感器和 23 个执行器构成的像蝗虫一样能做六足行走的机器虫,虽然不具有像人那样的推理、规划能力,却具有灵活的防碰撞和漫游行为。布鲁克将物理符号系统和他的物理基础假说相对照。在物理基础假说中,一个智能体不采用集中式的模式而运用其不同的行为模块与环境相互作用来进行复杂的行为,对传统人工智能进行了批评和否定。

第 3 类是联结主义,又称仿生学派或生理学派。其代表成果是 1943 年由麦克洛奇和皮兹创立的脑模型(MP 模型)。20 世纪 60 年代到 20 世纪 70 年代中期,联结主义出现过热潮,20 世纪 70 年代到 20 世纪 80 年代初期落入低潮,直到 1982 年提出了霍普菲尔德神经网络模型(HNN 模型)后才开始复苏,1986 年提出了反向传播(B-P)算法,联结主义在新的技术条件下又掀起了一个研究热潮。

9.1.2 人工智能简史

人工智能作为一个新兴学科的名称被使用以来,已经有四十多年的历史了,回顾其产生与发展过程有助于加深对人工智能的认识。

1. 孕育期(1956 年以前)

古希腊伟大的哲学家亚里士多德(公元前 384~322 年)在着手解释和编纂他称之为三段论的演绎推理时就迈出了向人工智能发展的早期步伐。

莱布尼兹把形式逻辑符号化。莱布尼兹(1646~1716 年)把“有一天所有的知识,包括精神和无形的真理,能够通过通用的代数演算放入一个单一的演绎系统”的梦想变为可能。莱布尼兹称这个系统为微积分原理机,或推理机。

英国数学家布尔(Boole)于 1854 年建立并发展了命题逻辑(布尔代数),这方面有了实质性的进展。布尔的意图是要“把有关人类意识的本质和构成的某些可能的暗示收集起来”。

到了 19 世纪末期,弗雷格提出了用于机械推理的符号表示系统,从而发明了我们现在熟知的谓词演算,他称之为“概念书写”。

另外不能忘记的是图灵、冯·诺依曼、莫克利、维纳、申农、麦克洛奇和皮兹等。

2. 形成期(1956 年~1970 年)

从 20 世纪 60 年代至 70 年代初探讨了问题表示、搜索技术和通用启发等一系列问题,并把它们运用于计算机程序中来解迷、博奕(塞缪尔的跳棋程序)和检索信息。

1956 年,麦卡锡和申农合作编著了一本名为《自动机研究》的书。1956 年夏季的历史性聚会符号主义者最先正式采用人工智能来命名。在该次会议上发表了许多重要论文,包括由纽威尔、肖和西蒙编写的名为《逻辑理论家》的程序,它可以证明《数学原理》书中 38 条定理。

1956 年, 赛缪尔研制成功具有自学习能力的启发式博弈程序。

1956 年, 乔姆斯基提出了语言文法, 开创了用于思维与感知的形式语言研究。

1957 年, 罗森勃劳特研制成感知机, 具有学习功能, 模仿视觉, 进行模式分类研究。

1958 年, 麦卡锡建议在他称之为“意见采纳者”的系统中采用谓词演算这种语言来表示和运用知识。这一系统要求拥有它所需要知道的而不是事先程序设计好的知识。Covdell Green 在他的 QA3 系统中适度地、颇有影响地实现了这些思想。谓词演算和一些它的变形构成了人工智能知识表示的基础。

1960 年, 纽威尔、肖、西蒙合作研制了 GPS “通用解题者”, 该程序当时可以解决 11 种不同类型的问题, 如不定积分、三角函数、代数方程、猴子摘香蕉等。从而使启发式程序有更大的普适性。

1960 年, 麦卡锡研制成功 LISP “表处理语言”, 不仅可处理数值, 而且可方便地处理符号。

1961 年, 明斯基发表了论文“走向人工智能的步骤”。

1965 年, 费根鲍姆开创了从“知识库”观点研制 DENDRAL 专家系统的工作。

20 世纪的逻辑学家, 包括哥德尔、凯勒、波斯特、乔奇和图灵, 对哪些能和哪些不能由逻辑和计算机系统完成的任务做了形式化分类。计算机科学家, 包括 Stephen Cook 和 Richard Karp 证明有些计算在原则上可能需要根本不切实际的时间和存储空间。许多从逻辑学和计算机科学中所得到的结果是: “真理不可能被演绎”、“计算不可能被执行”。也许这些负面的发现令许多哲学家和其他人振奋, 他们将之理解为再一次否定了人类的智能可以机械化, 他们猜想人类不存在机械所固有的计算局限。然而多数逻辑学家和计算机科学家却认为这些负面结果并不暗示机器具有人类所不具有的任何局限。

3. 知识应用期(1971 年~80 年代末)

为了应用于更重要的现实问题而对这些程序和技术进行升级的尝试表明这些系统只能解决“玩具问题”。更有效的系统要求对应用领域具有更多内在的知识。20 世纪 70 年代末 80 年代初发展了一些更高级的程序, 包括在完成一定任务时模拟专业知识、分析、设计和诊断等。从一般思维规律探讨转向知识工程开发。一些表达具体问题的知识得到了探讨和发展。

第 1 个能演示具体领域知识的重要程序化学专家 DENDRAL 是一个根据所提供的化学分子式和质谱分析图来预测有机物分子结构的系统。接着, 其他“专家系统”, 包括医学专家 MYCIN, INTERNIST; 矿藏评估专家 PROSPECTOR; 生物专家 MOLGEN; 故障诊断专家 DART; 教育专家 GUIDON; 结构专家 SACON; 法律专家 LDS; 管理专家 CALLISTO 等, 相继研制成功, 并开始应用。

人类的智能包括洞察和分析可视化场景、理解并运用语言等许多方面的能力。关于这些能力的专题得到了高度重视。Larry Roberts 开发了早期场景分析程序之一。这一工作之后对机器视觉作了大量研究, 对动物视觉系统也作了研究。

20 世纪 70 年代, 在一个多方项目中, 连续语言理解系统原型被开发出来; 由 William Woods 开发的 LUNAR 系统能回答用口语提出的关于由美国航空航天局(NASA)从月球收集的岩石样品的问题。尽管现在存在一些自然语言理解系统, 但它们的能力仅局限于特定

的话题和词汇。打破这些局限有待于开发出更大量的知识表示。CYC 项目的一个目标就是尽可能多地收集、表达这些所需的知识。

4. 综合集成期(20 世纪 80 年代末至今)

20 世纪 80 年代神经网络又恢复了活力。具有强度可调互连系统的非线性元素网络如今已被视为一类重要的非线性建模工具。现在已存在神经网络方面的一些重要应用。动态方法与神经网络相结合,促使人工智能的研究集中到把符号处理过程与处于物理环境中的机器人的传感器与受动器联系起来的问题上来。

博奕这一领域有了实质性的进展。1997 年 5 月 11 日,一个名为“更深的蓝”的 IBM 电脑在 6 局比赛中以 3.5 比 2.5 的总比分战胜了世界象棋冠军 Garry Kasparov(盖利·卡斯帕洛夫)。“更深的蓝”是美国 IBM 电脑公司生产的世界第一台超级国际象棋电脑,它拥有 32 个微处理器,每个微处理器上有八块专门设计的国际象棋芯片,平均运算速度为每秒 200 万步。所有 256 块芯片集成在 RS6000SP 并行计算系统中,构成了每秒 2 亿步令人生畏的运算速度。下棋程序运用复杂的搜索算法,思索当前局面下每一种可能的下法,排除种种不利的选择,从中挑选最佳着法。

今后,人工智能技术向着大型分布式人工智能、大型分布式多专家协同系统、广义知识表达、综合知识库、并行推理、多种专家系统开发工具、大型分布式人工智能开发环境和多智能体协同系统等方向发展。不断提高和完善机器人和软件智能体的能力将促进并引导人工智能研究。

9.1.3 人工智能问题的通用解答

在非人工智能的程序设计中,程序员通常先定出逐步解决某一类问题的方法,或者叫做“算法”,然后,他把这些算法步骤翻译成特定的程序语言,这样就得到一种程序,使计算机有可能去解决类似的许多问题。然而这种方法存在两个困难:有相当多的一类问题,数学家已经证明没有任何算法可以解答其中任何一个问题;另外,纵使有一种算法能够解决某一类别中的所有问题,可是这个算法只对其中很小一部分问题才有实际意义,对于一些较大的实际问题却仍然是无能为力的。

一般地说,执行一个算法所需要的时间是随着问题规模大小(即 n 值)而异的,不过对不同的算法所需的时间也是不同的。与 n 、 n^2 、 n^3 等成正比例的算法执行时间叫做多项式运行时间。以多项式时间来运行的算法被认为是能够实际执行的。

有些算法运行的时间是和形如指数 2^n ,阶乘 $n!$ 和幂指数 n^n 等成比例的,这样的算法叫做以指数时间运行的算法。以指数时间运行的算法,被认为是不能实际执行的,这一类算法只能成功地用于该类问题中非常少的一些方面。随着问题规模的增大,运行时间增加得非常迅速,以致即使一些中等大小的问题,也需要高速计算机数千个计算机机时,至于规模庞大的问题则需要几千年的计算机机时。

在人类每天所遇到的许多非常重要的问题中,其中有名的一些算法都是以指数时间来运行的。而且数学家认为,这类问题中,大多数问题所有算法都是需要以指数时间运行的。

这类问题甚至不存在任何一种可以解决它的一切问题的算法；而且即使存在有这样一种算法，它也是以指数运行的，可是人类却能应付自如地对付这类问题，其原因大概有以下3个方面：

- 任何一种算法都必须保证对于某一类问题的每个问题都起作用，即算法必须是完备的，对算法的要求是要绝对地正确无误。
- 人类所用的方法则只需用于一些合理的问题，或者是一些在实际中容易遇到的问题。对人类的做法来说允许有偶然的失败和错误。当一个做法在某一个问题上遭受失败时，可以马上转向另一方法。
- 人类并不总是坚持要找出一个准确的或者是最好的问题解答，而只要求这个解答足以近似地解决手头的问题就够了，而不必浪费时间去寻求比它更好的解答。

这样看来，好像人类的算法不如计算机的算法好；可是，实际上，人类在解决大多数实际问题时，其方法要比计算机的算法成功得多。典型的情况是，有时候一个算法在理论上说能解决某类问题中的全部问题，但当把它用于实际问题时，由于算法是以指数时间运行，却是根本不可能实现的。

不过，一个人虽然通常用的是一些可能会发生错误的、近似的、粗略的方法，可是它却能对付实际问题，使他能得到一个即使不是理论上最好的、但却是一个现实可行的解答。所以，人工智能科学家不打算建立能解决大量问题的算法，而是集中精力来发明一些有时候虽然可能错误，但却是人类所使用的那些往往可能更为成功的一些方法。不过人工智能对实际上存在着的能够解决当前问题的一些算法也并不放弃。

人工智能研究基础的一些技术是：搜索法、启发式搜索法、模式识别、规划和知识表达。象棋程序主要应用了启发式搜索法。

假如根据某种提示和经验规则来引导搜索朝正确方向进行，就可避免发生组合爆炸，那么根据提示就可不必检验每个可能的解答，这样也就可以缩小范围，把我们进行的搜索限定在一些可能是最可行的范围内。

“启发式”(heuristic)一词源出于希腊语，它的意思是“帮助发现”。一个启发式就是一种暗示或者一种经验规则，它可以帮助用户发现问题的解答，运用启发式进行的搜索法，就叫做“启发式搜索法”。

格言或谚语往往是一些很好的启发式的例子，例如：当头炮，马来跳；马对单士必胜；失败为成功之母。

每个格言都是一个良好的忠告，也就是说，每个格言在相应的情景下都是值得考虑的。可是另一方面，格言并不总是保证人们能得到想要得到的结果。引用格言的时候，不顾条件照搬照抄是不行的，还得具体问题具体分析，在某一种情景下某个格言是可能应用的，可是却不当应用它。譬如，当用户正在做一项根本不可能成功的工作时，那么“失败为成功之母”这句格言，就不能用了。

所有的启发式都有以下共同特点：它们都只适合于特定的情况，在这种情形下它们确实是值得考虑的，但是仍然不过是值得尝试的一些事情，它们在特定的形式下，或许起作用，也或许根本就不起作用。

9.2 敌对搜索

敌对搜索中, 双方都希望自己能够获胜, 从多种可供选择的行动方案中必须防止那种对自己最为不利的情况发生。

9.2.1 双方博弈的一般性理论

博弈是一类富有智能行为的竞争活动, 分为完备信息博弈和机遇性博弈。所谓机遇性博弈是指存在不可预测性的博弈, 如掷硬币。以下讨论双人完备信息博弈。

双人完备信息博弈过程中, 双方都希望自己能赢。这时, 一方建立的计划可以考虑到其他方的行为影响。特殊情况下, 这两方的行为是相互交替的, 它们可以考虑到对方的行为。其中之一先行动, 然后是另一个, 接着如此反复。

假如两个博弈者, 分别命名为“黑方”和“红方”。它们可以向其所在的行或列中的相邻一格交替地移动(例如, 红方先移动), 而且轮到其中一个时, 它必须移动。假设红方的目标是与黑方在同一格, 而黑方的目标是避免发生这种情况。红方就可建立一棵搜索树。

两个博弈者轮流移动, 直到其中任何一方获胜(另一方因此失败), 或双方和局。每个博弈者完全熟悉环境及自己和对方可能的移动方式和影响(尽管每个博弈者都不知道另一方在任何情况下究竟会怎样移动)。

为了选取最佳的起始动作, 红方需要分析这棵树来决定可能的结果——即: 考虑黑方会为了阻止红方采取对红方最不利的方法实现此目标。在这种冲突的情况下, 一个博弈者可能发现一个移动, 使得无论对方如何移动, 它都可达到目标。不过更常见的情况是, 由于计算和时间的限制, 无论哪方都不能找到一个动作以保证取得成功。在任何情况下, 在决定了第 1 步之后, 执行移动, 考虑另一方的可能移动, 然后在感知→计划→动作方式中重复计划过程。

可以看出, 有许多常见的博弈, 如国际象棋、西洋跳棋和围棋等, 都属于这种类型。

9.2.2 最小最大化过程

在以下讨论中, 命名两个博弈者 MAX 和 MIN。下面的任务是为 MAX 寻找最佳的移动。假设 MAX 先移动, 然后两个博弈者轮流移动。因此, 深度为偶数的节点, 对应于 MAX 下一步移动的位置, 称为 MAX 节点; 深度为奇数的节点对应于 MIN 下一步移动的位置, 称为 MIN 节点(博弈树的顶节点深度为 0)。K 层包括深度为 $2k$ 和 $2k+1$ 的节点。通常用层数给出博弈树的搜索程度, 它可以表示出向前预测的 MAX 和 MIN 交替运动的回合数。

完全的搜索(赢、输或和局)对于简单的博弈可以生成整个博弈树, 找到必胜的路径, 而对于大多数博弈来说是不可行的。据估计, 完全的国际象棋博弈图解大约有 10^{40} 个节点。即使假设一个后继节点可在 $1/3ns$ 内产生, 也需要 10^{24} 年才能产生国际象棋博弈完全搜索图解(据推测, 宇宙也只有大约 10^{10} 年的历史)。而且, 启发式搜索方法并不会减少起作用

的有效分枝因子。因此,对于复杂的博弈,必须认识到搜索到终点是不可能的(除了在博弈快结束时),所以,应该使用有限范围搜索方法。

用户可使用广度优先搜索、深度优先搜索或启发式搜索。除非必须马上修改终止条件,几个现实的终止条件为时间限制、存储空间限制以及在搜索树中最深节点的深度。例如,通常在国际象棋中,如果有任何叶节点代表可继续的好位置,比如说将军或吃子,即存在可以继续移动的好位置,就不终止。

搜索结束后,需从搜索树中选取一个最佳首次移动,这个选取方法可以采用静态评估函数对搜索树的叶节点进行静态估值。这种衡量基于影响这个值的许多不同特性;例如,在象棋中,一些有用的特性衡量相关部分优势、关键棋格的控制权、出子、机动性、中心控制、王的中心控制等等。通常分析博弈树时,对 MAX 有利的位置,评估函数将赋予正值;对 MIN 有利的位置赋予负值,接近零的值表示该位置对 MAX 和 MIN 都一样。

一个最佳首步可以由一个最小最大化过程产生(为简单起见,在描述这个过程和基于它的其他过程时,把博弈图当作一棵树)。假设轮到 MAX 从搜索树的叶节点中选取,他肯定选择拥有最大值的节点。因此,MIN 叶节点的一个 MAX 节点双亲的倒推值就等于叶节点的静态评估值中的最大值。另一方面,MIN 从叶节点中选取时,必须选择值最小的节点(即最负的值)。既然如此,MAX 叶节点的 MIN 双亲节点被分配一个倒推值,它等于叶节点静态评估值的最小值。在所有叶节点的父节点被赋予倒推值后,开始倒推另一层,假定 MAX 将选择有最大倒推值的 MIN 后继节点,而 MIN 会选择有最小倒推值的 MAX 后继节点。

继续逐层对节点评估,直到最后开始节点的后继者被赋予倒推值。假定 MAX 首先移动,MAX 将选择有最大倒推值的节点作为它的首步。

整个过程的有效性基于下面的假设:起始节点的后继节点的倒推值是这些位置最终相对价值的可靠量度,它比直接对这些位置使用静态估价函数求取的值更加可靠。倒推值依赖于“向前看几步”。由于倒推值在博弈树中的预先推算,并且取决于在博弈结束时发生的一些特性,这些值往往更加切合实际。

9.2.3 象棋 $\alpha - \beta$ 过程

当搜索一棵博弈树时,用户的目标只能是寻找一步好棋。在走完这一步之后,等待对手的回步,并从这个新的棋局来再次考虑下一步好棋。

刚才叙述的搜索过程将搜索树的产生和位置评估完全分开,只有在树完全产生后才能对位置评估。这种分离导致算法粗糙和效率低下。假如将叶节点的评估、计算倒推值与树的产生同时进行,就可能大量减少所需搜索的数目(有时非常大)。

象棋博弈是敌我双方的智能活动,任何一方不能单独控制博弈过程,而是双方轮流实施其控制对弈的过程,是博弈树的一种特殊的“与/或”树。其中,不同级别(深度)的结点,分别交替属于敌我双方,在博弈树生成过程中,由敌我双方轮流进行扩展,新生成的子结点是双方交替出现的。

“或”结点的 α 值,是指其当前子结点中的最大倒推估值,对应于轮到棋手走时的节点值,它是博弈树的下界。

“与”结点的 β 值, 是指其当前子结点中的最小倒推估值, 对应于轮到对手走时的节点值, 它是博弈树的上界。

若“或”结点的 α 值, 不能使其父结点的 β 值降低, 则该“或”结点以下的分枝树可剪除(停止搜索, 取博弈树的值为 α), 称为 β 剪枝。

若“与”结点的 β 值, 不能使其父结点的 α 值升高, 则该“与”结点以下的分枝树可剪除(停止搜索, 取博弈树的值为 β), 称为 α 剪枝。

如图 9.1 所示的博弈树, 从根节点 A 点开始向左搜索, 到点 B 后再到 C 是终止节点, 假定计算出它的局面评价函数值得 48, 返回 B 又到 D, 属终止节点, 假定计算它的局面评价函数值得 128, 在 48 与 128 之间取较小者 48, 经过比较求出 B 点的极小值为 48, 而退到 A 是求极大值的, 虽然不知道 E 的分支计算结果如何, 但可以肯定 A 的期望起码是 48 或写作 ≥ 48 , 因为 E 分支如算出来的值小于 48, 则 A 点求极大值得 48, 如 E 分支算出来的值大于 48, 则 A 求极大值为 >48 。

现在(如图 9.2 所示)根节点 A 点向右边搜索到 E, 再向前到 F, 是终止节点, 假定计算出它的局面评价函数值得 32, 由于 E 点是求极小值, 故无论 G 节点的局面评价函数值如何, E 的期望值最多是 32, 或写成 ≤ 32 , 再退回 A 点求极大值时, 在 $B=48$ 或 $E \leq 32$ 之间选择较大值, 确定 A 的期望值为 48, 也就是说, 应该选择 $A \rightarrow B$ 的着法为最优着法, 然后 $B \rightarrow C$ 最终会到达 C 点。这样一来, 可以剪去 EG 棋路分支, 并略去 C 节点局面评价函数的计算, 同样能达到搜索选优的目标。

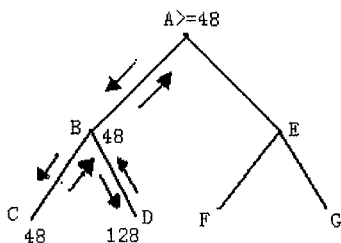


图 9.1 从左分支求 A 点的极大值

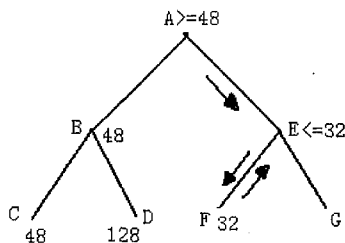


图 9.2 从左右分支求 A 点的极大值

图 9.3 所示的是单兵杀将棋局, 图 9.4 所示绘出红先两种着法及黑应着棋路树, 其他如动帅的变化删去。

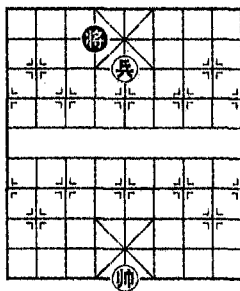


图 9.3 单兵杀将

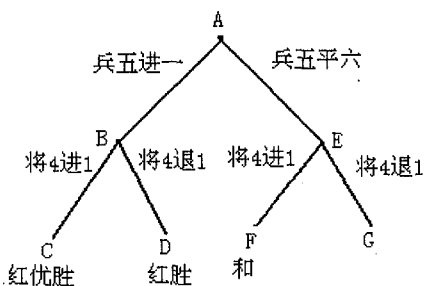


图 9.4 红先两种着法棋路树

假设计算机先搜索兵五进一这步棋,黑方只有将4进1与将4退1两种应着,假定黑方选择对红方最不利的着法,也就是说决定将4进1,局面属红胜。

接着计算机回退到A点另搜索兵五平六这步棋,黑方应着只有将4进1与将4退1两种,当搜索到F时成和棋,就不用再往G点搜索了,这表示黑方起码能求和。

对于A点来说,如走兵五进一可取胜,而走兵五平六可能成和,显然红方应选择最有利的着法兵五进一。这样就略去兵五平六时黑方将4退1的分析。

上述剪枝法,棋路树原点(深度为零)是求极大值者,第2层节点(深度为1)是求极小值者,剪掉第3层的一些节点,称为 α 剪枝。如果棋路树还有第4层节点才是终止节点,那么按剪枝原理剪掉第4层的一些节点,称为 β 剪枝。在一个庞大的棋路树中进行剪枝,通常既包括 α 剪枝又包括 β 剪枝,故称为 $\alpha-\beta$ 剪枝。从理论上讲,剪枝的机会越多,则搜索的效率就越高,在同样的时间内搜索的深度越大,计算机的棋力就相应提高。

$\alpha-\beta$ 剪枝法,是在博弈树生成过程中,边生成,边估算,剪除没有搜索价值的棋路树分枝,提高了搜索效率,减少了搜索空间。不过,如果评价函数设计不当,仍会有一些好的节点被遗漏掉。

9.3 中国象棋简论

中国象棋是中华民族的瑰宝,也是传统的琴、棋、书、画四种艺术之一。它不仅历史悠久,源远流长,更以其集文化、科学、艺术、竞技为一身的特点而受到亿万人的喜爱。列宁说:“象棋是智慧的体操”。歌德说:“象棋是人类智慧的试金石。”一些棋手说:“象棋是生命”,“象棋是生活”……所有这些给人的印象使从科学的角度来分析中国象棋,显得苍白无力。究其原因,是子与势的关系难以理清。

9.3.1 中国象棋的难点——势

据《中国象棋词典》载,势是指双方棋子的分布和对立之势的综合状态。此句的描述复杂性很低,但是中国象棋的势分析起来却很有深度:只有人们分析了它的足够多的部分,才能够发现它有意义的规律性,掌握它的秘密。势表面上是随机的,但内在是冗余的。实际上很简单,但掩盖在复杂的计算中。即使把势进行分类,由于每种对象类都有着丰富的相对随机的对象,这样的对象不可有效地区分。因此,势具有不可压缩性。

象棋对局过程中某个棋子的意义取决于棋局形势,即该棋子同其他棋子的相互作用。因此除开棋子的“名义”价值外,还形成了一种由棋局形势所决定的价值,这两种价值的表现形式是不同的。因此,计算的全部复杂性就在于“棋局价值”只能通过棋子的总和的相互作用来表现,相互之间的作用特别复杂。

如果想在象棋的某一局面中用兵的价值表现车和马的价值的话,就必须考虑它们之间的相互影响,换句话说,棋子的价值意义组成一个复杂的系统。

全局对弈,变化繁多。开局红方第一着有40种变化,黑方应着大体如此变化,则第1回合之后出现可能局面数约为 $40 \times 40 = 40^2 = 1600$

一局棋平均 35 回合即 70 步, 每步可能走法平均 40 种, 则一局棋的可能变化有: $40^1 + 40^2 + 40^3 + \dots + 40^{70} \approx 5.5^{113}$ 种

在实战对局过程中, 势不只是取决于棋局的视觉画面, 同时还伴有对典型局面的评估, 它通过归纳, 寻求诸子力之间的联系方式以及可能的着法。势不仅具有客观性, 还具有主观性、思辨性、概念性。势除了棋子本身之间的客观复杂性外, 还和棋手自身的经验、所掌握的理论知识、注意力、美学观点、思维的逻辑性、年龄等也有千丝万缕的关系。有经验的棋手对局面的作出估价, 不是根据明显表露的知识, 而是根据关于某种局面特点的隐蔽和非言语知识。在时间恐慌中, 棋手往往产生虚幻的势, 倾向于得出直露、肤浅、自然或相对稳定的简化判断。正因为此, 古往今来, 产生了大量的棋谱, 这些都是极富价值的经验总结, 值得深入分析研究。

9.3.2 势是多项式

根据序列的统计性质可知, 即使最随机的序列也具有对数阶的非常规则的子序列。实际上, 人们在努力寻找具有强大预测能力和很多应用的科学定律, 也取得了惊人的成功。这些事实, 即使没有直接证明, 也间接表明了宇宙的内在有序性。

科学可被看作数据压缩的艺术。当把大量的经验数据压缩为短的自然定律的形式后, 人们就具有很强的预测能力。

科学是否存在, 似乎依赖于大量的经验数据能否形成可压缩的序列。如果不能对中国象棋中的势作出有效的概括, 就不能说形成了象棋科学。

关于科学理论体系及其特点, 有必要考察一下爱因斯坦的有关论述。爱因斯坦作为 20 世纪最伟大的科学家, 不仅对丰富多彩的现象世界始终充满着童心般的好奇, 而且处处都在努力“寻求一个能把观察到的事实联结在一起的思想体系”, 爱因斯坦将严格意义上科学理论体系划分为 3 个不同的层次: 首先是位于最底层的基础性层次, 即该理论所涉及范围内的事实材料, 因为这些事实材料是构成理论体系的经验基础部分; 然后是一些基本的公理和假设, 它们是在一定的事实材料的启示之下“发明出来的”, 同时又是构成整个理论大厦的逻辑基础; 最后第 3 个层次是具体推论, 它是在基本公理假设的基础上通过推理而得出的具体结论, 这些具体结论是否有效与可靠可通过与事实材料相比较而得到验证。

在理论体系的上述 3 个层次中, 处于核心地位的是“基本公理或假设”。在爱因斯坦看来, 这些基本公理或假设要具有科学性, 就必须能够满足两个方面的基本要求。其一, 它必须“具有最大可能的简单性”, 即“所包含的彼此独立的假设或公理最少”; 其二, 由这些公理所推导出来的具体结论必须能够尽可能完备地解释大量的感觉经验事实。总而言之, “科学的目的, 一方面是通过最少个数的原始概念和原始关系的使用来达到这个目的”。

作者无意建立中国象棋的科学理论体系, 那不是作者企求的。作者只是针对其中专题之一——即势——提出一己之见。

如果采用数学中的常用记法, 则表示如下:

势 = 战略 + 战术 + 其他

战略 = 子力 + 攻击性 + 棋子的位置 + 棋子的活动性 + 协调性 + 保护性

战术 = 捉双 + 牵制 + 闪击 + 拦截 + 抽子 + 腾挪 + 兑子 + 诱离 + 封锁 + 过载 + 弃子 + 运子

其他 = 特殊的特征组合 + 特殊的模式 + 知识

以下分而述之：

- 子力：是衡量局面优劣的首要因素。一般而言，子力多者占优，子力少者为劣。车的战斗力最强，相当于双炮。炮与马的价值大体相等，相当于双相。相与士的价值大体相等，相当于双兵。将(或帅)的战斗力不大，但价值最大。因为象棋以杀死对方的将(或帅)作为取胜的标志，将(或帅)的价值大于其余所有子力价值之和。
- 攻击性：是棋子在该位置对对方阵地(特别是对对方将(或帅)所在区域)的威胁程度。如果移到该位置的棋子攻击对方的棋子的数目增加的话，就给该棋步加分。
- 棋子的位置：每个棋子在棋盘上不同位置，其价值是不同的，开局阶段要考虑棋子在该位置对阵形的维护作用，从它的初始位置移出参战，对这样的棋步要给分。中局则更多地考虑棋子在该位置对对方子力、将(或帅)的威胁作用。移到对方一侧的棋步要给分，相反，对于后撤这些棋子的棋步要扣分。例如车塞象眼，炮镇当头，马跳卧槽，兵入九宫，都对对方的将(或帅)有威胁作用，要加分。

有道是“三着不出车，棋在家里输”，鼓励开车。中炮局时鼓励先开直车，如不便才开横车，士相局最好开贴身车。直车开出后，升至巡河，过河卒林，过河象眼等，均应有不同的鼓励分。

马在原位，应扣分。跳正马，边马，盘河马，外肋马，卧槽马，挂角马，钓鱼马，均应有不同的鼓励分。

有道是“炮无虚发”。摆当头炮，七路炮，士角炮都是较好的位置，另外摆边炮也不错。过河炮以打中卒较好，另外沉底炮也是好位置。

过河兵的价值明显上升，靠近及进入九宫就更加升值，但到底老兵减值。

由于棋盘位置是对称的，棋子的位置值也是左右对称的。

车、马、炮、兵的棋子位置值如表 9.1~ 表 9.4 所示(其中行、列数均从|开始以符合习惯)。

表 9.1 车的位置值表

行 \ 列	第 1 列	第 2 列	第 3 列	第 4 列	第 5 列	第 6 列	第 7 列	第 8 列	第 9 列
第 1 行	-26	128	76	180	154	180	76	128	-26
第 2 行	102	52	102	204	102	204	102	52	102
第 3 行	52	102	76	180	154	180	76	102	52
第 4 行	76	128	102	180	180	180	102	128	76
第 5 行	154	204	180	256	256	256	180	204	154
第 6 行	154	180	154	230	230	230	154	180	154
第 7 行	154	230	204	282	282	282	204	230	154
第 8 行	154	154	154	230	230	230	154	154	154
第 9 行	204	256	230	308	332	308	230	256	204
第 10 行	180	180	154	230	204	230	154	180	180

表 9.2 马的位置值表

列 行	第 1 列	第 2 列	第 3 列	第 4 列	第 5 列	第 6 列	第 7 列	第 8 列	第 9 列
第 1 行	0	-52	0	0	0	0	0	-52	0
第 2 行	0	26	52	52	-26	52	52	26	0
第 3 行	52	26	102	102	52	102	102	26	52
第 4 行	26	76	102	76	128	76	102	76	26
第 5 行	52	154	204	180	154	180	204	154	52
第 6 行	76	204	180	230	204	230	180	204	76
第 7 行	102	308	230	308	256	308	230	308	102
第 8 行	154	180	204	256	230	256	204	180	154
第 9 行	52	128	358	204	102	204	358	128	52
第 10 行	52	102	204	154	52	154	204	102	52

表 9.3 炮的位置值表

列 行	第 1 列	第 2 列	第 3 列	第 4 列	第 5 列	第 6 列	第 7 列	第 8 列	第 9 列
第 1 行	0	0	26	76	76	76	26	0	0
第 2 行	0	26	52	76	76	76	52	26	0
第 3 行	52	0	102	76	128	76	102	0	52
第 4 行	0	0	0	26	52	26	0	0	0
第 5 行	-26	0	52	26	76	26	52	0	-26
第 6 行	0	0	0	26	102	26	0	0	0
第 7 行	0	0	-26	52	128	52	-26	0	0
第 8 行	26	26	0	-128	-102	-128	0	26	26
第 9 行	26	26	0	-52	-180	-52	0	26	26
第 10 行	76	52	0	-128	-154	-128	0	52	76

表 9.4 兵的位置值表

列 行	第 1 列	第 2 列	第 3 列	第 4 列	第 5 列	第 6 列	第 7 列	第 8 列	第 9 列
第 1 行	0	0	0	0	0	0	0	0	0
第 2 行	0	0	0	0	0	0	0	0	0
第 3 行	0	0	0	0	0	0	0	0	0
第 4 行	0	0	-26	0	52	0	-26	0	0
第 5 行	26	0	102	0	102	0	102	0	26
第 6 行	76	154	230	230	256	230	230	154	76
第 7 行	128	256	384	435	512	435	384	256	128
第 8 行	180	332	538	768	1024	768	538	332	180
第 9 行	230	460	716	1024	1536	1024	716	460	230
第 10 行	0	38	76	115	154	115	76	38	0

- 棋子的活动性：是棋子在该位置向各处调动的可能性。棋子的威力能否充分发挥，与它的活动性直接相关。例如车在纵横线上遇到障碍物，马被周围棋子绊脚等，都降低了活动性，也降低了威力的发挥。活动性的计算是把棋子在棋盘上所能移动到达的位置数作为活动性，给予评分，而能移动到达对方九宫内位置者，评分稍高。

棋子活动性 = 实际可移动步数(A_0) / 最大可移动步数(A)

这样 $A_0(\text{车}) = 17$, $A_0(\text{马}) = 8$, $A_0(\text{炮}) = 17$ 。

根据经验，车马炮之外的棋子活动性与否对棋局优劣影响不大，其活动性忽略不计。

- 协调性：是棋子在该位置与己方其他棋子联系紧密的程度。如果棋子在该位置导致数个棋子(同兵种或不同兵种)有目的的配合作战，它从属于总的行动计划，服务于特定的作战目的，那么这样的棋步要给分。
- 保护性：如果棋子下一步会被吃掉，这样的棋步要扣分。
- 捉双：同时对对方造成两个或两个以上的威胁，即用一个棋子同时攻击对方两个或两个以上的目标。常见的捉双形式有：
 - ◆ 用一个棋子同时攻击对方两个大子或兵；
 - ◆ 将军对方的将(或帅)，同时攻击对方的一个棋子；
 - ◆ 威胁将杀对方的将(或帅)，同时又攻击对方的一个棋子；
 - ◆ 同时攻击对方的一个棋子和一个要害据点；
 - ◆ 同时攻击对方的两个或两个以上的要害据点。

产生捉双的基础条件：

- ◆ 对方子力无防御或缺乏足够的防御；
- ◆ 对方子力部署在不成功的位置阵营中。

捉双往往带有突然性和强制性，从而使对方防不胜防，是赢得物质优势的有效手段，在多项式中应为正数。

- 牵制：进攻者用一个子(如炮、车、帅、马、象)，在某一线路上攻击对方的将(或帅)、大子、兵或重要据点，对方采用其他的棋子来遮盖这些被攻击的目标。

牵制战术有效地限制了对方的子力活动，甚至使它们无法行动，为整个全局战略的进攻创造战机，在多项式中应为正数。

- 闪击：是由己方两个棋子相互配合同时打击对方两个目标。当进攻一方的远射程大子瞄住的目标，中间还有一个己方的棋子进时，使得进攻更加隐蔽。中间棋子突然移动，寻求新的打击目标，与后面的大子密切配合，共同完成对对方造成的双重打击。使对方顾此失彼，两处必损其一。在多项式中应为正数。
- 拦截：用穿插的手段分割对方两子间的联系，截断对方某子的势力控制线，以及在对方兵力和重要通道上制造障碍等。其主要目的是破坏对方子力的协调性。由于其具有助攻性的特点，不仅使自己的子力最大限度地发挥进攻作用，而且破坏性极大，使对方阵营中关键的弱点暴露出来，以便实施最终的打击。在多项式中应为正数。
- 抽子：指“一面叫将，一面吃子或要吃子”的一类着法。常以歼灭对方有生力量为目标，其运子迅速巧妙，攻击力强。在多项式中应为正数。

- 腾挪：将妨碍进攻的大子或兵腾出位置、让出空间。快速地腾挪，不给对方以喘息的机会，来改善防御的结构，往往能得先、得子而造成优势。在多项式中应为正数。
- 兑子：就是交换子力。如能抓住时机适当地运用兑子，则往往可以通过双方短兵相接式的子力互换，而达到争先抢攻或占位取势，甚至通过轻重兑换而获取子力优势。
- 诱离：当局部地区获得优势的一方，在展开进攻的过程中，往往由于对方顽强的抵抗而受阻。特别是在攻守双方投入战斗的子力数目相等时，就出现了暂时的相持状态。为了打开僵局，扩大攻势，常常采用诱离战术。

诱离战术能够有效地削减对方防御兵力，使局部地区战斗保持优势，为实施最终的打击奠定基础。诱离战术引诱对方阵营中关键的大子或兵，离开防御的重要格子、线路。使整个防线出现裂痕，暴露出新的弱点，为进攻一方敞开道路。在多项式中应为正数。

- 封锁：是限制对方子力活动，破坏对方子力协调性、灵活性的一种常用战术手段。封锁对方将(或帅)的出路，强制对方大子或兵占据某些不重要的格子，减少将(或帅)的活动空间，为实施最后的打击创造战机。由于其具有助攻性质，确保主战场上战斗顺利进行，常常能迅速取得成效——获得重大物质优势，或者导致直接将杀对方的将(或帅)。在多项式中应为正数。
- 过载：参加战斗的棋子有时担当的防御任务，需要同时完成两个或两个以上的功能，那么这个棋子就被认为过载。具体表现为：同时保护两个或两个以上的大子或兵；同时保护两个或两个上的重要格子；同时保护一个大子或兵和一个重要格子。由于过载现象经不起战斗的考验，不能担负起战斗所赋予的使命，对方就有可能对它所保护的目标实施打击。在多项式中应为负数。
- 弃子：通过舍弃某个棋子，强制性地达到预定目的，使局面发生戏剧性的变化。
- 运子：当局势处于战略进攻或战略防御的关键时刻，从整体上看局面尚存在致命的弱点，这时就需要及时地重新改善子力的部署，为其他的战术创造局面条件。由于其隐蔽得很深，人可以利用经验、知识、直觉细心挖掘，而在计算机中却比较难定义。
- 知识：将象棋游戏的理论知识和实践知识输入电子计算机，编制相应的函数来描写象棋局面，能提高象棋程序的弈棋水平。

引用“开局指南”能够提高计算机弈棋的等级或至少能缩短计算机“思考”开局的几步棋的时间。

9.4 程序编制

本程序编译成执行文件后，可运行于 Windows 95/98 之上。其功能是既可以进行全局对弈，也可以摆成任意中局或残局，既可以观看机器演示对弈，也可以由机器给出提示，既可以读入棋谱，也可以根据局面评价函数判断形势而走子。总之，操作界面十分友好，功能十分强大。

9.4.1 中国象棋的表达

1. 棋盘与棋规

中国象棋棋盘、棋规如表 9.5 所示。

表 9.5 中国象棋的棋盘和棋规

棋盘/子	棋规说明
棋盘	10 行 9 列
将	仅限于九宫内，且每次只能直走一格
士	仅限于九宫内，且每次只能斜走一格
象	仅限于河界一边，且每次只能走田字格，如果田心有子占住，象不能飞动
马	整个棋盘均可走动，但每次只能走日字格，如果马脚被绊住，马不能走动
车	整个棋盘均可走动，纵横均可，且可连续走动
炮	整个棋盘均可走动，纵横均可，且可连续走动，但吃子时须有隔子
兵	未过河时，只能向前走一格，过河后，每次只能走一格，但不能后退
其他	连将两次要变招。60 回合为自然限招

2. 棋子的表示

棋子的表示如表 9.6 所示。

表 9.6 棋子的表示

棋子名称	计算机方	对手方
将	 BKING	 RKING
士	 BASSIST	 RASSIST
象	 BBISHOP	 RBISHOP
马	 BKKNIGHT	 RKKNIGHT
车	 BROOK	 RROOK
炮	 BGUNNER	 RGUNNER
兵	 BPAWN	 RPAWN

3. 棋盘的表示

中国象棋棋盘(如图 9.5 所示)有 10 行 9 列，采用十六进制能较好地表示。十六进制数右移 4 位即得行数，对 16 取模即得列数。

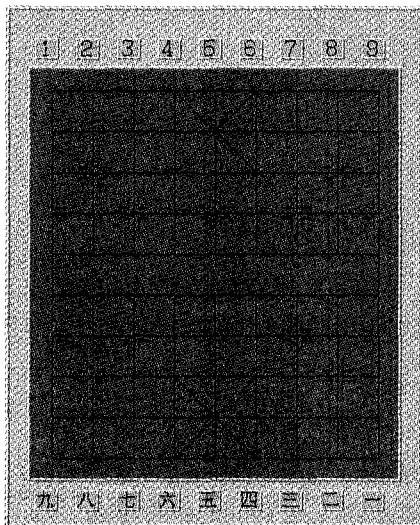


图 9.5 中国象棋棋盘

4. 棋步的表示

下棋的程序必须知道所有棋子的合法走步。由于棋子的位置采用十六进制表示，加减一个数字代表了一个棋步走向，所以加上 1 就等于把棋子向右移动一步，减去 1 就等于把棋子向左移动一步，加上 0X10 就等于把棋子向上移动一步，减去 0X10 就等于把棋子向下移动一步。加上 0X11 就等于把棋子向右上移动一步。等等。

9.4.2 中国象棋规则的实现

根据以上原则编制出中国象棋规则实现代码，它们存在于 CHESS 项目中的 GlobalFun.cpp 文件中，属于自定义全局函数，所有的可视化类函数均可调用。具体制作过程如下：

- (1) 选择 File | Open Project 命令，打开 Open Project 对话框，利用该对话框打开 CHESS 项目；
- (2) 选择 View | Project Manager 命令，打开 Project Manager 对话框，在该对话框中查找源文件双击 GlobalFun.cpp 打开它；
- (3) 在打开的文件中手工添加各种函数。

在 GlobalFun.cpp 文件中手工添加的实现中国象棋规则的自定义全局函数如下所示：

1. 函数名称：void CalcAttackTable(void)

说明： 计算攻击表。对于每个棋子通过合法步数所能到达的位置，不管是对方的棋子还是自己的棋子，还是没有棋子，则所能到的位置为受攻击的对象。马走日字，此时不考虑蹩脚；象走田字，不考虑象心有无棋子；炮纵横任意走

数格, 不管有无炮架; 士每次斜走一格, 将或纵或横走一格, 此时不一定要在九宫; 车纵横任意走数格, 不管有无棋子挡住。

函数定义如下:

```
void CalcAttackTable(void)
{
    DIRTYPE dir;
    int sq;
    unsigned char i;

    for (sq = -0x99; sq <= 0x99; sq++)
    {
        AttackTable[sq].pieceset = 0;
        AttackTable[sq].direction = 0;
    }

    for (dir = 3; dir >= 0; dir--)
    {
        for (i = 1; i < 10; i++)
        {
            AttackTable[DirectionTab[dir]*i].pieceset = SETOFPIECE(BitTab[rook] +
            BitTab[gunner] + BitTab[king]) ;
            AttackTable[DirectionTab[dir]*i].direction = DirectionTab[dir];
        }
    }

    for (dir = 3; dir >= 0; dir--)
    {
        AttackTable[AssistDir[dir]].pieceset = SETOFPIECE(BitTab[assist]);
        AttackTable[AssistDir[dir]].direction = AssistDir[dir];
    }

    for (dir = 3; dir >= 0; dir--)
    {
        AttackTable[BishopDir[dir]].pieceset = SETOFPIECE(BitTab[bishop]);
        AttackTable[BishopDir[dir]].direction = BishopDir[dir];
    }

    for (dir = 7; dir >= 0; dir--)
    {
        AttackTable[KnightDir[dir]].pieceset = SETOFPIECE(BitTab[knight]);
        AttackTable[KnightDir[dir]].direction = KnightDir[dir];
    }
}
```

```

    }

    for (dir = 3; dir >= 0; dir--)
    {
        AttackTable[DirectionTab[dir]].pieceset = SETOFPIECE(BitTab[pawn] +
        BitTab[rook] + BitTab[gunner] + BitTab[king]) ;
        AttackTable[DirectionTab[dir]].direction = DirectionTab[dir] ;
    }
}

```

2. 函数名称: short PieceAttacks(PIECETYPE apiece, COLORTYPE acolor, SQUARE TYPE asquare, SQUARETYPE square)

说明: 计算方格中的棋子是否攻击另外某个方格。

函数定义如下:

```

short PieceAttacks(PIECETYPE apiece, COLORTYPE acolor,
                  SQUARETYPE asquare, SQUARETYPE square)
{
    EDGESQUARETYPE sq;
    int x;
    int tmpx;
    int guncap = 0; //记录棋子的移动路线上是否有别的棋子

    x = square - asquare;
    // if (apiece == pawn) /* 兵的攻击性。包括过河兵和未过河兵 */
    /* 如果别的棋子能移到该格子, 则该棋格将受到攻击 */

    if (AttackTable[x].pieceset & BitTab[apiece])
    {
        if ( apiece == pawn )
            if (( asquare >= 0x30) && ( asquare <= 0x98))
                if (ChessBoard[asquare].color == red)
                    if ( x == 0x10)
                        return 1;

        if ( apiece == pawn )
            if (( asquare >= 0x50) && ( asquare <= 0x98))
                if (ChessBoard[asquare].color == red)
                    if ((abs(x)==1) || ( x == 0x10))
                        return 1;
    }
}

```



```

if ( apiece == pawn )
    if (( asquare <= 0x68) && ( asquare <= 0x98))
if (ChessBoard[asquare].color == black)
    if ( x == -0x10)
return 1;

if ( apiece == pawn )
    if (( asquare <= 0x48) && ( asquare <= 0x98))
if (ChessBoard[asquare].color == black)
    if ((abs(x)==1) || ( x == -0x10))
return 1;
/*将只对位于九宫内的棋子构成威胁*/
if ( apiece == king )
    if (( asquare >=0) && ( asquare <= 0x28) && ((asquare & 0xf) >=3)
        && ((asquare & 0xf) <=5))
if (ChessBoard[asquare].color == red)
    if (( square >=0) && ( square <= 0x28) && ((square % 16) >=3)
        && ((square & 0xf) <=5))
    if ((abs(x)==1) || (abs(x)==0x10))
        return 1;
    if ( apiece == king )
if (( asquare >= 0x70) && ( asquare <= 0x98) && ((asquare % 16) >=3)
    && ((asquare % 16) <=5))
if (ChessBoard[asquare].color == black)
    if (( square >= 0x70) && ( square <= 0x98) && ((square % 16) >=3)
        && ((square % 16) <=5))
    if ((abs(x)==1) || (abs(x)==0x10))
        return 1;

if ((apiece == king ))
    if (ChessBoard[asquare].color == red)
    if ((square >= 0x70) && (square <= 0x98))
        if (((square % 16) >=3) && ((square % 16) <=5))
            if ((ChessBoard[square].piece == king ))
                if ((ChessBoard[square].color == black ))
                    if (x==0x50)
{
/* 中间有棋子阻挡吗? 因为两个将不能照面 */

```

```

        sq = asquare;
        guncap = 0;
        do
        {
            sq += 0x10;
            if (ChessBoard[sq].piece != empty )
                guncap =guncap + 1;
        } while ( sq != square );
        return (((guncap-1)==0) && (sq == square)) ;
    }

    if ((apiece == king ))
    if (ChessBoard[asquare].color == red)
        if ((square >= 0x70) && (square <= 0x98))
            if (((square % 16) >=3) && ((square % 16) <=5))
                if ((ChessBoard[square].piece == king ))
                    if ((ChessBoard[square].color == black ))
                        if (x==0x60)
    {
        /* 中间有棋子阻挡吗? 因为两个将不能照面 */
        sq = asquare;
        guncap = 0;
        do
        {
            sq += 0x10;
            if (ChessBoard[sq].piece != empty )
                guncap =guncap + 1;
        } while ( sq != square );
        return (((guncap-1)==0) && (sq == square)) ;
    }

    if ((apiece == king ))
    if (ChessBoard[asquare].color == red)
        if ((square >= 0x70) && (square <= 0x98))
            if (((square % 16) >=3) && ((square % 16) <=5))
                if ((ChessBoard[square].piece == king ))
                    if ((ChessBoard[square].color == black ))
                        if (x==0x70)
    {

```

```
/* 中间有棋子阻挡吗? 因为两个将不能照面 */
sq = asquare;
guncap = 0;
do
{
sq += 0x10;
if (ChessBoard[sq].piece != empty )
guncap =guncap + 1;
} while ( sq != square );
return (((guncap-1)==0) && (sq == square)) ;
}

if ((apiece == king ))
if (ChessBoard[asquare].color == red)
if ((square >= 0x70) && (square <= 0x98))
if (((square % 16) >=3) && ((square % 16) <=5))
if ((ChessBoard[square].piece == king ))
if ((ChessBoard[square].color == black ))
if (x==0x80)
{
/* 中间有棋子阻挡吗? 因为两个将不能照面 */
sq = asquare;
guncap = 0;
do
{
sq += 0x10;
if (ChessBoard[sq].piece != empty )
guncap =guncap + 1;
} while ( sq != square );
return (((guncap-1)==0) && (sq == square)) ;
}

if ((apiece == king ))
if (ChessBoard[asquare].color == red)
if ((square >= 0x70) && (square <= 0x98))
if (((square % 16) >=3) && ((square % 16) <=5))
if ((ChessBoard[square].piece == king ))
if ((ChessBoard[square].color == black ))
if (x==0x90)
{
```

[illegible]

```
{
/* 中间有棋子阻挡吗? 因为两个将不能照面 */
sq = asquare;
guncap = 0;
do
{
sq -= 0x10;
if (ChessBoard[sq].piece != empty )
guncap =guncap + 1;
} while ( sq != square);
return (((guncap-1)==0) && (sq == square)) ;
}

if (apiece == king )
if (ChessBoard[asquare].color == black)
if ((square >= 0) && (square <= 0x28))
if (((square % 16) >=3) && ((square % 16) <=5))
if (ChessBoard[square].piece == king )
if ((ChessBoard[square].color == red ))
if (x==0x70)
{
/* 中间有棋子阻挡吗? 因为两个将不能照面 */
sq = asquare;
guncap = 0;
do
{
sq -= 0x10;
if (ChessBoard[sq].piece != empty )
guncap =guncap + 1;
} while ( sq != square );
return (((guncap-1)==0) && (sq == square)) ;
}

if (apiece == king )
if (ChessBoard[asquare].color == black)
if ((square >= 0) && (square <= 0x28))
if (((square % 16) >=3) && ((square % 16) <=5))
if (ChessBoard[square].piece == king )
if ((ChessBoard[square].color == red ))
```

```

        if (x==0x80)
        {
/* 中间有棋子阻挡吗? 因为两个将不能照面 */
sq = asquare;
guncap = 0;
do
{
sq -= 0x10;
if (ChessBoard[sq].piece != empty )
guncap =guncap + 1;
} while ( sq != square );
return (((guncap-1)==0) && (sq == square)) ;
}

/* 中间有棋子阻挡吗? 因为两个将不能照面 */
if (apiece == king )
if (ChessBoard[asquare].color == black)
if ((square >= 0) && (square <= 0x28))
if (((square % 16) >=3) && ((square % 16) <=5))
if (ChessBoard[square].piece == king )
if ((ChessBoard[square].color == red ))
if (x==0x90)
{
/* 中间有棋子阻挡吗? 因为两个将不能照面 */
sq = asquare;
guncap = 0;
do
{
sq -= 0x10;
if (ChessBoard[sq].piece != empty )
guncap =guncap + 1;
} while ( sq != square );
return (((guncap-1)==0) && (sq == square)) ;
}

/*士只对位于九宫内的棋子构成威胁*/
if ( apiece == assist )
if (ChessBoard[asquare].color == red)
if (( square >=0) && ( square <= 0x28) && ((square % 16) >=3)
&& ((square % 16) <=5))

```

```
        return 1;
    if ( apiece == assist )
        if (ChessBoard[asquare].color == black)
            if (( square >= 0x70) && ( square <= 0x98) && ((square % 16) >=3)
                && ((square % 16) <=5))
                return 1;
    /*如果田心没有棋子，象将对位于其对角的棋子构成威胁*/
    if ( apiece == bishop )
    {
        if (x== 0x1E)
            tmpx = 0x0F;
        if (x== -0x1E)
            tmpx= -0x0F;
        if (x== 0x22)
            tmpx= 0x11;
        if (x== -0x22)
            tmpx= -0x11;
        if (( ChessBoard[square - tmpx].piece == empty ))
            if (ChessBoard[asquare].color == red)
                if (( square >=0) && ( square <= 0x48))
                    return 1;
        if (x== 0x1E)
            tmpx = 0x0F;
        if (x== -0x1E)
            tmpx= -0x0F;
        if (x== 0x22)
            tmpx= 0x11;
        if (x== -0x22)
            tmpx= -0x11;
        if (( ChessBoard[square - tmpx].piece == empty ))
            if (ChessBoard[asquare].color == black)
                if (( square >= 0x50) && ( square <= 0x98))
                    return 1;
    }

    /*如果马没有绊脚，将对位于其对角的棋子构成威胁*/
    if ( apiece == knight )
    {
        if (x == 0x0E)
```

```

        tmpx= 0x0F;
    if ( x== -0x0E)
        tmpx= -0x0F;
    if ( x== 0x12)
        tmpx = 0x11;
    if ( x== -0x12)
        tmpx = -0x11;
    if ( x== 0x1F)
        tmpx = 0x0F;
    if (x == -0x1F)
        tmpx = -0x0F;
    if ( x== 0x21)
        tmpx = 0x11;
    if ( x== -0x21)
        tmpx = -0x11;
    if ( ChessBoard[square - tmpx].piece == empty )
        return 1;
    }

/*如果直线上没有别的棋子，车将对位于其射程内的棋子构成威胁*/
if (apiece == rook )
{
    /* 直线上有没有别的棋子? */
    sq = asquare;
    do
    {
        sq += AttackTable[x].direction;
        sq = sq;

    } while (sq != square && ChessBoard[sq].piece == empty );
    return ((sq == square) &&
        ((asquare >> 4) == (square >> 4) || (asquare % 16) == (square %
        16)));
}

/*如果直线上有炮架子，炮将对位于其射程内的棋子构成威胁*/
if (apiece == gunner )
{

```



```

    /* 直线上有没有别的棋子? */
    sq = asquare;
    guncap = 0;
    do
    {
        sq += AttackTable[x].direction;
        sq = sq;
        if (ChessBoard[sq].piece != empty )
            guncap =guncap + 1;
    } while (sq != square );
    return (((sq == square) && ((guncap-1)==1)) &&
        ((asquare >> 4) == (square >> 4) || (asquare % 16) == (square %
        16))) ;
    }
}
return 0;
}

```

3. 函数名称: short Attacks(COLORTYPE acolor, SQUARETYPE square)

说明: 计算 acolor 方是否攻击 square。

函数定义如下:

```

short Attacks(COLORTYPE acolor, SQUARETYPE square)
{
    INDEXTYPE i;

    for (i = PawnNumber[acolor]; i >= 0; i--)
        if ((PieceTable[acolor][i].ipiece != empty))
            if (PieceAttacks(PieceTable[acolor][i].ipiece, acolor,
                PieceTable[acolor][i].isquare, square))
                return 1;
    return 0;
}

```

4. 函数名称: inline short RepeatMove(MOVETYPE *move)

说明: 检查移动过的棋子有没有被吃过。

函数定义如下:

```

inline short RepeatMove(MOVETYPE *move)
{

```

```

    return (move->movpiece != empty && move->content == empty );
}
/*****

```

5. 函数名称: SIXTYTYPE SixtyMoveCnt(void)

说明: 60 回合中没有吃子, 则算为平局。

函数定义如下:

```

SIXTYTYPE SixtyMoveCnt(void)
{
    SIXTYTYPE cnt = 0;

    while (RepeatMove(&MoveTable[Depth - cnt]))
        cnt++;
    return cnt;
}

```

6. 函数名称: REPEATTYPE Repetition(short immediate)

说明: 计算在没有发生和棋前计算棋格被占据的次数。movetab[back..Depth] 包含先前的移动。当 immediate 设置时, 只有接连重复才算。连续 3 次局面重复算是和棋

函数定义如下:

```

REPEATTYPE Repetition(short immediate)
{
    DEPTHTYPE lastdep, compdep, tracedep, checkdep, samedepth;
    SQUARETYPE tracesq, checksq;
    REPEATTYPE repeatcount;

    repeatcount = 1;
    lastdep = samedepth = DEPTHTYPE(Depth + 1); /* 当前位置的深度 */
    compdep = DEPTHTYPE(samedepth - 4); /* 与第一次棋子位置进行比较 */

    /*MoveTable[lastdep..Depth]包含了先前相关的移动*/
    while (RepeatMove(&MoveTable[lastdep - 1]) && (compdep < lastdep ||
        !immediate))
    {
        lastdep--;
        if (compdep < lastdep)
            return repeatcount;
        checkdep = samedepth;
    }
}

```

```

do
{
    checkdep--;
    checksq = MoveTable[checkdep].newpos1;
    for (tracedep = DEPTHTYPE(checkdep + 2); tracedep < samedepth;
tracedep += DEPTHTYPE(2))
        if (MoveTable[tracedep].oldpos == checksq) goto TENMOV;

    /* 回溯移动, 看是否以前出现过悔棋 */
    tracedep = checkdep;
    tracesq = MoveTable[tracedep].oldpos;
    do
    {
        if (tracedep-2 < lastdep) return repeatcount;
        tracedep -= (DEPTHTYPE)2;
        /* 检查棋子先前是否移动过 */
        if (tracesq == MoveTable[tracedep].newpos1) tracesq =
            MoveTable[tracedep].oldpos;
    } while (tracesq != checksq || tracedep > compdep + 1);
    if (tracedep < compdep) /* 调整比较的深度 */
    {
        compdep = tracedep;
        if ((samedepth - compdep) % 2 == 1)
        {
            if (compdep == lastdep) return repeatcount;
            compdep--;
        }
        checkdep = samedepth;
    }
    /*
在同一深度和比较深度之间进行移动检查, 决定是否找到重复局面
*/
TENMOV : if (checkdep <= compdep)
{
    repeatcount++;
    if (compdep - 2 < lastdep) return repeatcount;
    checkdep = samedepth = compdep;
    compdep -= (DEPTHTYPE)2;
}

```

```
    } while (1);  
}
```

7. 函数名称: static void Generate()

说明: 把生成的移动保存在缓冲区。

函数定义如下:

```
static void Generate()  
{  
    BufCount++;  
    Buffer[BufCount] = Next;  
}
```

8. 函数名称: static void CapMovGen()

说明: 根据棋子表生成吃子移动。

函数定义如下:

```
static void CapMovGen()  
{  
    INDEXTYPE i;  
    Next.content = ChessBoard[Next.newpos1].piece;  
    for (i = PawnNumber[Player]; i >= 0; i--)  
        if (PieceTable[Player][i].ipiece != empty)  
            if (PieceAttacks(PieceTable[Player][i].ipiece, Player,  
                             PieceTable[Player][i].isquare, Next.newpos1))  
            {  
                Next.oldpos = PieceTable[Player][i].isquare;  
                Next.movpiece = PieceTable[Player][i].ipiece;  
                Generate();  
            }  
}
```

9. 函数名称: static void NonCapMovGen()

说明: 根据棋子原先的位置生成非吃子移动。

函数定义如下:

```
static void NonCapMovGen()  
{  
    DIRTYPE first, last, dir;  
    int direction;
```

```

EDGESQUARETYPE newsq;
int tmpBishopDir,tmpKnightDir;

Next.spe = 0;
Next.movpiece = ChessBoard[Next.oldpos].piece;
Next.content = empty;
switch (Next.movpiece)
{

```

```

/*

```

生成几个方向的非吃子移动。兵未过河时，只有一个方向；过河后，有3个方向

```

*/

```

```

case pawn :
    newsq = Next.oldpos + 0x10;
    if (((newsq)<=0x98) && (newsq >= 0 ))
        if (((newsq>>4)<10) && ((newsq % 16)<9) && ((newsq | 0xff)>=0))
            if (ChessBoard[Next.oldpos].color == red)
                if (( newsq >= 0x40) && ( newsq <= 0x98))
                    if (ChessBoard[newsq].piece == empty)
                        {
                            Next.newpos1 = newsq;
                            Generate();
                        }
    newsq = Next.oldpos + 1;
    if (((newsq)<=0x98) && (newsq >= 0 ))
        if (((newsq>>4)<10) && ((newsq % 16)<9) && ((newsq | 0xff)>=0))
            if (ChessBoard[Next.oldpos].color == red)
                if (( newsq >= 0x50) && ( newsq <= 0x98))
                    if (ChessBoard[newsq].piece == empty)
                        {
                            Next.newpos1 = newsq;
                            Generate();
                        }
    newsq = Next.oldpos - 1;
    if (((newsq)<=0x98) && (newsq >= 0 ))
        if (((newsq>>4)<10) && ((newsq % 16)<9) && ((newsq | 0xff)>=0))
            if (ChessBoard[Next.oldpos].color == red)
                if (( newsq >= 0x50) && ( newsq <= 0x98))
                    if (ChessBoard[newsq].piece == empty)
                        {

```

```

        Next.newpos1 = newsq;
        Generate();
    }

    newsq = Next.oldpos - 0x10;
    if (((newsq)<=0x98) && (newsq >= 0 ))
    if (((newsq>>4)<10) && ((newsq % 16)<9) && ((newsq | 0xff)>=0))
        if (ChessBoard[Next.oldpos].color == black)
            if (( newsq <= 0x58) && ( newsq <= 0x98))
                if (ChessBoard[newsq].piece == empty)
                {
                    Next.newpos1 = newsq;
                    Generate();
                }

    newsq = Next.oldpos + 1;
    if (((newsq)<=0x98) && (newsq >= 0 ))
    if (((newsq>>4)<10) && ((newsq % 16)<9) && ((newsq | 0xff)>=0))
        if (ChessBoard[Next.oldpos].color == black)
            if (( newsq <= 0x48) && ( newsq <= 0x98))
                if (ChessBoard[newsq].piece == empty)
                {
                    Next.newpos1 = newsq;
                    Generate();
                }

    newsq = Next.oldpos - 1;
    if (((newsq)<=0x98) && (newsq >= 0 ))
    if (((newsq>>4)<10) && ((newsq % 16)<9) && ((newsq | 0xff)>=0))
        if (ChessBoard[Next.oldpos].color == black)
            if (( newsq <= 0x48) && ( newsq <= 0x98))
                if (ChessBoard[newsq].piece == empty)
                {
                    Next.newpos1 = newsq;
                    Generate();
                }
    }
    break;
    /* 为将生成 4 个方向的非吃子移动 */
case king :
    newsq = Next.oldpos + 1;
    if (((newsq)<=0x98) && (newsq >= 0 ))

```

```

if (((newsq>>4)<10) && ((newsq % 16)<9) && ((newsq | 0xff)>=0))
    if (ChessBoard[Next.oldpos].color == red)
        if (( newsq >=0) && ( newsq <= 0x28) && ((newsq % 16) >=3)
            && ((newsq % 16) <=5))
            if (ChessBoard[newsq].piece == empty)
                {
                    Next.newpos1 = newsq;
                    Generate();
                }
newsq = Next.oldpos - 1;
if (((newsq)<=0x98) && (newsq >= 0 ))
if (((newsq>>4)<10) && ((newsq % 16)<9) && ((newsq | 0xff)>=0))
    if (ChessBoard[Next.oldpos].color == red)
        if (( newsq >=0) && ( newsq <= 0x28) && ((newsq % 16) >=3)
            && ((newsq % 16) <=5))
            if (ChessBoard[newsq].piece == empty)
                {
                    Next.newpos1 = newsq;
                    Generate();
                }
newsq = Next.oldpos + 0x10;
if (((newsq)<=0x98) && (newsq >= 0 ))
if (((newsq>>4)<10) && ((newsq % 16)<9) && ((newsq | 0xff)>=0))
    if (ChessBoard[Next.oldpos].color == red)
        if (( newsq >=0) && ( newsq <= 0x28) && ((newsq % 16) >=3)
            && ((newsq % 16) <=5))
            if (ChessBoard[newsq].piece == empty)
                {
                    Next.newpos1 = newsq;
                    Generate();
                }

newsq = Next.oldpos - 0x10;
if (((newsq)<=0x98) && (newsq >= 0 ))
if (((newsq>>4)<10) && ((newsq % 16)<9) && ((newsq | 0xff)>=0))
    if (ChessBoard[Next.oldpos].color == red)
        if (( newsq >=0) && ( newsq <= 0x28) && ((newsq % 16) >=3)
            && ((newsq % 16) <=5))
            if (ChessBoard[newsq].piece == empty)

```

```
{
    Next.newpos1 = newsq;
    Generate();
}

newsq = Next.oldpos + 1;
if (((newsq)<=0x98) && (newsq >= 0 ))
    if (((newsq>>4)<10) && ((newsq % 16)<9) && ((newsq | 0xff)>=0))
        if (ChessBoard[Next.oldpos].color == black)
            if (( newsq >= 0x70) && ( newsq <= 0x98) && ((newsq % 16) >=3)
                && ((newsq % 16) <=5))
                if (ChessBoard[newsq].piece == empty)
                {
                    Next.newpos1 = newsq;
                    Generate();
                }

newsq = Next.oldpos - 1;
if (((newsq)<=0x98) && (newsq >= 0 ))
    if (((newsq>>4)<10) && ((newsq % 16)<9) && ((newsq | 0xff)>=0))
        if (ChessBoard[Next.oldpos].color == black)
            if (( newsq >= 0x70) && ( newsq <= 0x98) && ((newsq % 16) >=3)
                && ((newsq % 16) <=5))
                if (ChessBoard[newsq].piece == empty)
                {
                    Next.newpos1 = newsq;
                    Generate();
                }

newsq = Next.oldpos + 0x10;
if (((newsq)<=0x98) && (newsq >= 0 ))
    if (((newsq>>4)<10) && ((newsq % 16)<9) && ((newsq | 0xff)>=0))
        if (ChessBoard[Next.oldpos].color == black)
            if (( newsq >= 0x70) && ( newsq <= 0x98) && ((newsq % 16) >=3)
                && ((newsq % 16) <=5))
                if (ChessBoard[newsq].piece == empty)
                {
                    Next.newpos1 = newsq;
                    Generate();
                }
```



```

    }

    newsq = Next.oldpos - 0x10;
    if (((newsq)<=0x98) && (newsq >= 0 ))
        if (((newsq>>4)<10) && ((newsq % 16)<9) && ((newsq | 0xff)>=0))
            if (ChessBoard[Next.oldpos].color == black)
                if (( newsq >= 0x70) && ( newsq <= 0x98) && ((newsq % 16) >=3)
                    && ((newsq % 16) <=5))
                    if (ChessBoard[newsq].piece == empty)
                        {
                            Next.newpos1 = newsq;
                            Generate();
                        }
    break;

    /* 为士生成 4 个方向的非吃子移动 */
case assist :
for (dir = 3; dir >= 0; dir--)
{
    newsq = Next.oldpos + AssistDir[dir];
    if (((newsq)<=0x98) && (newsq >= 0 ))
    if (((newsq>>4)<10) && ((newsq % 16)<9) && ((newsq | 0xff)>=0))
        if (ChessBoard[Next.oldpos].color == red)
            if (( newsq >=0) && ( newsq <= 0x28) && ((newsq % 16) >=3)
                && ((newsq % 16) <=5))
                if (ChessBoard[newsq].piece == empty)
                    {
                        Next.newpos1 = newsq;
                        Generate();
                    }
}

for (dir = 3; dir >= 0; dir--)
{
    newsq = Next.oldpos + AssistDir[dir];
    if (((newsq)<=0x98) && (newsq >= 0 ))
        if (((newsq>>4)<10) && ((newsq % 16)<9) && ((newsq | 0xff)>=0))
            if (ChessBoard[Next.oldpos].color == black)
                if (( newsq >= 0x70) && ( newsq <= 0x98) && ((newsq % 16) >=3)

```

```

        && ((newsq % 16) <=5))
    if (ChessBoard[newsq].piece == empty)
    {
        Next.newpos1 = newsq;
        Generate();
    }
}
break;
/* 为象生成 4 个方向的非吃子移动 */
case bishop:
    first = 3;
    last = 0;
    for (dir = first; dir >= last; dir--)
    {
        direction = BishopDir[dir];
        newsq = Next.oldpos + direction;
        if (BishopDir[dir]== 0x1E)
            tmpBishopDir = 0x0F;
        if (BishopDir[dir]== -0x1E)
            tmpBishopDir= -0x0F;
        if (BishopDir[dir]== 0x22)
            tmpBishopDir= 0x11;
        if (BishopDir[dir]== -0x22)
            tmpBishopDir= -0x11;
        if ((newsq <= 0x98) && (newsq >=0))
        if (((newsq>>4)<10) && ((newsq % 16)<9) && ((newsq | 0xff)>=0))
        if (((newsq-tmpBishopDir) <= 0x98) && ((newsq-tmpBishopDir) >=0))
        if (((newsq-tmpBishopDir)>>4)<10) && (((newsq-tmpBishopDir) %
        16)< 9))
        if (( ChessBoard[newsq - tmpBishopDir].piece == empty ))
            if (ChessBoard[newsq].piece == empty)
                if (ChessBoard[Next.oldpos].color == red)
                    if (( newsq >=0) && ( newsq <= 0x48))
                        Next.newpos1 = newsq;
                        Generate();
    }
    for (dir = first; dir >= last; dir--)
    {
        direction = BishopDir[dir];

```

```

        newsq = Next.oldpos + direction;
        /* 生成四个方向的非吃子移动 */
        if (BishopDir[dir]== 0x1E)
            tmpBishopDir = 0x0F;
        if (BishopDir[dir]== -0x1E)
            tmpBishopDir= -0x0F;
        if (BishopDir[dir]== 0x22)
            tmpBishopDir= 0x11;
        if (BishopDir[dir]== -0x22)
            tmpBishopDir= -0x11;
        if ((newsq <= 0x98) && (newsq >=0))
        if (((newsq>>4)<10) && ((newsq % 16)<9) && ((newsq | 0xff)>=0))
        if (((newsq - tmpBishopDir)<= 0x98) && ((newsq - tmpBishopDir) >=0))
            if (((newsq-tmpBishopDir)>>4)<10) && (((newsq-tmpBishopDir) %
            16)< 9) &&
                (((newsq-tmpBishopDir) | 0xff)>= 0))
            if (( ChessBoard[newsq - tmpBishopDir].piece == empty ))
                if (ChessBoard[newsq].piece == empty)
                    if (ChessBoard[Next.oldpos].color == black)
                        if (( newsq >= 0x50) && ( newsq <= 0x98))
                            Next.newpos1 = newsq;
                            Generate();
                    }
                break;

case knight :    //马
    for (dir = 7; dir >= 0; dir--)
    {
        newsq = Next.oldpos + KnightDir[dir];
        /* 为马生成 8 个方向的非吃子移动 */
        if (KnightDir[dir] == 0x0E)
            tmpKnightDir= 0x0F;
        if (KnightDir[dir]== -0x0E)
            tmpKnightDir= -0x0F;
        if (KnightDir[dir]== 0x12)
            tmpKnightDir = 0x11;
        if (KnightDir[dir]== -0x12)
            tmpKnightDir = -0x11;
        if (KnightDir[dir]== 0x1F)

```

```

        tmpKnightDir = 0x0F;
    if (KnightDir[dir]== -0x1F)
        tmpKnightDir = -0x0F;
    if (KnightDir[dir]== 0x21)
        tmpKnightDir = 0x11;
    if (KnightDir[dir]== -0x21)
        tmpKnightDir = -0x11;

    if ((newsq <= 0x98) && (newsq >=0))
    if (((newsq % 16)>=0) && ((newsq % 16)< 9) && ((newsq | 0xff)>=0))
    if (((newsq-tmpKnightDir) <= 0x98) && ((newsq-tmpKnightDir) >=0))
    if (((newsq-tmpKnightDir) /16)<10) && (((newsq-tmpKnightDir) % 16)
    < 9) &&
        (((newsq-tmpKnightDir) | 0xff) >= 0))
    if ( ChessBoard[newsq - tmpKnightDir].piece == empty )
        if (ChessBoard[newsq].piece == empty)
        {
            Next.newpos1 = newsq;
            Generate();
        }
    }
    break;

case rook :      //车
case gunner :    //炮
    first = 3;
    last = 0;
    for (dir = first; dir >= last; dir--)
    {
        direction = DirectionTab[dir];
        newsq = Next.oldpos + direction;
        /* 为炮生成 4 个方向的非吃子移动 */
        while (((newsq>>4)<10) && ((newsq % 16)<9) &&
            (newsq <= 0x98) && (newsq >=0) && ((newsq | 0xff)>=0))
        {
            if (ChessBoard[newsq].piece != empty) goto TENNMOV;
            Next.newpos1 = newsq;
            Generate();
            newsq = Next.newpos1 + direction;

```

```

    }
TENNMOV:        continue;
    }
    break;
} /* switch */
}

```

10. 函数名称: void InitMovGen()

说明: 生成移动。生成所有可能的移动并把它逐个放在缓冲区内。

开始:

Player 指移动棋子的一方;

MoveTable[Depth-1] 上一次执行过的移动。

结束:

缓冲区包含全部生成的移动。

生成移动的顺序:

吃子

非吃子

函数定义如下:

```

void InitMovGen()
{
    INDEXTYPE index;

    BufCount = BufPnt = 0;
    /* 从将开始生成吃子移动 */
    for (index = 0; index <= PawnNumber[Opponent]; index++)
        if (PieceTable[Opponent][index].ipiece != empty)
        {
            Next.newpos1 = PieceTable[Opponent][index].isquare;
            CapMovGen();
        }

    /* 从兵开始生成非吃子移动 */
    for (index = PawnNumber[Player]; index >= 0; index--)
        if (PieceTable[Player][index].ipiece != empty)
        {
            Next.oldpos = PieceTable[Player][index].isquare;
            NonCapMovGen();
        }
}

```

11. 函数名称: void MovGen()

说明: 把生成的移动一个接一个地保存在 Next 链表中。没有棋子可移动时生成零移动。

函数定义如下:

```
void MovGen()
{
    if (BufPnt >= BufCount)
        Next = ZeroMove;
    else
    {
        BufPnt++;
        Next = Buffer[BufPnt];
    }
}
```

9.4.3 局面(势)评价函数的实现

局面(势)评价函数的编制非常复杂。本程序主要考虑了四种因素: 棋子本身价值、棋子位置值、棋子攻击性、棋子灵活性。

由于开放了源程序, 读者可以根据自己的需要, 改变、增加、减少 Evaluate.cpp 中的局面(势)评价函数, 或者编制格式能被 Globalfun.cpp 中相应读谱函数识别的棋谱库文件, 从而改变程序的弈棋水平, 从中可以感受到计算机技术的巨大魅力, 也可以体会中国象棋的博大精深。以下函数位于 CHESS 项目的 Evaluate.cpp 文件中, 属于自定义函数, 所有的可视化类函数均可调用。具体制作过程如下:

- (1) 选择 File | Open Project 命令, 打开 Open Project 对话框, 利用该对话框打开 CHESS 项目;
- (2) 选择 View|Project Manager 命令, 打开 Project Manager 对话框, 在其中选择源文件, 双击 Evaluate.cpp 打开它;
- (3) 在打开的文件中手工添加各种函数。

在 Evaluate.cpp 文件中, 手工添加的实现局面(势)评价的自定义函数如下所示:

1. 函数名称: inline int PiecePosVal(PIECETYPE piece, COLORTYPE color, SQUARE TYPE square)

说明: 计算棋子位置值。

函数定义如下:

```
inline int PiecePosVal(PIECETYPE piece, COLORTYPE color, SQUARETYPE square)
{
```

```

    return (PieceValue[piece] + PVTable[color][piece][square]);
}

```

2. 函数名称: void CalcPVTable(void)

说明: 计算棋子值表, 后面的静态估价函数需要。

函数定义如下:

```

void CalcPVTable(void)
{
    typedef SETOFFILE PAWNTABTYPE[9];

    /* 存储棋子的攻击性 */
    char attackvalue[2][0x99];
    /* 棋子的控位功能值 */
    int pvcontrol[2][6][0x99];

    COLORTYPE losingcolor;    /* 子力交换时吃亏的一方 */
    int posval;                /* 棋子位置值 */
    int attval;                /* 棋格的攻击值 */
    FILETYPE line;            /* 棋子的列数 */
    RANKTYPE rank;            /* 棋子的行数 */
    char dist, kingdist;      /* 到棋盘中心的距离, 离将(帅)的距离 */
    short direct;             /* 指示直接攻击 */
    int cnt;                   /* 攻击值变量 */
    COLORTYPE color, oppcolor; /* 己方和对方 */
    PIECETYPE piececount;
    SQUARETYPE square;

    SQUARETYPE asquare;
    COLORTYPE acolor;
    PIECETYPE apiece;

    DIRTYPE dir;              /* 棋子可移动的方向(左、右、前、后) */
    EDGESQUARETYPE sq;
    int colortemp, pietemp;    /* 临时变量 */

    material = pawntotalmaterial = totalmaterial = mating = 0;
}

```

```

for (square = 0; square < 0x99; square++)
    if (((square>>4)<10) && ((square % 16)<9) && ((square | 0xff)>=0))
        if (ChessBoard[square].piece != empty)
            if (ChessBoard[square].piece != king)
            {
                temp = PieceValue[ChessBoard[square].piece];
                totalmaterial += temp;
                if (ChessBoard[square].piece == pawn)
                    pawntotalmaterial += PieceValue[pawn];
                if (ChessBoard[square].color == red) temp = -temp;
                material -= temp;
            }
materiallevel = max(0, totalmaterial - 0x2000) / 0x100;

/* 根据物质力量来确定是否进行子力交换 */
if (material < 0)
    losingcolor = red;
else
    losingcolor = black;
mating = ((totalmaterial - abs(material)) / 2 <= PieceValue[gunner] * 2)
    && (abs(material) >= PieceValue[rook] - PieceValue[gunner]);
/* 计算攻击值 */
for (rank = 0; rank < 10; rank++)
    for (line = 0; line < 9; line++)
    {
        square = (rank << 4) + line;
        attval = max(0, 9 - 3 * (distanCR[rank] + distanCL[line]));
        /* 占据中心的重要性 */
        /* 棋子所在行数的重要性 */
    }
/*
在 C++Builder 中枚举常量加一个常数, 不能成为后一个常量。
只得采用整数间接来达到目的。
*/
for (colortemp = 0; colortemp <= 1; colortemp++)
{
    { if (colortemp==0)
        color = red;
        if (colortemp==1)
            color = black;
    }
}

```



```

    }
    attackvalue[color][square] = char(((squaresrankvalue[rank] * 3 *
        (materiallevel + 8)) >> 5) + attval);
    square = (((0xa0 - square) / 16) << 4) + line;
    if ((square == 0xa0) || (square == 0x90) || (square == 0x80) ||
        (square == 0x70) || (square == 0x60) || (square == 0x50) ||
        (square == 0x40) || (square == 0x30) || (square == 0x20) ||
        (square == 0x10) || (square == 0x0))
    {
        square -= 0x10;
    }
    else
    {
        square = square;
    }
}
}
/*
在 C++Builder 中枚举常量加一个常数, 不能成为后一个常量。
只得采用整数间接来达到目的。
*/
for (colortemp = 0; colortemp <= 1; colortemp++)
{
    {
        if (colortemp==0)
            color=red;
        if (colortemp ==1)
            color = black;
    }
    oppcolor = (COLORTYPE)(1 - color);
    if (materiallevel > 0)
        /* 计算围绕对方将(或帅)的四个位置的重要性 */
        for (dir = 0; dir < 4; dir++)
        {
            sq = PieceTable[oppcolor][0].isquare + DirectionTab[dir];
            if (((square>>4)<10) && ((square % 16)<9) && ((square | 0xff)>=0))
                attackvalue[color][sq] += char(((12 * (materiallevel + 8)) >> 5));
        }
    }
}

```

```
/* 计算棋子的局面控制值 */
```

```
/*
```

在 C++Builder 中枚举常量加一个常数，不能成为后一个常量。

只得采用整数间接来达到目的。

```
*/
```

```
for (square = 0; square <= 0x98; square++)
    if (((square>>4)<10) && ((square % 16)<9) && ((square | 0xff)>=0))
        for (colortemp = 0; colortemp <= 1; colortemp++)
            {
                {
                    if (colortemp==0)
                        color = red;
                    if (colortemp==1)
                        color = black;
                }
                piececount = rook;
                pvcontrol[color][piececount][square] = 0;
            }
}
```

```
/*
```

在 C++Builder 中枚举常量加一个常数，不能成为后一个常量。

只得采用整数间接来达到目的。

```
*/
```

```
for (square = 0; square <= 0x98; square++)
    if (((square>>4)<10) && ((square % 16)<9) && ((square | 0xff)>=0))
        for (colortemp = 0; colortemp <= 1; colortemp++)
            {
                {
                    if (colortemp==0)
                        color = red;
                    if (colortemp==1)
                        color = black;
                }
                for (dir = 3; dir > 0; dir--)
                    {
                        piececount = rook;
```

```
/* 计算棋子每个方位的攻击性。己方对对方同样棋子的攻击，算是正常攻击；
```

```
对不是同样棋子的攻击，算是半个攻击。
```

```
*/
```

```

        cnt = 0;
        sq = square;
        direct = 1;
        do
        {
            sq += DirectionTab[dir];
            if (((sq >> 4) > 10) || ((sq % 16) > 9)
            || ((sq | 0xff) < 0)) goto TEN;
            temp = attackvalue[color][sq];
            if (direct)
                cnt += temp;
            else
                cnt += (temp >> 1);
            if (ChessBoard[sq].piece != empty)
                if ((ChessBoard[sq].piece != piececount))
                    direct = 0;
        } while (ChessBoard[sq].piece != pawn);
TEN:    pvcontrol[color][piececount][square] += (cnt >> 2);
    }
}

```

/* 逐个计算棋子表 */

/*

在 C++Builder 中枚举常量加一个常数，不能成为后一个常量。

只得采用整数间接来达到目的。

*/

```

for (square = 0; square <= 0x98; square++)
    if (((square >> 4) < 10) && ((square % 16) < 9) && ((square | 0xff) >= 0))
    {
        for (colortemp = 0; colortemp <= 1; colortemp++)
        {
            {
                if (colortemp == 0)
                    color = red;
                if (colortemp == 1)
                    color = black;
            }
        }
    }

```

```

line = FILETYPE(square % 16);

```

```

rank = RANKTYPE(square >> 4);
if (color == black) rank = RANKTYPE(9 - rank);
dist = char(distanR[rank] + distanL[line]);
    if ((rank<=9) && (line<=8))
kingdist = char(abs((square >> 4) - (PieceTable[oppcolor][0].
isquare
    >> 4)) + ((square - PieceTable[oppcolor][0].isquare) % 16));
/*

```

在C++Builder中枚举常量加一个常数,不能成为后一个常量。

只得采用整数间接来达到目的。

```

*/
    for (piecetemp = 0; piecetemp <= 6; piecetemp++)
    {
        {
            if (piecetemp==0)
                piececount=king;
            if (piecetemp==1)
                piececount=rook;
            if (piecetemp==2)
                piececount=gunner;
            if (piecetemp==3)
                piececount=knight;
            if (piecetemp==4)
                piececount=assist;
            if (piecetemp==5)
                piececount=bishop;
            if (piecetemp==6)
                piececount=pawn;
        }

        posval = 0;          /* 根据棋格的位置计算棋子位置值 */
        if (mating)
        {
            if (piececount == king)
            if (color == losingcolor) /* 子力交换估算 */
            {
                posval = 128 - 16 * distanR[rank]
                    - 12 * distanL[line];
            }
        }
    }
}

```

```

        if (distanR[rank] == 3)
            posval -= 16;
    }
    else
    {
        posval = 128 - 4 * kingdist;
        if ((distanR[rank] >= 2) || (distanL[line] == 3))
            posval -= 16;
    }
}
else
{
    temp = pvcontrol[color][rook][square];
    /* 正常估价函数 */
    switch (piececount)
    {
        case king :
            if (materiallevel <= 0) posval = -2 * dist;
            break;
        case assist :
            case bishop:
                posval = temp >> 2;
                break;
            case rook :
                posval = temp;
                break;
            case gunner :
                posval = temp >> 1;
                break;
            case knight :
                cnt = 0;
                for (dir = 0; dir < 8; dir++)
                {
                    sq = square + KnightDir[dir];
                    if (((square >> 4) < 10) && ((square % 16) < 9) && ((square | 0xff) >= 0))
                        cnt += attackvalue[color][sq];
                }
                posval = (cnt >> 1) - dist * 3;
            }
    }
}

```

```

        break;
        case pawn :
            posval = pawnrank[rank] +
                pawnfilefactor[line] * (rank + 2) - 12;

        }
    }
    PVTable[color][piececount][square] = posval;
}
}
}

```

```
RootValue = 0;
```

//双重攻击, 应该加分

```
int tmp = 0;
```

```
int i = 0;
```

```
SQUARETYPE squar[12];
```

```
squar[0] = 0;
```

```
/*
```

在 C++Builder 中枚举常量加一个常数, 不能成为后一个常量。

只得采用整数间接来达到目的。

```
*/
```

```
for (asquare = 0; asquare <= 0x98; ((int)asquare)++)
```

```
if (((asquare>>4)<10) && ((asquare % 16)<9) && ((asquare | 0xff)>=0))
```

```
{
```

```
    for ( colortemp = 0 ;colortemp <= 1; colortemp++)
```

```
    {
```

```
        if (colortemp==0)acolor=red;
```

```
        if (colortemp==1)acolor=black;
```

```
    }
```

```
    for (piecetemp = 0; piecetemp <= 6; piecetemp++)
```

```
    {
```

```
        if (piecetemp==0)apiece=king;
```

```
        if (piecetemp==1)
```

```
            apiece=rook;
```

```
        if (piecetemp==2)
```

```
            apiece=gunner;
```

```
        if (piecetemp==3)
```

```
        apiece=knight;
    if (piecetemp==4)
        apiece=assist;
    if (piecetemp==5)
        apiece=bishop;
    if (piecetemp==6)
        apiece=pawn;
    }

    for (square = 0; square <= 0x98; (int) square++)
        if (((square>>4)<10) && ((square % 16)<9) && ((square | 0xff)>=0))
            if ((ChessBoard[square].piece == apiece) && (ChessBoard[square].color
                == acolor))
            {

                if (PieceAttacks(apiece, acolor, asquare, square))
                    tmp +=1;
                squar[i] = square;
                i +=1;
            }
    }

    if (tmp >= 2)
    {
        {
            if (colortemp==0)acolor=red;
            if (colortemp==1)acolor=black;
        }
        {
            if (piecetemp==0)
                apiece=king;
            if (piecetemp==1)
                apiece=rook;
            if (piecetemp==2)
                apiece=gunner;
            if (piecetemp==3)
                apiece=knight;
            if (piecetemp==4)
                apiece=assist;
            if (piecetemp==5)
                apiece=bishop;
```

```

        if (piecetemp==6)
            apiece=pawn;
    }
    if (!Attacks((COLORTYPE)!acolor, asquare))
    if (PieceValue[ChessBoard[squar[0]].piece] >= PieceValue[ChessBoard
[squar[1]].piece])
        PVTable[acolor][apiece][asquare] += PieceValue[ChessBoard[squar
[0]].piece] / 10;
    else
    PVTable[acolor][apiece][asquare] += PieceValue[ChessBoard[squar[1]].
piece] / 10;
    }

```

/*

在 C++Builder 中枚举常量加一个常数，不能成为后一个常量。

只得采用整数间接来达到目的。

*/

//过载，应该减分

tmp = 0;

for (square = 0; square <= 0x98; (int)square++)

if (((square>>4)<10) && ((square % 16)<9) && ((square | 0xff)>=0))

{

{

for (colortemp = 0; colortemp <= 1; colortemp++)

for (piecetemp = 0; piecetemp <= 6; piecetemp++)

for (asquare = 0; asquare <= 0x98; (int)asquare++)

if (((asquare>>4)<10) && ((asquare % 16)<9) && ((asquare | 0xff)>=0))

{

{

if (colortemp==0)acolor=red;

if (colortemp==1)acolor=black;

}

{

if (piecetemp==0)apiece=king;

if (piecetemp==1)

apiece=rook;

if (piecetemp==2)

apiece=gunner;

if (piecetemp==3)


```

        apiece=knight;
        if (piecetemp==4)
            apiece=assist;
        if (piecetemp==5)
            apiece=bishop;
        if (piecetemp==6)
            apiece=pawn;
    }

    if (PieceAttacks(apiece, acolor, asquare, square))
        tmp +=1;
    }

    if ((tmp % 2) == 0)
        PVTable[ChessBoard[square].color][ChessBoard[square].piece][square]+=
        PieceValue[ChessBoard[square].piece] / 10;
    else
        PVTable[ChessBoard[square].color][ChessBoard[square].piece][square]-=
        PieceValue[ChessBoard[square].piece] / 10;

}

/*计算车、马、炮棋子的活动性*/
for (square = 0; square <= 0x98; square++)
    if (((square>>4)<10) && ((square % 16)<9) && ((square | 0xff)>=0))
    {
        if (ChessBoard[square].piece == rook)
        {
            RookActive = -1;
            sq = square;
            do
            {
                sq += 0x10;
                RookActive += 1;
                sq = sq;
            } while ((sq >> 4) <= 9 && ChessBoard[sq].piece == empty );
            sq = square;
            do
            {
                sq -= 0x10;

```

```

    RookActive += 1;
        sq = sq;
    } while ((sq >> 4) <= 9    && ChessBoard[sq].piece == empty );
    sq = square;
        do
        {
            sq += 0x01;
            RookActive += 1;
                sq = sq;
        } while ((sq % 16) <= 8    && ChessBoard[sq].piece == empty );
    sq = square;
        do
        {
            sq -= 0x01;
            RookActive += 1;
                sq = sq;
        } while ((sq % 16) <= 8    && ChessBoard[sq].piece == empty );
    }

if (ChessBoard[square].piece == gunner)
{
    GunnerActive = -1;
    sq = square;
        do
        {
            sq += 0x10;
            GunnerActive += 1;
                sq = sq;
        } while ((sq >> 4) <= 9    && ChessBoard[sq].piece == empty );
    sq = square;
        do
        {
            sq -= 0x10;
            GunnerActive += 1;
                sq = sq;
        } while ((sq >> 4) <= 9    && ChessBoard[sq].piece == empty );
    sq = square;
        do
        {

```

```

sq += 0x01;
GunnerActive += 1;
    sq = sq;
} while ((sq % 16) <= 8  && ChessBoard[sq].piece == empty );
sq = square;
do
{
sq -= 0x01;
GunnerActive += 1;
    sq = sq;
} while ((sq % 16) <= 8  && ChessBoard[sq].piece == empty );
}

if (ChessBoard[square].piece == knight)
{
KnightActive = 0;
sq = square;
sq += 0x10;
    if (((sq >> 4) <= 9)  && (ChessBoard[sq].piece == empty))
        KnightActive += 2;
sq = square;
sq -= 0x10;
    if (((sq >> 4) <= 9)  && (ChessBoard[sq].piece == empty))
        KnightActive += 2;

sq = square;
sq += 0x01;
    if (((sq % 16) <= 8)  && (ChessBoard[sq].piece == empty))
        KnightActive += 2;
sq = square;
sq -= 0x01;
    if (((sq % 16) <= 8)  && (ChessBoard[sq].piece == empty))
        KnightActive += 2;
}
}

/*估算车、马、炮棋子活动性值*/
for (square = 0; square <= 0x98; square++)
    if (((square>>4)<10) && ((square % 16)<9) && ((square | 0xff)>=0))

```

```

    {
        if ((ChessBoard[square].piece == rook) && (ChessBoard[square].
color==red))
            PVTable[red][rook][square] = RookActive / 19 * 10;
        if ((ChessBoard[square].piece == rook) && (ChessBoard[square].
color==black))
            PVTable[black][rook][square] = RookActive / 19 * 10;

        if ((ChessBoard[square].piece == gunner) && (ChessBoard[square].
color==red))
            PVTable[red][gunner][square] = GunnerActive / 19 * 10;
        if ((ChessBoard[square].piece == gunner) && (ChessBoard[square].
color==black))
            PVTable[black][rook][square] = GunnerActive / 19 * 10;

        if ((ChessBoard[square].piece == knight) && (ChessBoard[square].
color==red))
            PVTable[red][knight][square] = KnightActive / 8 * 10;
        if ((ChessBoard[square].piece == knight) && (ChessBoard[square].
color==black))
            PVTable[black][knight][square] = KnightActive / 8 * 10;
    }

/*逐个计算棋子(车、马、炮、兵)的位置值,保存在棋子值表中*/
    for (sq = 0; sq <= 0x98; sq++)
        if (((sq>>4)<10) && ((sq % 16)<9) && ((sq | 0xff)>=0))
        {
            if ((ChessBoard[sq].piece == pawn) && (ChessBoard[sq].color == red))
                PVTable[red][pawn][sq] += pawnposvalu[sq >> 4][sq % 16] / 10;
        }
    for (sq = 0; sq <= 0x98; sq++)
        if (((sq>>4)<10) && ((sq % 16)<9) && ((sq | 0xff)>=0))
        {

            if ((ChessBoard[sq].piece == pawn) && (ChessBoard[sq].color ==
black))
                PVTable[black][pawn][sq] += pawnposvalu[9 - (sq >> 4)][sq % 16] / 10;
        }
}

```

```
for (sq = 0; sq <= 0x98; sq++)
    if (((sq>>4)<10) && ((sq % 16)<9) && ((sq | 0xff)>=0))
    {
        if ((ChessBoard[sq].piece == gunner) && (ChessBoard[sq].color == red))
            PVTable[red][gunner][sq] += gunposvalu[sq >> 4][sq % 16] / 10;
    }
for (sq = 0; sq <= 0x98; sq++)
    if (((sq>>4)<10) && ((sq % 16)<9) && ((sq | 0xff)>=0))
    {

        if ((ChessBoard[sq].piece == gunner) && (ChessBoard[sq].color ==
            black))
            PVTable[black][gunner][sq] += gunposvalu[9 - (sq >> 4)][sq % 16] /
            10;
    }

for (sq = 0; sq <= 0x98; sq++)
    if (((sq>>4)<10) && ((sq % 16)<9) && ((sq | 0xff)>=0))
    {
        if ((ChessBoard[sq].piece == knight) && (ChessBoard[sq].color == red))
            PVTable[red][knight][sq] += knightposvalu[sq >> 4][sq % 16] / 10;
    }
for (sq = 0; sq <= 0x98; sq++)
    if (((sq>>4)<10) && ((sq % 16)<9) && ((sq | 0xff)>=0))
    {
        if ((ChessBoard[sq].piece == knight) && (ChessBoard[sq].color == black))
            PVTable[black][knight][sq] += knightposvalu[9 - (sq >> 4)][sq % 16]
            / 10;
    }

for (sq = 0; sq <= 0x98; sq++)
    if (((sq>>4)<10) && ((sq % 16)<9) && ((sq | 0xff)>=0))
    {
        if ((ChessBoard[sq].piece == rook) && (ChessBoard[sq].color == red))
            PVTable[red][rook][sq] += rookposvalu[sq >> 4][sq % 16] / 10;
    }
for (sq = 0; sq <= 0x98; sq++)
    if (((sq>>4)<10) && ((sq % 16)<9) && ((sq | 0xff)>=0))
```

```

{
    if ((ChessBoard[sq].piece == rook) && (ChessBoard[sq].color == black))
        PVTable[black][rook][sq] += rookposvalu[9 - (sq >> 4)][sq % 16] / 10;
}

for (square = 0; square <= 0x98; square++) /* 计算根值 */
    if (((square >> 4) < 10) && ((square % 16) < 9) && ((square | 0xff) >= 0))
        if (ChessBoard[square].piece != empty)
            if (ChessBoard[square].color == Player)
                RootValue +=
                    PiecePosVal(ChessBoard[square].piece, Player, square);
            else
                RootValue -=
                    PiecePosVal(ChessBoard[square].piece, Opponent, square);
}

/*****

```

3. 函数名称: int StatEval(MOVETYPE *move)

说明: 移动棋子的静态评估。

函数定义如下:

```

int StatEval(MOVETYPE *move)
{
    value = 0;

    if (move->content != empty) /* 正常移动 */
    {
        value += PiecePosVal(move->content, Opponent, move->newpos1);
        /* 在子力落后的情况下进行子力交换应给予扣分 */
        if (abs(MainEvaluat) >= 0x100)
            if (move->content != pawn)
                if ((ProgramColor == Opponent) == (MainEvaluat >= 0))
                    value -= EXCHANGEVALUE;
    }

    return (value + PiecePosVal(move->movpiece, Player, move->newpos1) -
            PiecePosVal(move->movpiece, Player, move->oldpos));
}

```

9.4.4 搜索过程的实现

象棋对弈过程是使用 $\alpha \sim \beta$ 博弈树搜索算法而实现的, 根据如此原理, 编制了如下函数。它们位于 CHESS 项目中的 TreeSearch.cpp 文件中, 属于自定义全局函数, 所有的可视化类函数均可调用。具体制作过程如下:

- (1) 选择 File | Open Project 命令, 打开 Open Project 对话框, 利用该对话框打开 CHESS 项目;
- (2) 选择 View | Project Manager 命令, 打开 Project Manager 对话框, 在其中选择源文件, 双击 TreeSearch.cpp 打开它;
- (3) 在打开的文件中手工添加各种函数。

在 TreeSearch.cpp 文件中手工添加的实现搜索过程的自定义全局函数如下所示:

1. 函数名称: static void RemoveKillMove(void)

说明: 初始化将军表。

函数定义如下:

```
static void RemoveKillMove(void)
{
    CheckTable[-1] = 0;
}
```

2. 函数名称: static void getprogramstate(void)

说明: 备份搜索且设置对话环境。

函数定义如下:

```
static void getprogramstate(void)
{
    COLORTYPE oldplayer;

    searchstatedepth = Depth;
    while (Depth > 0)
    {
        Depth--;
        oldplayer = Opponent;
        Opponent = Player;
        Player = oldplayer;
        Perform(&MoveTable[Depth], 1); //回退一步
    }
    Depth--;
```

```
    if (Opan) TakeBackMove(&MoveTable[Depth]);  
}
```

3. 函数名称: static void getsearchstate(void)

说明: 恢复搜索环境。

函数定义如下:

```
static void getsearchstate(void)  
{  
    COLORTYPE oldplayer;  
  
    if (Opan) MakeMove(&MoveTable[Depth+1]);  
    Depth++;  
    while (Depth < searchstatedepth)  
    {  
        Perform(&MoveTable[Depth], 0); //向前看一步  
        oldplayer = Player;  
        Player = Opponent;  
        Opponent = oldplayer;  
        Depth++;  
    }  
}
```

4. 函数名称: short generatedbefore(PARAMTYPE *P)

说明: 检查以前是否生成过棋子移动。

函数定义如下:

```
short generatedbefore(PARAMTYPE *P)  
{  
  
    if (P->S->movgentype != mane)  
    {  
        IF_EQMOVE(MoveTable[Depth], P->bestpath[Depth]) return 1;  
    }  
    return 0;  
}
```

5. 函数名称: inline short cut(MAXTYPE cutval, PARAMTYPE *P)

说明: 测试剪枝。剪枝值包含最大可能估值。

函数定义如下:

```
inline short cut(MAXTYPE cutval, PARAMTYPE *P)
{
    short ct = 0;

    if (cutval <= P->alpha)
    {
        ct = 1;
        if (P->S->maxval < cutval) P->S->maxval = cutval;
    }
    return ct;
}
```

6. 函数名称: static short update(PARAMTYPE *P)

说明: 执行移动, 计算估值, 测试剪枝等。

函数定义如下:

```
static short update(PARAMTYPE *P)
{
    short selection;

    AddNode(&Nodes);
    P->S->nextply = P->ply - 1; /* 计算下一个回合 */
    if (Level == matesearch) /* 将死搜索 */
    {
        Perform(&MoveTable[Depth], 0); /* 向前看一步 */
        /* 检查移动是否合法 */
        if (Attacks(Opponent, PieceTable[Player][0].isquare) || (Repetition(0)
>= 2))
        {
            goto TAKEBACKMOVE;
            if (!Depth) LegalMoves++;
            CheckTable[Depth] = 0;
            P->S->next.value = P->S->next.evaluation = 0;
            if (P->S->nextply <= 0) /* 计算将军并执行剪枝 */
            {
                if (!P->S->nextply)
                    CheckTable[Depth] = Attacks(Player,
                    PieceTable[Opponent][0].isquare);
                if (!CheckTable[Depth])
```

```

        if (cut(P->S->next.value, P)) goto TAKEBACKMOVE;
    }
    goto ACCEPTMOVE;
}
/* 在第一轮中保证有限的特殊吃子优先得到搜索 */
if (MaxDepth <= 1)
    if (P->S->capturesearch && Depth >= 2)
        if (!(MoveTable[Depth].content < MoveTable[Depth].movpiece)
            || (P->S->movgentype == specialcap) ||
            (MoveTable[Depth].oldpos
                == MoveTable[Depth-2].newpos1)))
            goto CUTMOVE;
    /* 计算下一步静态增量估值 */
    P->S->next.value = -P->inf->value + StatEvalu(&MoveTable[Depth]);
    /* 计算将军表(只计算能将军的移动)不能算是一个回合 */
    CheckTable[Depth] = PieceAttacks(MoveTable[Depth].movpiece, Player,
        MoveTable[Depth].newpos1, PieceTable[Opponent][0].isquare);
    if (CheckTable[Depth]) P->S->nextply = P->ply;
    /* 在最后的吃子搜索中进行选择 */
    selection = ((P->S->nextply <= 0) && !CheckTable[Depth] && (Depth > 0));
    if (selection) /* 检查估值 */
        if (cut(P->S->next.value + 0, P)) goto CUTMOVE;
    Perform(&MoveTable[Depth], 0); /* 向前看一步 */
    /* 检查移动是否合法 */
    if (Attacks(Opponent, PieceTable[Player][0].isquare) || (Repetition(0)
        >= 2))
        goto TAKEBACKMOVE;

    if (!Depth)
    {
        LegalMoves++;
        P->S->next.value += random(4);
    }
    P->S->next.evaluation = P->S->next.value;
ACCEPTMOVE:
    if (Analysis)
        PrintMove();
    return 0;
TAKEBACKMOVE:

```

```

    Perform(&MoveTable[Depth], 1); //回退一步
CUTMOVE:
    if (Analysis)
        PrintMove();
    return 1;
}

```

7. 函数名称: static short drawgame(SEARCHTYPE *S)

说明: 给予和棋加分或扣分, 如果确实是和棋的话就定为和棋
函数定义如下:

```

static short drawgame(SEARCHTYPE *S)
{
    int drawcount;
    REPEATTYPE searchrepeat;
    SIXTYTYPE searchsixty;

    if (Depth == 1)
    {
        searchsixty = SixtyMoveCnt();
        searchrepeat = Repetition(0);
        if (searchrepeat >= 3)
        {
            S->next.evaluation = 0;
            return 1;
        }

        drawcount = 0;
        if (searchsixty >= 116) /* 58 次移动中没有吃子 */
            drawcount = 3;
        else
        {
            if (searchrepeat >= 2) /* 第二次重复 */
                drawcount = 2;
            else if (searchsixty >= 20) /* 10 次移动中没有吃子 */
                drawcount = 1;
        }

        S->next.value += (repeatevalu / 4) * drawcount;
        S->next.evaluation += (repeatevalu / 4) * drawcount;
    }
}

```

```

    }
    if (Depth >= 3)
    {
        searchrepeat = Repetition(1);
        if (searchrepeat >= 2)      /* 连续重复算平局 */
        {
            S->next.evaluation = 0;
            return 1;
        }
    }
    return 0; //能赢就要赢
}

```

8. 函数名称: inline void updatebestpath(PARAMTYPE *P)

说明: 根据计算出的路线和最大值更改最好路线和最大估值。

函数定义如下:

```

inline void updatebestpath(PARAMTYPE *P)
{
    memcpy(P->bestpath, &P->S->path[0], sizeof(PATHTYPE));
    /* *bestpath = P->S->path; */
    P->bestpath[Depth] = MoveTable[Depth];
    if (!Depth)
    {
        MainEvaluat = P->S->maxval;
        if (Level == matesearch)
            P->S->maxval = alphawindow;
        if (Analysis) PrintBestMove(&MainPath[0], MainEvaluat);
    }
}

```

9. 函数名称: static short loopbody(PARAMTYPE *P)

说明: 棋路树搜索程序的内部循环。MoveTable[Depth]表包含了移动。

函数定义如下:

```

static short loopbody(PARAMTYPE *P)
{
    COLORTYPE oldplayer;
    short lastanalysis;

```

```

if (generatedbefore(P)) return 0;
if (Depth < MAXPLY)
{
    P->S->path[Depth + 1] = ZeroMove;
    if (P->S->movgentype == mane)
        memmove(&P->S->path[0], P->bestpath, sizeof(PATHTYPE));
        /* P->S->path = *bestpath; */
}
/* Principvar 意味着主要变着的搜索 */
/* 零窗口意味着宽度为 0 的  $\alpha \sim \beta$  窗口 */
P->S->next.principvar = 0;
P->S->zerowindow = 0;
if (P->inf->principvar)
    if (P->S->movgentype == mane)
        P->S->next.principvar = (P->bestpath[Depth+1].movpiece !=
empty);
    else
        P->S->zerowindow = (P->S->maxval >= P->alpha);
REPEATSEARCH:
if (update(P)) return 0;
if (Level == matesearch) /* 无子可动、将死时停止搜索 */
    if ((P->S->nextply <= 0) && !CheckTable[Depth]) goto NOTSEARCH;
if (drawgame(P->S)) goto NOTSEARCH;
if (Depth >= MAXPLY)
    goto NOTSEARCH;
/* 使用递归调用搜索分析下一个回合 */
oldplayer = Player;
Player = Opponent;
Opponent = oldplayer;
Depth++;
if (P->S->zerowindow)
    P->S->next.evaluation = -search(-P->alpha - 1, -P->alpha, P->S-
>nextply,
        &P->S->next, &P->S->path[0]);
else
    P->S->next.evaluation = -search(-P->beta, -P->alpha, P->S->nextply,
        &P->S->next, &P->S->path[0]);
Depth--;
oldplayer = Opponent;

```

```

    Opponent = Player;
    Player = oldplayer;
NOTSEARCH:
    Perform(&MoveTable[Depth], 1); /* 退回一步 */
    if (SkipSearch)
        return 1;

    lastanalysis = Analysis; /* 扫描消息并检查是否设置 SkipSearch */
    MessageScan();
    if (!SkipSearch)
        if (Analysis && !SingleStep && ((!Depth) || !lastanalysis))
        {
            StopTime(&ChessClock);
            if (MainEvaluat > alphawindow)
                SkipSearch = ChessClock.totaltime >= (WantedTime * 1.5);
        }
    if (Analysis && (MaxDepth <= 1))
        SkipSearch = 0;
    P->S->maxval = max(P->S->maxval, P->S->next.evaluation); /* 更改最大
值 */
    IF_EQMOVE(P->bestpath[Depth], MoveTable[Depth]) /* 走子没有变化的话,
更改最好路线 */
        updatebestpath(P);
    if (P->alpha < P->S->maxval) /* 更改α窗口并检查剪枝 */
    {
        updatebestpath(P);
        if (P->S->maxval >= P->beta)
            return 1;
        /* 调整最大值, 在原来最大值的基础上加上一个容忍常数 */
        if (P->ply >= 2 && P->inf->principvar && !P->S->zerowindow)
            P->S->maxval = min(P->S->maxval + TOLERANCE, P->beta - 1);
        P->alpha = P->S->maxval;
        if (P->S->zerowindow && ! SkipSearch)
        {
            /* 全宽度搜索(一个不漏) */
            P->S->zerowindow = 0;
            goto REPEATSEARCH;
        }
    }
    return SkipSearch;

```

```

}
```

10. 函数名称: static short capmovgen(SQUARETYPE newsq, PARAMTYPE *P)

说明: 在新的位置生成吃子移动。

函数定义如下:

```

static short capmovgen(SQUARETYPE newsq, PARAMTYPE *P)
{
    INDEXTYPE i;

    MoveTable[Depth].content = ChessBoard[newsq].piece;
    MoveTable[Depth].spe = 0;
    MoveTable[Depth].newpos1 = newsq;
    for (i = PawnNumber[Player]; i >= 0; i--)
        if (PieceTable[Player][i].ipiece != empty)
            if (PieceAttacks(PieceTable[Player][i].ipiece, Player,
                PieceTable[Player][i].isquare, newsq))
            {
                MoveTable[Depth].oldpos = PieceTable[Player][i].isquare;
                MoveTable[Depth].movpiece = PieceTable[Player][i].ipiece;
                if (loopbody(P)) return 1;
            }
    return 0;
}
```

11. 函数名称: static short noncapmovgen(SQUARETYPE oldsq, PARAMTYPE *P)

说明: 在老的位置生成非吃子移动。

函数定义如下:

```

static short noncapmovgen(SQUARETYPE oldsq, PARAMTYPE *P)
{
    DIRTYPE dir;
    int direction;
    EDGESQUARETYPE newsq;
    int tmpBishopDir, tmpKnightDir;

    MoveTable[Depth].spe = 0;
    MoveTable[Depth].oldpos = oldsq;
    MoveTable[Depth].movpiece = ChessBoard[oldsq].piece;
    MoveTable[Depth].content = empty;
```

```

switch (MoveTable[Depth].movpiece)
{
case king : /*将每次直走一格, 且不能超出九宫*/
    for (dir = 3; dir >= 0; dir--)
    {
        newsq = MoveTable[Depth].oldpos + DirectionTab[dir] ;
        if ((newsq<=0x98) && (newsq>=0))
        if (((newsq>>4)<10) && ((newsq % 16)<9)
            && ((newsq | 0xff)>=0))
        if (ChessBoard[MoveTable[Depth].oldpos].color == red)
        if (( newsq >=0) && ( newsq <= 0x28)
            && ((newsq % 16) >=3) && ((newsq % 16) <=5))
        if (ChessBoard[newsq].piece == empty)
        {
            MoveTable[Depth].newpos1 = newsq;
            if (loopbody(P))
                return 1;
        }
    }
    for (dir = 3; dir >= 0; dir--)
    {
        newsq = MoveTable[Depth].oldpos + DirectionTab[dir] ;
        if ((newsq<=0x98) && (newsq>=0))
        if (((newsq>>4)<10) && ((newsq % 16)<9)
            && ((newsq | 0xff) >=0))
        if (ChessBoard[MoveTable[Depth].oldpos].color == black)
        if (( newsq >= 0x70) && (( newsq ) <= 0x98)
            && ((newsq % 16) >=3) && ((newsq % 16) <=5))
        if (ChessBoard[newsq].piece == empty)
        {
            MoveTable[Depth].newpos1 = newsq;
            if (loopbody(P))
                return 1;
        }
    }
    break;
case assist : /*士每次斜走一格, 且不能超出九宫*/
    for (dir = 3; dir >= 0; dir--)
    {

```



```

    newsq = MoveTable[Depth].oldpos + AssistDir[dir];
    if ((newsq<=0x98) && (newsq>=0))
    if (((newsq>>4)<10) && ((newsq % 16)< 9)
        && ((newsq | 0xff) >=0))
    if (ChessBoard[MoveTable[Depth].oldpos].color == red)
    if (( newsq >=0) && ( newsq <= 0x28)
        && ((newsq % 16) >=3) && ((newsq % 16) <=5))
        if (ChessBoard[newsq].piece == empty)
        {
            MoveTable[Depth].newpos1 = newsq;
            if (loopbody(P))
                return 1;
        }
    } ;
for (dir = 3; dir >= 0; dir--)
    {
        newsq = MoveTable[Depth].oldpos + AssistDir[dir];
        if ((newsq<=0x98) && (newsq>=0))
        if (((newsq>>4)<10) && ((newsq % 16)< 9)
            && ((newsq | 0xff) >=0))
        if (ChessBoard[MoveTable[Depth].oldpos].color == black)
        if (( newsq >= 0x70) && ( newsq <= 0x98)
            && ((newsq % 16) >=3) && ((newsq % 16) <=5))
            if (ChessBoard[newsq].piece == empty)
            {
                MoveTable[Depth].newpos1 = newsq;
                if (loopbody(P))
                    return 1;
            }
    }
break;
case bishop : /*象走田字路, 且田心无棋子*/
    for (dir = 3; dir >= 0; dir--)
    {
        newsq = MoveTable[Depth].oldpos + BishopDir[dir];
        {
            if (BishopDir[dir]== 0x1E)
                tmpBishopDir = 0x0F;
            if (BishopDir[dir]== -0x1E)

```

```

        tmpBishopDir= -0x0F;
    if (BishopDir[dir]== 0x22)
        tmpBishopDir= 0x11;
    if (BishopDir[dir]== -0x22)
        tmpBishopDir= -0x11;
    }
    if ((newsq<=0x98) && (newsq>=0))
    if (((newsq>>4)<10) && ((newsq % 16)< 9)
        && ((newsq | 0xff) >=0))
    if (ChessBoard[MoveTable[Depth].oldpos].color == red)
    if (( ChessBoard[newsq - tmpBishopDir].piece == empty ))
    if (( newsq >=0) && ( newsq <= 0x48))
    if (ChessBoard[newsq].piece == empty)
    {
        MoveTable[Depth].newpos1 = newsq;
        if (loopbody(P))
            return 1;
    }
    }
    for (dir = 3; dir >= 0; dir--)
    {
        newsq = MoveTable[Depth].oldpos + BishopDir[dir];
        {
    if (BishopDir[dir]== 0x1E)
        tmpBishopDir = 0x0F;
    if (BishopDir[dir]== -0x1E)
        tmpBishopDir= -0x0F;
    if (BishopDir[dir]== 0x22)
        tmpBishopDir= 0x11;
    if (BishopDir[dir]== -0x22)
        tmpBishopDir= -0x11;
        }

        if ((newsq<=0x98) && (newsq>=0))
    if (((newsq>>4)<10) && ((newsq % 16)< 9)
        && ((newsq | 0xff) >=0))
    if (ChessBoard[MoveTable[Depth].oldpos].color == black)
    if (((newsq - tmpBishopDir)<=0x98) && ((newsq - tmpBishopDir)>=0))
    if (((newsq - tmpBishopDir)>>4)<10) && (((newsq - tmpBishopDir)
        % 16)< 9)

```

```

        && ((newsq - tmpBishopDir) | 0xff) >= 0))
    if (( ChessBoard[newsq - tmpBishopDir].piece == empty ))
    if (( newsq >= 0x50) && ( newsq <= 0x98))
    if (ChessBoard[newsq].piece == empty)
    {
        MoveTable[Depth].newpos1 = newsq;
        if (loopbody(P))
        return 1;
    }
}
break;

case knight :    /*马走日字路, 且不能有蹩脚子*/
    for (dir = 7; dir >= 0; dir--)
    {
        newsq = MoveTable[Depth].oldpos + KnightDir[dir];

        if ( KnightDir[dir]== 0x0E)
            tmpKnightDir= 0x0F;
        if ( KnightDir[dir]== -0x0E)
            tmpKnightDir= -0x0F;
        if ( KnightDir[dir]== 0x12)
            tmpKnightDir = 0x11;
        if ( KnightDir[dir]== -0x12)
            tmpKnightDir = -0x11;
        if ( KnightDir[dir]== 0x1F)
            tmpKnightDir = 0x0F;
        if (KnightDir[dir]== -0x1F)
            tmpKnightDir = -0x0F;
        if ( KnightDir[dir]== 0x21)
            tmpKnightDir = 0x11;
        if ( KnightDir[dir]== -0x21)
            tmpKnightDir = -0x11;

    if ((newsq<=0x98) && (newsq>=0))
        if (((newsq>>4)<10) && ((newsq % 16)< 9)
            && ((newsq | 0xff) >=0))
        if (((newsq - tmpKnightDir)<=0x98) && ((newsq - tmpKnightDir)>=0))
        if (((newsq - tmpKnightDir)/16)<10) && (((newsq - tmpKnightDir) % 16)<
9)

```

```

        && (((newsq - tmpKnightDir) | 0xff) >= 0))
    if ( ChessBoard[newsq - tmpKnightDir].piece == empty )
        if (ChessBoard[newsq].piece == empty)
        {
            MoveTable[Depth].newpos1 = newsq;
            if (loopbody(P))
                return 1;
        }
    }
    break;

/*车、炮可以连续走子*/
case rook :
case gunner :
    for (dir = 3; dir >= 0; dir--)
    {
        direction = DirectionTab[dir];
        newsq = MoveTable[Depth].oldpos + direction;
        while (((newsq >> 4) < 10) && ((newsq % 16) < 9)
            && (newsq <= 0x98) && (newsq >= 0) && ((newsq | 0xff) >= 0))
        {
            if (ChessBoard[newsq].piece != empty) goto TENS;
            MoveTable[Depth].newpos1 = newsq;
            if (loopbody(P))
                return 1;

            newsq = MoveTable[Depth].newpos1 + direction;
        }
TENS:
        continue;
    }
    break;
case pawn :
    /* 加 0x10 意味着棋子向前走一格 */
    MoveTable[Depth].newpos1 = MoveTable[Depth].oldpos + 0x10 ;
    if ((MoveTable[Depth].newpos1 <= 0x98) && (MoveTable[Depth].
        newpos1 >= 0))
        if (((MoveTable[Depth].newpos1 >> 4) < 10) && ((MoveTable
            [Depth].newpos1 % 16) < 9)

```

```

        && ((MoveTable[Depth].newpos1 | 0xff) >=0))
    if (ChessBoard[MoveTable[Depth].oldpos].color == red)
        if (MoveTable[Depth].oldpos >= 0x30 )
            if (ChessBoard[MoveTable[Depth].newpos1].piece == empty)
                if (loopbody(P))
                    return 1;

/* 加1 意味着棋子向右走一格 */
MoveTable[Depth].newpos1 = MoveTable[Depth].oldpos + 1 ;
    if ((MoveTable[Depth].newpos1<=0x98) && (MoveTable[Depth].
newpos1>=0))
    if (((MoveTable[Depth].newpos1>>4)<10) && ((MoveTable[Depth].
newpos1 % 16)< 9)
        && ((MoveTable[Depth].newpos1 | 0xff) >=0))
    if (ChessBoard[MoveTable[Depth].oldpos].color == red)
        if (MoveTable[Depth].oldpos >= 0x50 )
            if (ChessBoard[MoveTable[Depth].newpos1].piece == empty)
                if (loopbody(P))
                    return 1;

/* 减1 意味着棋子向左走一格 */
MoveTable[Depth].newpos1 = MoveTable[Depth].oldpos - 1 ;
    if ((MoveTable[Depth].newpos1<=0x98) && (MoveTable[Depth].
newpos1>=0))
    if (((MoveTable[Depth].newpos1>>4)<10) && ((MoveTable[Depth].
newpos1 % 16)< 9)
        && ((MoveTable[Depth].newpos1 | 0xff) >=0))
    if (ChessBoard[MoveTable[Depth].oldpos].color == red)
        if (MoveTable[Depth].oldpos >= 0x50 )
            if (ChessBoard[MoveTable[Depth].newpos1].piece == empty)
                if (loopbody(P))
                    return 1;

/* 减0x10 意味着棋子向后走一格 */
MoveTable[Depth].newpos1 = MoveTable[Depth].oldpos - 0x10 ;
    if ((MoveTable[Depth].newpos1<=0x98) && (MoveTable[Depth].
newpos1>=0))
    if (((MoveTable[Depth].newpos1>>4)<10) && ((MoveTable[Depth].
newpos1 % 16)< 9)

```

```

        && ((MoveTable[Depth].newpos1 | 0xff) >=0))
        if (ChessBoard[MoveTable[Depth].oldpos].color == black)
            if (MoveTable[Depth].oldpos <= 0x68 )
                if (ChessBoard[MoveTable[Depth].newpos1].piece == empty)
                    if (loopbody(P))
                        return 1;

        /* 加1意味着棋子向右走一格 */
        MoveTable[Depth].newpos1 = MoveTable[Depth].oldpos + 1 ;
        if ((MoveTable[Depth].newpos1<=0x98) && (MoveTable[Depth].
            newpos1>=0))
            if (((MoveTable[Depth].newpos1>>4)<10) && ((MoveTable[Depth].
                newpos1 % 16)< 9)
                && ((MoveTable[Depth].newpos1 | 0xff) >=0))
                if (ChessBoard[MoveTable[Depth].oldpos].color == black)
                    if (MoveTable[Depth].oldpos <= 0x48 )
                        if (ChessBoard[MoveTable[Depth].newpos1].piece == empty)
                            if (loopbody(P))
                                return 1;

        /* 减1意味着棋子向左走一格 */
        MoveTable[Depth].newpos1 = MoveTable[Depth].oldpos - 1 ;
        if ((MoveTable[Depth].newpos1<=0x98) && (MoveTable[Depth].
            newpos1>=0))
            if (((MoveTable[Depth].newpos1>>4)<10) && ((MoveTable[Depth].
                newpos1 % 16)< 9)
                && ((MoveTable[Depth].newpos1 | 0xff) >=0))
                if (ChessBoard[MoveTable[Depth].oldpos].color == black)
                    if (MoveTable[Depth].oldpos <= 0x48 )
                        if (ChessBoard[MoveTable[Depth].newpos1].piece == empty)
                            if (loopbody(P))
                                return 1;

        } /* switch */
    return 0;
}

```

12. 函数名称: static void searchmovgen(PARAMTYPE *P)

说明: 在分析之前生成下一个移动。按以下顺序生成移动:

主要变化, 吃子, 将军, 常规移动

函数定义如下:

```

static void searchmovgen(PARAMTYPE *P)
{
    INDEXTYPE index;

    /* 从主要变化中生成移动 */
    if (P->bestpath[Depth].movpiece != empty)
    {
        MoveTable[Depth] = P->bestpath[Depth];
        P->S->movgentype = mane;
        if (loopbody(P)) return;
    }
    if (MoveTable[Depth - 1].movpiece != empty)
        if (MoveTable[Depth - 1].movpiece != king)
        {
            P->S->movgentype = specialcap;
            if (capmovgen(MoveTable[Depth - 1].newpos1, P)) return;
        }
    P->S->movgentype = norm1;
    for (index = 0; index <= PawnNumber[Opponent]; index++)
        if (PieceTable[Opponent][index].ipiece != empty)
            if (MoveTable[Depth-1].movpiece == empty ||
                PieceTable[Opponent][index].isquare !=
                MoveTable[Depth-1].newpos1)
                if (capmovgen(PieceTable[Opponent][index].isquare, P))
                    return;

    if (!P->S->capturesearch) /* 非吃子 */
    {
        for (index = PawnNumber[Player]; index >= 0; index--)
            if (PieceTable[Player][index].ipiece != empty)
                if (noncapmovgen(PieceTable[Player][index].isquare, P))
                    return;
    }
}

```

13. 函数名称: static MAXTYPE search(MAXTYPE alpha, MAXTYPE beta, int ply, INFTYPE *inf, MOVETYPE *bestpath)

说明: 执行搜索。

入口:

Player 指下一次移动的玩家
 MoveTable[Depth-1] 表包含上一次移动
 alpha, beta 指示搜索宽度
 ply 意指搜索深度
 inf 包含各种信息
 出口:
 Bestpath 包括最好的变化
 search 包括对玩家的估值

函数定义如下:

```
static MAXTYPE search(MAXTYPE alpha, MAXTYPE beta, int ply, INFTYPE *inf,
    MOVETYPE *bestpath)
{
    SEARCHTYPE S;
    PARAMTYPE P;
    /* 如果 ply <= 0 且没有将军, 执行吃子搜索 */
    S.capturesearch = ((ply <= 0) && !CheckTable[Depth-1]);
    if (S.capturesearch) /* 初始化最大值 */
    {
        S.maxval = -inf->evaluation;
        if (alpha < S.maxval)
        {
            alpha = S.maxval;
            if (S.maxval >= beta) goto STOP;
        }
    }
    else
        S.maxval = -(LOSEVALUE - Depth*DEPTHFACTOR);
    P.alpha = alpha;
    P.beta = beta;
    P.ply = ply;
    P.inf = inf;
    P.bestpath = bestpath;
    P.S = &S;
    searchmovgen(&P); /* 搜索循环 */
    if (SkipSearch) goto STOP;
    if (S.maxval == -(LOSEVALUE - Depth * DEPTHFACTOR)) /* 检查无子可动局面*/
        if (!Attacks(Opponent, PieceTable[Player][0].isquare))
        {
            S.maxval = 0;
        }
    }
```



```
        goto STOP;
    }
STOP:
    return S.maxval;
}
```

14. 函数名称: static MAXTYPE callsearch(MAXTYPE alpha, MAXTYPE beta)

说明: 开始搜索。

函数定义如下:

```
static MAXTYPE callsearch(MAXTYPE alpha, MAXTYPE beta)
{
    MAXTYPE maxval;

    startinf.principvar = (MainPath[0].movpiece != empty);
    LegalMoves = 0;
    maxval = search(alpha, beta, MaxDepth, &startinf, &MainPath[0]);
    if (!LegalMoves)
        MainEvaluat = maxval;
    return maxval;
}
```

15. 函数名称: inline short timeused(void)

说明: 检查搜索时间是否已到。

函数定义如下:

```
inline short timeused(void)
{
    short tu = 0;
    if (Analysis && !SingleStep)
    {
        StopTime(&ChessClock);
        tu = (ChessClock.totaltime >= WantedTime);
    }
    return tu;
}
```

16. 函数名称: void SearchMove(int maxlevel)

说明: 调整搜索。

函数定义如下:

```
void SearchMove(int maxlevel)
{
    MAXTYPE maxval;
    double calcpvtime;
    InitTime(&ChessClock);
    StartTime(&ChessClock);
    InitNode(&Nodes);
    SkipSearch = 0;
    RemoveKillMove();
    CalcPVTable();
    StopTime(&ChessClock);
    calcpvtime = ChessClock.totaltime;
    startinf.value = startinf.evaluation = -RootValue;
    MaxDepth = 0;
    MainPath[0] = ZeroMove;
    MainEvaluat = RootValue;
    alphawindow = MAXINT;
    ComputerThinking = true;
    do
    {
        /* 更改最大值变量 */
        if (MaxDepth <= 1) repeatevalu = MainEvaluat;
        alphawindow = min(alphawindow, MainEvaluat - 0x80);
        if (Level == matesearch)
        {
            alphawindow = 0x6000;
            if (MaxDepth > 0) MaxDepth++;
        }
        MaxDepth++;
        maxval = callsearch(alphawindow, 0x7F00); /* 执行搜索 */
        if (maxval <= alphawindow && !SkipSearch && Level != matesearch &&
            LegalMoves > 0)
        {
            /* 如果局面值低于最大窗口值就重复搜索 */
            MainEvaluat = alphawindow;
            maxval = callsearch(-0x7F00, alphawindow - TOLERANCE * 2);
            LegalMoves = 2;
        }
    }
```

```
    } while (!SkipSearch && !timeused() && (MaxDepth < maxlevel) &&
        (LegalMoves > 1) &&
        (abs(MainEvaluat) < MATEVALUE - 24 * DEPTHFACTOR));
    ComputerThinking = false;
    StopTime(&ChessClock);

    if (Analysis)
        PrintNodes(&Nodes, (ChessClock.totaltime - calcpvtime));
}
```

第 10 章 CHESS 项目非可视化子窗体

采用静态窗体往往占用较多的内存，动态窗体又可能是生硬地将各种不同的功能模式放在一起，而子窗体技术则把一系列不相干的独立的窗体连接起来，不需要弹出来就能使每个窗体一起显现出来，各司其职。

10.1 棋盘编辑子窗体 EditForm

棋盘子窗体 EditForm 是一个独立的单元，由 CHESS 项目的 EditForm2.h 包含文件、EditForm2.cpp 函数源文件和 EditForm2.DFM 窗体文件共同决定，是供用户在编辑棋盘时选择棋子使用。其外形由 CHESS 项目(见 4.2 节)中的 EditForm2.dfm 文件确定，如图 10.1、图 10.2 所示，它能随着父窗体一起移动，表面看起来象是父窗体的一个组件。其功能由 EditForm2.cpp 函数定义的方法决定。

要说明的是，棋盘子窗体的创建不依赖于主窗体，它们是各自独立的，类似于创建静态窗体，仅集成在 CHESS 项目中而已，只是在运行时受主窗体制约(参见第 4 章 4.1.2 节“设计对弈主窗体”中的函数 TMainForm::FormCreate(TObject *Sender))。

10.1.1 EditForm 窗体的属性

EditForm 窗体虽然是动态创建的，但在 CHESS 项目的创建中不依赖于主窗体，仍类似于静态窗体，相当于在项目中添加一个空白新窗体(图 10.1)，其大部分属性采用默认值，如表 10.1 所示，察看属性的具体操作如下：

- (1) 选择 IDE 环境中 File | Open Project 命令，利用弹出的 Open Project 对话框打开 CHESS 项目；
- (2) 选择 View | Project Manager 命令，在弹出的 Project 对话框中选 EditForm2 打开子窗体 EditForm；
- (3) 选择 View | Object Inspector 命令，打开对象观察器，并切换至 Properties 选项卡。

之所以不新设置 EditForm 的属性，是为了配合在主窗体中的显示。EditForm 窗体的属性由如下两个函数设定：

```
void __fastcall TEditForm::Loaded()  
void __fastcall TEditForm::CreateParams(TCreateParams& Params)
```

经过重载后其默认属性将不起作用，没有重载的默认属性仍然起作用。

表 10.1 子窗体 EditForm 的属性

属 性	属性值
Color	clBtnFace
OnMouseUp	FormMouseUp
Name	'EditForm'
OnPaint	FormPaint
PixelsPerInch	96
TextHeight	13

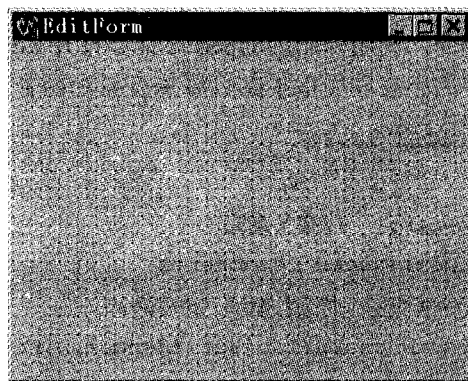


图 10.1 设计时的编辑子窗体 EditForm, 其上没有放任何组件

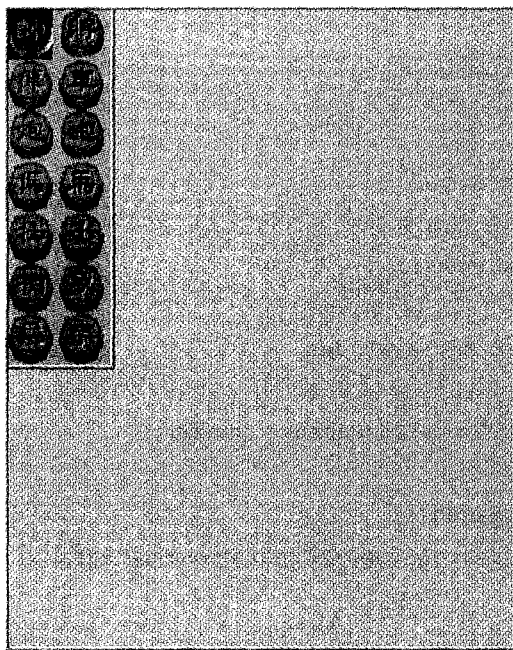


图 10.2 运行时的编辑子窗体 EditForm

10.1.2 EditForm 窗体的方法

通过向 EditForm 窗体中添加方法使得空白的窗体充满灵气(如图 10.2 所示)。这些方法(函数)存放在 CHESS 项目的 EditForm2.cpp 文件中。其声明放在 EditForm2.H 中,是通过在 TEditForm 类中的私有和受保护部分添加用户函数而形成的,扩展了 IDE 自动管理的功能。

具体制作过程如下:

- (1) 选择 File | Open Project 命令, 利用弹出的 Open Project 对话框打开 CHESS 项目;
- (2) 选择 File | Open 命令, 利用弹出的 Open 对话框打开 EditForm2.H;
- (3) 在 TEditForm 类中的私有和受保护部分添加变量和函数, 如下所示:

```
class TEditForm : public TForm
{
    friend class TMainForm;    //定义友元
    __published:              // IDE-managed Components
        void __fastcall FormPaint(TObject *Sender);
        void __fastcall FormMouseUp(TObject *Sender, TMouseButton Button,
            TShiftState Shift, int X, int Y);
    private:                  // User declarations
        int SelectedItem;
        RECT EditBarRect;
        RECT EditRect;
        int GetSelectedItem()
        {
            return SelectedItem;
        }
        void SetEditWindowBk();

    public:                   // User declarations

        __fastcall TEditForm(TComponent* Owner);
    protected:
        void __fastcall Loaded();
        void __fastcall CreateParams(TCreateParams& Params);
};
```

- (4) 单击 View | Project Manager 命令, 在弹出的 Project Manager 对话框中双击 EditForm2 打开 EditForm2.cpp;
- (5) 在 EditForm2.cpp 中手工编写 TEditForm 类的各成员函数。

手工编写的 TEditForm 类的各成员函数如下所示:

1. 函数名称: void __fastcall TEditForm::Loaded()

说明: 在将标准的 VCL 窗体转换为一个子窗体时, 即在 EditForm 还没有显示之前进行一些操作, 就需要重载 Loaded()函数。

重载的具体做法分两步:

首先, 调用基类的 Loaded()函数, 以保证基类的动作被继承。然后, 设置 Form 的一些新属性, 内置函数 SetBounds()重新设置窗体的边框大小。

函数定义如下:

```
void __fastcall TEditForm::Loaded()
{
    TForm::Loaded(); //继承基类的动作
    Visible = false;
    Position = poDefault;
    BorderIcons = TBorderIcons();
    BorderStyle = bsNone;
    Ctl3D = false;
    Color = clMenu;
    HandleNeeded();

    int X = MainForm->Width/2;
    int Y = 0;
    int H = MainForm->Height;
    int W = MainForm->Width/2;

    SetBounds(X, Y, W, H);
}
//-----
```

2. 函数名称: void __fastcall TEditForm::CreateParams(TCreateParams& Params)

说明: 当 VCL 要构造元件的窗口类时, 就会调用 CreateParams()函数。

CreateParams()函数声明如下:

```
virtual void __fastcall CreateParams(TCreateParams& Params);
```

惟一的参数是一个 TCreateParams 结构。TCreateParams 声明如下:

```
struct TCreateParams
{
    char *Caption;
```

```

int Style;
int ExStyle;
int X;
int Y;
int Width;
int Height;
HWND WndParent;
void *Param;
tagWNDCLASSA WindowClass();
char WinClassName[64];
};

```

大多数情况下需要重载 CreateParams()函数，以便改变元件的窗口风格。

具体做法是：

首先，调用基类的 CreateParams()函数，以便继承默认的行为。然后，在原有的窗口风格基础上加上 WS_CHILD | WS_CLIPSIBLINGS。

函数定义如下：

```

void __fastcall TEditForm::CreateParams(TCreateParams& Params)
{
    TForm::CreateParams(Params);
    TForm* owner = dynamic_cast<TForm*>(Owner);
    Params.WndParent = owner->Handle;
    Params.Style = WS_CHILD | WS_CLIPSIBLINGS;
    Params.X = 0;
    Params.Y = 0;

    EditBarRect.right = (SQUARE_SIZE*2) + 6;
    EditBarRect.bottom = (SQUARE_SIZE * 7) + 6;
    EditBarRect.left = EditBarRect.top = 0;

    EditRect.right = MainForm->ClientWidth/2;
    EditRect.bottom = MainForm->ClientHeight;
    EditRect.left = EditRect.top = 0;

    SelectedItem = 0;
}
//-----

```


3. 函数名称: void TEditForm::SetEditWindowBk()

说明: 设置编辑窗体背景。采用文件方式载入位图。对于位图没有保存在资源文件 (.res) 内, C++Builder 不能把从文件装入的位图连接入最后的可执行文件。

因此, 必须确保在 c:\BcbChess 下有必需的位图文件。

很明显, 使用 VCL 函数比原始的 API 函数要方便得多。使用后, 应将 Canvas->Brush->Bitmap 这个特性设为 NULL, 以释放位图。如果不删除位图, 将导致程序内存不足。

函数定义如下:

```
void TEditForm::SetEditWindowBk()
{
    Graphics::TBitmap *BLKBrushBmp = new Graphics::TBitmap();
    BLKBrushBmp->LoadFromFile("c:\\BcbChess\\BKBrush.bmp");
    EditForm->Canvas->Brush->Bitmap=BLKBrushBmp;
    EditForm->Canvas->FillRect(EditForm->GetClientRect());
    EditForm->Canvas->Brush->Bitmap=NULL;
    delete BLKBrushBmp;
}
```

//-----

4. 函数名称: void __fastcall TEditForm::FormPaint(TObject *Sender)

说明: 编辑窗体重画函数。包括对位图进行正常或反相显示, 编辑窗体背景用位图填充。注意, 由于位图采用文件方式载入, 不能链接入可执行文件中, 请确保 c:\BcbChess 下有需要的位图文件。

函数定义如下:

```
void __fastcall TEditForm::FormPaint(TObject *Sender)
{
    int i;
    BITMAP Bitmap;
    HBITMAP hBitmap, hOldBmp, hMaskBmp;
    SetEditWindowBk();

    HDC hDC= GetDC(EditForm->Handle);
    DrawFrame(hDC, EditBarRect, true);
    ReleaseDC(EditForm->Handle, hDC);
    hDC = GetDC(EditForm->Handle);
    DrawFrame(hDC, EditRect);
}
```

```

    ReleaseDC(EditForm->Handle, hDC);
    hDC= GetDC(EditForm->Handle);
    HDC hMemoryDC = CreateCompatibleDC(hDC);
    GetObject(PieceBmpArray[0][red], sizeof(BITMAP), (LPSTR)&Bitmap);

    for (i = 0; i < 7; i++)
    {
        hBitmap = PieceBmpArray[i][red];
        hMaskBmp = MaskArray[i];
        hOldBmp = (HBITMAP)SelectObject(hMemoryDC, hMaskBmp);

        BitBlt(hDC, 3, 3 + (SQUARE_SIZE * i), Bitmap.bmWidth, Bitmap.bmHeight,
            hMemoryDC, 0, 0, SRCAND);
        SelectObject(hMemoryDC, hBitmap);

        BitBlt(hDC, 3, 3 + (SQUARE_SIZE * i), Bitmap.bmWidth, Bitmap.bmHeight,
            hMemoryDC, 0, 0, (SelectedItem == i)
                ? NOTSRCERASE : SRCINVERT);
        SelectObject(hMemoryDC, hOldBmp);
    }

    for (i = 0; i < 7; i++)
    {
        hBitmap = PieceBmpArray[i][black];
        hMaskBmp = MaskArray[i];
        hOldBmp = (HBITMAP)SelectObject(hMemoryDC, hMaskBmp);
        BitBlt(hDC, 3 + SQUARE_SIZE, 3 + (SQUARE_SIZE * i), Bitmap.bmWidth,
            Bitmap.bmHeight, hMemoryDC, 0, 0, SRCAND);
        SelectObject(hMemoryDC, hBitmap);
        BitBlt(hDC, 3+SQUARE_SIZE, 3 + (SQUARE_SIZE * i), Bitmap.bmWidth,
            Bitmap.bmHeight, hMemoryDC, 0, 0, (SelectedItem == (i + 7))
                ? NOTSRCERASE : SRCINVERT);
        SelectObject(hMemoryDC, hOldBmp);
    }

    DeleteDC(hMemoryDC);
    ReleaseDC(EditForm->Handle, hDC);
}

//-----

```

5. 函数名称: void __fastcall TEditForm::FormMouseUp(TObject *Sender, Tmouse Button Button, TShiftState Shift, int X, int Y)

说明: 编辑窗体中单击鼠标左键事件处理函数。单击时位图将反相显示。
函数定义如下:

```
void __fastcall TEditForm::FormMouseUp(TObject *Sender,
    TMouseButton Button, TShiftState Shift, int X, int Y)
{
    POINT point;
    int i, j, temp;
    RECT Rect;
    BITMAP Bitmap;
    HBITMAP hBitmap, hOldBmp, hMaskBmp;
    if (Button != mbLeft)
        return;
    HDC hDC = GetDC(EditForm->Handle);
    HDC hMemoryDC = CreateCompatibleDC(hDC);

    int PreviousItem = SelectedItem;

    point = Point(X,Y);
    EditForm->ScreenToClient(point);

    for (i = 0; i < 7; i++)
        for (j = 0; j < 2; j++)
        {
            Rect.left = (j * SQUARE_SIZE) + 3;
            Rect.right = Rect.left + SQUARE_SIZE;
            Rect.top = (i * SQUARE_SIZE) + 3;
            Rect.bottom = Rect.top + SQUARE_SIZE;
            if (PtInRect(&Rect,point))
                SelectedItem = (i + ((j) ? 7 : 0));
        }
    if (SelectedItem == PreviousItem)
        return;

    GetObject(PieceBmpArray[0][red], sizeof(BITMAP), (LPSTR)&Bitmap);
```

```

temp = PreviousItem % 7;

hBitmap = PieceBmpArray[temp][(PreviousItem < 7) ? red : black];
hMaskBmp = MaskArray[temp];
hOldBmp = (HBITMAP)SelectObject(hMemoryDC, hMaskBmp);
BitBlt(hDC, (PreviousItem < 7) ? 3 : 3 + SQUARE_SIZE,
        3 + (SQUARE_SIZE * temp), Bitmap.bmWidth, Bitmap.bmHeight,
        hMemoryDC, 0, 0, SRCERASE);
SelectObject(hMemoryDC, hBitmap);
BitBlt(hDC, (PreviousItem < 7) ? 3 : 3 + SQUARE_SIZE,
        3 + (SQUARE_SIZE * temp), Bitmap.bmWidth, Bitmap.bmHeight,
        hMemoryDC, 0, 0, SRCINVERT);
SelectObject(hMemoryDC, hOldBmp);

temp = SelectedItem % 7;

hBitmap = PieceBmpArray[temp][(SelectedItem < 7) ? red : black];
hMaskBmp = MaskArray[temp];
hOldBmp = (HBITMAP)SelectObject(hMemoryDC, hMaskBmp);
BitBlt(hDC, (SelectedItem < 7) ? 3 : 3 + SQUARE_SIZE,
        3 + (SQUARE_SIZE * temp), Bitmap.bmWidth, Bitmap.bmHeight,
        hMemoryDC, 0, 0, SRCAND);
SelectObject(hMemoryDC, hBitmap);
BitBlt(hDC, (SelectedItem < 7) ? 3 : 3 + SQUARE_SIZE,
        3 + (SQUARE_SIZE * temp), Bitmap.bmWidth, Bitmap.bmHeight,
        hMemoryDC, 0, 0, NOTSRCERASE);
SelectObject(hMemoryDC, hOldBmp);
DeleteDC(hMemoryDC);
ReleaseDC(EditForm->Handle, hDC);
}
//-----

```

10.2 对弈信息子窗体 InfoForm

对弈信息子窗体 InfoForm 是一个独立的单元，由 CHESS 项目的 InfoForm3.H 包含文件、InfoForm3.CPP 函数源文件和 InfoForm3.DFM 窗体文件共同决定，起着向用户显示对弈信息的作用。其外形由 CHESS 项目(见 4.2 节)中的 InfoForm3.DFM 文件确定，如图 10.3、

图 10.4 所示, 它能随着父窗体一起移动, 表面看起来象是父窗体的一个组件。其功能由 InfoForm3.Cpp 函数定义的方法决定。

要说明的是, 对弈信息子窗体的创建不依赖于主窗体, 类似于创建静态窗体, 它们是各自独立的, 仅集成在 CHESS 项目中而已, 只是在运行时受主窗体制约(参见第 4 章 4.1.2 节“设计对弈主窗体”中的函数 TMainForm::FormCreate(TObject *Sender))。

10.2.1 InfoForm 窗体的属性

虽然子窗体 InfoForm 是动态创建的, 但在 CHESS 项目的创建中不依赖于主窗体, 仍类似于静态窗体, 其上的许多组件是预先就安放好的(见图 10.3), 具体操作如下:

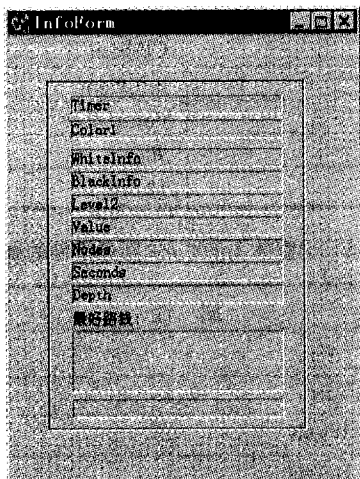


图 10.3 设计时子窗体 InfoForm 的外观

- (1) 选择 IDE 环境中 File | Open Project 命令, 利用弹出的 Open Project 对话框打开 CHESS 项目;
- (2) 选择 View | Project Manager 命令, 在弹出的 Project Manager 对话框中选中 InfoForm3 打开子窗体 InfoForm;
- (3) 选择 View | Object Inspector 命令, 激活对象观察器并切换到 Properties 选项卡, 就可设置其属性, 如表 10.2 所示;
- (4) 单击可视化组件栏 Standard 组件选项卡 Panel 组件, 再在 InfoMain 窗体中单击, 在 InfoMain 窗体中就设计好一个 Panel 组件作为容器;
- (5) 同理单击可视化组件栏 Additional 组件选项卡 StaticText 组件, 再在 InfoMain 窗体中单击, 在 InfoMain 窗体中就设计好一个 StaticText 组件作为输出字符串之用; 单击 Standard 组件选项卡中的 Label 组件设计标签;
- (6) 选择 View | Object Inspector 命令, 激活对象观察器, 并切换到 Properties 选项卡, 就可设置其属性。如表 10.3~表 10.7 所示。

为了配合在主窗体中的显示, 需要新设置 InfoForm 的部分属性。InfoForm 窗体的属性只有少数几个参数(如窗体的大小)是动态创建的, 由如下的两种函数设定:

```
void __fastcall TInfoForm::Loaded()
void __fastcall TInfoForm::CreateParams(TCreateParams& Params)
```

经过重载后其默认属性将不起作用，没有重载的默认属性仍然起作用。

子窗体 InfoForm 的属性值如表 10.2 所示。InfoForm 窗体上还包括其他一些组件：窗格 Panel1、标签 Label1、静态文本框 Color1、Timer、WhiteInfo、Level2、BlackInfo、Value、Seconds、Nodes、Depth、Message、BestLine，它们的属性及属性值如表 10.3~表 10.7 所示。

表 10.2 子窗体 InfoForm 的属性

属 性	属性值
Caption	'InfoForm'
Color	ClMenu
OnPaint	InfoPaint
OnShow	FormShow
PixelsPerInch	96
TextHeight	13

表 10.3 子窗体 InfoForm 部分组件的属性(1)

组 件	属 性	属性值
Panel1	Left	40
	Top	48
	Width	273
	Height	369
	BorderStyle	BsSingle
	Ctl3D	True
	TabOrder	0
Label1	Left	24
	Top	240
	Width	177
	Height	20
	AutoSize	False
	Caption	'最好路线： '
	Font.Charset	GB2312_CHARSET
	Font.Color	ClWindowText
	Font.Height	-16
Color1	Font.Name	'幼圆'
	Left	24
	Top	40

续表

组 件	属 性	属性值
Color1	Width	223
	Height	20
	AutoSize	False
	BorderStyle	SbsSunken
	Caption	Color1
	Font.Charset	GB2312_CHARSET
	Font.Color	ClWindowText
	Font.Height	-16
	Font.Name	幼圆
	TabOrder	0

表 10.4 子窗体 InfoForm 部分组件的属性 (2)

组 件	属 性	属性值
Timer	Left	66
	Top	64
	Width	223
	Height	20
	AutoSize	False
	BorderStyle	SbsSunken
	Caption	Timer
	Font.Color	ClWindowText
	Font.Height	-16
	Font.Name	幼圆
	TabOrder	1
WhiteInfo	Left	66
	Top	120
	Width	223
	Height	20
	AutoSize	False
	BorderStyle	sbsSunken
	Caption	WhiteInfo
	Font.Color	clWindowText
	Font.Height	-16
	Font.Name	幼圆
	TabOrder	2
BlackInfo	Left	66
	Top	144

续表

组 件	属 性	属性值
BlackInfo	Width	223
	Height	20
	AutoSize	False
	BorderStyle	sbsSunken
	Caption	BlackInfo
	Font.Color	clWindowText
	Font.Height	-16
	Font.Name	幼圆
	TabOrder	3

表 10.5 子窗体 InfoForm 部分组件的属性(3)

组 件	属 性	属性值
Level2	Left	66
	Top	168
	Width	223
	Height	20
	AutoSize	False
	BorderStyle	SbsSunken
	Caption	Level2
	Font.Color	ClWindowText
	Font.Height	-16
	Font.Name	幼圆
	TabOrder	4
Value	Left	66
	Top	192
	Width	223
	Height	20
	AutoSize	False
	BorderStyle	SbsSunken
	Caption	Value
	Font.Color	ClWindowText
	Font.Height	-16
	Font.Name	幼圆
	TabOrder	5
Nodes	Left	66
	Top	216
	Width	223
	Height	20
	AutoSize	False

续表

组 件	属 性	属性值
Nodes	BorderStyle	SbsSunken
	Caption	Nodes
	Font.Color	clWindowText
	Font.Height	-16
	Font.Name	幼圆
	TabOrder	6

表 10.6 子窗体 InfoForm 部分组件的属性(4)

组 件	属 性	属性值
Seconds	Left	66
	Top	240
	Width	223
	Height	20
	AutoSize	False
	BorderStyle	sbsSunken
	Caption	Seconds
	Font.Color	clWindowText
	Font.Height	-16
	Font.Name	幼圆
	TabOrder	7
Depth	Left	66
	Top	264
	Width	223
	Height	20
	AutoSize	False
	BorderStyle	sbsSunken
	Caption	Depth
	Font.Color	clWindowText
	Font.Height	-16
	Font.Name	幼圆
	TabOrder	8
Message	Left	66
	Top	384
	Width	223
	Height	20
	AutoSize	False
	BorderStyle	sbsSunken
	Color	clSilver

续表

组 件	属 性	属性值
Message	Font.Color	clWindowText
	Font.Height	-16
	Font.Name	幼圆
	TabOrder	9

表 10.7 子窗体 InfoForm 部分组件的属性(5)

组 件	属 性	属性值
BestLine	Left	66
	Top	312
	Height	65
	AutoSize	False
	BorderStyle	sbsSunken
	Color	clSilver
	Font.Color	clWindowText
	Font.Height	-16
	Font.Name	幼圆
	TabOrder	10

10.2.2 InfoForm 窗体的方法

InfoForm 的这些方法(函数)存放在 CHESS 项目的 InfoForm3.cpp 文件中。其声明放在 InfoForm3.H 中,是通过在 TInfoForm 类中的公有和受保护部分添加变量和函数而形成的,扩展了 IDE 自动管理的功能。

具体制作过程如下:

- (1) 选择 File | Open Project 命令,利用弹出的 Open Project 对话框打开 CHESS 项目;
- (2) 选择 File | Open 命令,利用弹出的 Open 对话框打开 InfoForm3.H;
- (3) 在 TInfoForm 类中的公有和受保护部分添加变量和函数,如下所示:

```
class TInfoForm : public TForm
{
__published:    // IDE-manage
    TStaticText *BestPath;
    TStaticText *Message;
    TStaticText *Timer;
    TStaticText *RedInfo;
    TStaticText *BlackInfo;
    TStaticText *Value;
```

```
TStaticText *Nodes;
TStaticText *Seconds;
TStaticText *Depth;
TStaticText *Level2;
TPanel *Panell;
TLabel *Labell;
TStaticText *Color1;
void __fastcall InfoPaint(TObject *Sender);
void __fastcall FormShow(TObject *Sender);
private:    // User declarations

public:     // User declarations
    RECT InfoRect;
    void SetTurnText(AnsiString text)
    {
        Color1->Caption="轮流: "+AnsiString(text);
    }
    void SetRedInfoText(AnsiString text)
    {
        RedInfo->Caption="红方:"+AnsiString(text);
    }
    void SetBlackInfoText(AnsiString text)
    {
        BlackInfo->Caption="黑方:"+AnsiString(text);
    }
    void SetTimerText(AnsiString text)
    {
        Timer->Caption="时间: "+AnsiString(text);
    }
    void SetLevelText(AnsiString text)
    {
        Level2->Caption="级别: "+AnsiString(text);
    }

    void SetValueText(AnsiString text)
    {
        Value->Caption="值: "+AnsiString(text);
    }
    void SetNodeText(AnsiString text)
```

```

    {
        Nodes->Caption="节点: "+AnsiString(text);
    }
void SetSecondsText(AnsiString text)
{
    Seconds->Caption="步/秒: "+AnsiString(text);
}
void SetDepthText(AnsiString text)
{
    Depth->Caption="深度: "+AnsiString(text);
}
void SetBestPathText(AnsiString text)
{
    BestPath->Caption=text;
}
void SetMessageText(AnsiString text)
{
    Message->Caption=text;
}

void DrawInfoFrame(HDC);
void Reset();
void IterReset();
void SetInfoWindowBk();
__fastcall TInfoForm(TComponent* Owner);
protected:
    void __fastcall Loaded();
    void __fastcall CreateParams(TCreateParams& Params);
};

```

(4) 选择 View | Project Manager 命令, 在弹出的 Project Manager 对话框中双击 InfoForm3 打开 InfotForm3.cpp;

(5) 在 InfoForm3.cpp 中手工编写 TInfoForm 类的各成员函数。

手工编写的 TInfoForm 类的成员函数如下所示:

1. 函数名称: void __fastcall TInfoForm::Loaded()

说明: 在将标准的 VCL 窗体转换为一个子窗体时, 即在 InfoForm 还没有显示之前进行一些操作, 就需要重载 Loaded()函数。

重载的具体做法分两步:

首先, 调用基类的 `Loaded()` 函数, 以保证基类的动作被继承。然后, 设置 `Form` 的一些新属性, 内置函数 `SetBounds()` 重新设置窗体的边框大小。

函数定义如下:

```
void __fastcall TInfoForm::Loaded()
{
    TForm::Loaded(); //继承基类的动作
    Visible = false;
    Position = poDefault;
    BorderIcons = TBorderIcons();
    BorderStyle = bsNone;
    Color=clMenu;
    HandleNeeded();

    int X = MainForm->Width/2;
    int Y = 0;
    int H =MainForm->Height ;
    int W = MainForm->Width/2;

    SetBounds(X,Y,W,H);
}
//-----
```

2. 函数名称: `void __fastcall TInfoForm::CreateParams(TCreateParams& Params)`

说明: 当 VCL 要构造元件的窗口类时, 就会调用 `CreateParams()` 函数。

`CreateParams()` 函数声明如下:

```
virtual void __fastcall CreateParams(TCreateParams& Params);
```

惟一的参数是一个 `TCreateParams` 结构。`TCreateParams` 声明如下:

```
struct TCreateParams
{
    char *Caption;
    int Style;
    int ExStyle;
    int X;
    int Y;
    int Width;
    int Height;
    HWND WndParent;
```

```

void *Param;
tagWNDCLASSA WindowClass();
char WinClassName[64];
};

```

大多数情况下需要重载 CreateParams()函数，以便改变元件的窗口风格。

具体做法是：

首先，调用基类的 CreateParams()函数，以便继承默认的行为。然后，在原有的窗口风格基础上加上 WS_CHILD | WS_CLIPSIBLINGS。

函数定义如下：

```

void __fastcall TInfoForm::CreateParams(TCreateParams& Params)
{
    TForm::CreateParams(Params);
    TForm* owner = dynamic_cast<TForm*>(Owner);
    Params.WndParent = owner->Handle;
    Params.Style = WS_CHILD | WS_CLIPSIBLINGS;
    Params.X = 0;
    Params.Y = 0;
}

//-----

```

3. 函数名称: void TInfoForm::Reset()

说明： 公有自定义函数，初始化静态文本框中的文本。AnsiString 类的文本赋值采用等号。

函数定义如下：

```

void TInfoForm::Reset()
{
    Timer->Caption="时间: ";
    Color1->Caption="红方: ";
    RedInfo->Caption="红方: ";
    BlackInfo->Caption="黑方: ";
    Level2->Caption="级别: ";
    Value->Caption="值: ";
    Nodes->Caption="节点: ";
    Seconds->Caption="步/秒: ";
    Depth->Caption="深度: ";
    BestPath->Caption="";
    Message->Caption="";
}

```

```

}
//-----

```

4. 函数名称: void TInfoForm::SetInfoWindowBk()

说明: 设置信息窗体背景。采用文件方式载入位图。如果位图没有保存在资源文件(.res)内, C++Builder 不能把从文件装入的位图连接入最后的可执行文件。

因此, 必须确保在 c:\BcbChess 下有必需的位图文件。

很明显, 使用 VCL 函数比原始的 API 函数要方便得多。使用后, 应将 Canvas->Brush->Bitmap 这个特性设为 NULL, 以释放位图。如果不删除位图, 将导致程序内存不足。

函数定义如下:

```

void TInfoForm::SetInfoWindowBk()
{
    Graphics::TBitmap *BLKBrushBmp = new Graphics::TBitmap();
    BLKBrushBmp->LoadFromFile("c:\\BcbChess\\BKBrush.bmp");
    InfoForm->Canvas->Brush->Bitmap=BLKBrushBmp;
    InfoForm->Canvas->FillRect(InfoForm->GetClientRect());
    InfoForm->Canvas->Brush->Bitmap=NULL;
    delete BLKBrushBmp;
}

//-----

```

5. 函数名称: void TInfoForm::IterReset()

说明: 公有自定义函数, AnsiString 类的文本赋值采用等号, 非常方便。

函数定义如下:

```

void TInfoForm::IterReset()
{
    Color1->Caption="红方";
    Value->Caption="";
    Nodes->Caption="";
    Seconds->Caption="";
    BestPath->Caption="";
}

//-----

```

6. 函数名称: void __fastcall TInfoForm::InfoPaint(TObject *Sender)

说明: 信息窗体重画函数。其背景用位图填充。注意, 由于位图采用文件方式载入, 不能链接入可执行文件中, 请确保 c:\BcbChess 下有需要的位图文件。

函数定义如下:

```
void __fastcall TInfoForm::InfoPaint(TObject *Sender)
{
    SetInfoWindowBk();
    HDC PaintDC=GetDC(InfoForm->Handle);
    DrawInfoFrame(PaintDC);
    ReleaseDC(InfoForm->Handle, PaintDC);
}
//-----
```

7. 函数名称: void TInfoForm::DrawInfoFrame(HDC hDC)

说明: 公有自定义函数, 绘制信息窗体凸起来的三维边框。

函数定义如下:

```
void TInfoForm::DrawInfoFrame(HDC hDC)
{
    DrawFrame(hDC, InfoRect);
}
//-----
```

8. 函数名称: void __fastcall TInfoForm::FormShow(TObject *Sender)

说明: 信息窗体显现事件处理函数, 窗体显现时获取信息窗体的矩形框大小。

函数定义如下:

```
void __fastcall TInfoForm::FormShow(TObject *Sender)
{
    InfoRect.right = MainForm->ClientWidth/2;
    InfoRect.bottom = MainForm->ClientHeight;
    InfoRect.left = 0 ;
    InfoRect.top = 0 ;
}
//-----
```

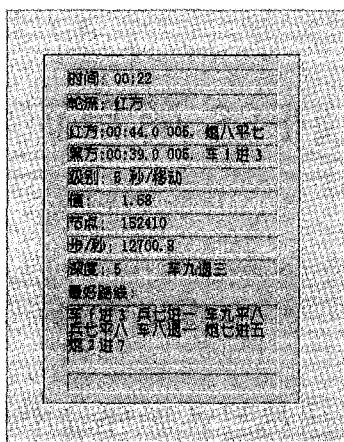



图 10.4 运行时的子窗体 InfoForm

10.3 象棋子窗体的显示

为了显示象棋子窗体, 不要让系统自动创建这些子窗体。为此, 需要打开 Project Options 对话框, 翻到 Form 选项卡, 然后将这些子窗体从列表框移到“Available Forms”列表框中。

必须由主窗体负责创建子窗体, 实现代码放在以下函数中:

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
```

前面两行分别创建两个子窗体, 接下来的两行将这些子窗体放到一个数组中。然后, 通过一个循环依次显示这些子窗体。具体制作过程如下:

- (1) 选择 File | Open Project 命令, 利用弹出的 Open Project 对话框打开 CHESS 项目;
- (2) 选择 View | Project Manager 命令, 在弹出的 Project Manager 对话框中双击 MainForm1 打开 MainForm1.cpp;
- (3) 选择 View | Object Inspector 命令, 激活对象观察器, 并切换到 Events 选项卡, 双击 OnCreate 的回调函数 FormCreate, 添加如下代码:

```
InfoForm = new TInfoForm(this);
EditForm = new TEditForm(this);
ChildForms[1] = EditForm;
ChildForms[2] = InfoForm;
for (int i=1;i<3;i++)
    ChildForms[i]->Show();
```

于是, 当程序运行时, MainForm 作为父窗体, 而 InfoForm 和 EditForm 作为子窗体成为父窗体的一部分。

第 11 章 C++Builder 5.0 项目可视化对话框

C++Builder 5.0 的 VCL 包括众多可重用控件，可以通过设置对象的属性来控制对象的外观，通过利用方法来管理对象。也可以利用已有的组件复合成更复杂的组件。

11.1 棋盘打开保存公用对话框

CHESS 项目进行文件输入输出时，需要选择目录和文件名，这时采用 C++Builder 中提供的公用对话框是最方便简捷的。创建公用打开对话框具体步骤如下：

- (1) 选择 File | Open Project 命令，弹出 Open Project 对话框，打开 CHESS 项目中的 MainForm1 窗体；
- (2) 打开 Dialogs 组件选项卡单击其中的 OpenFileDialog 组件；
- (3) 在主窗体中单击。

于是一个公用打开对话框就出现在主窗体中，默认名称是 OpenFileDialog1。

同样可以创建公用保存对话框，具体操作步骤如下：

- (1) 选择 File | Open Project 命令，弹出 Open Project 对话框，打开 CHESS 项目中的 MainForm1 窗体；
- (2) 打开 Dialogs 组件选项卡单击其中的 SaveDialog 组件；
- (3) 在主窗体中单击。

于是一个公用保存对话框就出现在主窗体中，默认名称是 SaveDialog1。

11.1.1 棋盘打开对话框 OpenFileDialog1 的主要属性

创建了棋盘打开对话框 OpenFileDialog1 后，接下来是设置其属性。棋盘打开对话框 OpenFileDialog1 的内置属性不多(如表 11.1 所示)，较常用的是 Filter 和 FileName 等属性。

表 11.1 棋盘打开对话框主要属性列表

属 性	含 义
DefaultExt	默认的文件扩展名
FileEditStyle	确定对话框风格样式
FileName	包含选中的文件路径和文件名
Files	允许多项选择时返回对话框所选的文件列表
Filter	确定文件的掩码
FilterIndex	确定默认文件过滤器
HistoryList	以前所选文件列表

续表

属 性	含 义
InitialDir	对话框显示时的初始路径
Name	对话框组件名称
Options	对话框风格样式
Title	对话框标题栏文本

设置打开对话框具体步骤如下：

- (1) 选择 File | Open Project 命令，利用弹出的 Open Project 对话框打开 CHESS 项目中的 MainForm1 窗体；
- (2) 选中 OpenFileDialog 组件，单击 Object Inspector，选中 Properties 选项卡；
- (3) 设置 Filter 属性的 Filter Name 为 Chess File(*.chs)，Filter 为 *.chs。

这样设置的目的是保证对扩展名不是 chs 的文件不会显示。如果用户有兴趣的话，可以继续设置其他属性。

11.1.2 棋盘打开对话框 OpenFileDialog 的方法

打开 OpenFileDialog 的方法只有 4 个(如表 11.2 所示)，常用的方法是 Execute。

表 11.2 棋盘打开对话框方法列表

方 法	含 义
~TopenDialog	析构函数。自动调用，删除对话框
Execute	显示对话框
GetStaticRect	保存上次对话框占用的矩形
TopenDialog	构造函数。创建并实例化对话框

CHESS 项目中棋盘打开对话框 OpenFileDialog 的方法很简单，在 CHESS 项目的 MainForm1.cpp 选择【游戏】|【恢复】命令，将引起【恢复】棋盘菜单事件，将调用其内置函数 Execute()。完整的代码如下：

```
void __fastcall TMainForm::RestoreGame1Click(TObject *Sender)
{
    if(OpenDialog1->Execute())
    {
        blNewChessGame = false;
        RestoreGame(MainForm->OpenDialog1->FileName.c_str());
        CurPlayer = Player;
        ComputerColor = Opponent;
        RedoUndo->Enabled = false;
        UndoMove->Enabled = false;
    }
}
```

```

    }
}

```

函数 `OpenDialog1->Execute()` 执行时, 将会模态显示打开对话框, 如图 11.1 所示。用户选择的文件名保存在 `OpenDialog1->FileName.c_str()` 中。

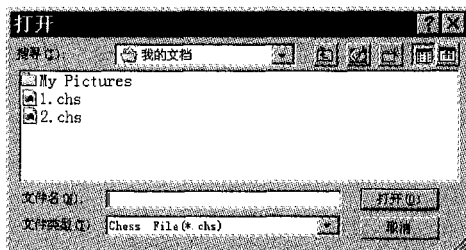


图 11.1 `OpenDialog1` 运行时的状态

11.1.3 棋盘保存对话框 `SaveDialog1` 的主要属性

创建了棋盘保存对话框 `SaveDialog1` 后, 接下来是设置其属性。棋盘保存对话框 `SaveDialog1` 的内置属性不多(如表 11.3 所示), 较常用的是 `Filter`、`FileName` 等属性。

表 11.3 棋盘保存对话框主要属性列表

属 性	含 义
<code>DefaultExt</code>	默认的文件扩展名
<code>FileEditStyle</code>	确定对话框风格样式
<code>FileName</code>	包含选中的文件路径和文件名
<code>Files</code>	允许多项选择时返回对话框所选的文件列表
<code>Filter</code>	确定文件的掩码
<code>FilterIndex</code>	确定默认文件过滤器
<code>InitialDir</code>	对话框显示时的初始路径
<code>Name</code>	对话框组件名称
<code>Options</code>	对话框风格样式
<code>Title</code>	对话框标题栏文本

CHESS 项目中设置保存对话框主要属性具体步骤如下:

- (1) 选择 `File | Open Project` 命令, 利用弹出的 `Open Project` 对话框打开 CHESS 项目中的 `MainForm1` 窗体;
- (2) 选中 `SaveDialog1` 组件单击 `Object Inspector`, 选中 `Properties` 选项卡。
- (3) 设置 `Filter` 属性的 `Filter Name` 为 `Chess File (*.chs)`, `Filter` 为 `*.chs`。这样设置的目的是保证对扩展名不是 `chs` 的文件不会显示。
- (4) 设置属性 `DefaultExt = 'chs'`, 该属性为所输入的文件名自动加上后缀 `chs` 扩展名。如果读者有兴趣的话, 可以继续设置其他属性。

11.1.4 棋盘保存对话框 SaveDialog1 的方法

棋盘保存对话框 SaveDialog1 的方法只有 3 个(如表 11.4 所示),常用的方法是 Execute。

表 11.4 棋盘保存对话框方法列表

方 法	含 义
~TSaveDialog	析构函数。自动调用,删除保存对话框
Execute	显示保存对话框
TSaveDialog	构造函数。创建并实例化保存对话框

CHESS 项目中棋盘保存对话框 SaveDialog1 的方法很简单,在 CHESS 项目的 MainForm1.cpp 中选择【游戏】|【保存】命令,将引发保存棋盘菜单事件,将调用其内置函数 Execute()。完整的代码如下:

```
void __fastcall TMainForm::SaveGame1Click(TObject *Sender)
{
    if (blNewChessGame == true)
        SaveGameAs();
    else
        SaveGame(MainForm->SaveDialog1->FileName.c_str());
}
```

其中的 SaveGameAs()函数定义如下:

```
void TMainForm::SaveGameAs()
{
    if(SaveDialog1->Execute())
    {
        blNewChessGame=false;
        SaveGame(MainForm->SaveDialog1->FileName.c_str());
    }
}
```

棋盘保存对话框 SaveDialog1 的方法 SaveDialog1->Execute()执行时将会模态显示保存对话框,如图 11.2 所示。

用户选择的文件名保存在 SaveDialog1->FileName.c_str()中。

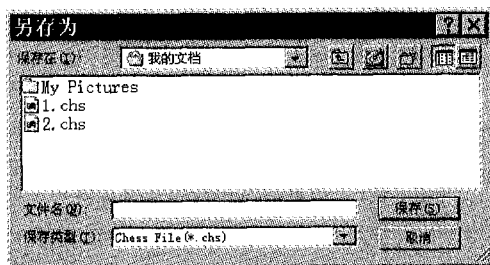


图 11.2 SaveDialog1 运行时的状态

11.2 自定义象棋对话框

C++Builder 的 VCL 类库提供了不少现成的对话框组件,但这些并不能满足所有的要求,有时需要增加一些新的组件,这时就需要利用库中已有的组件来创建新的组件。以下介绍几个对话框的制作。

11.2.1 象棋说明对话框 AboutDlg 的主要属性

象棋说明对话框 AboutDlg 是一个静态对话框(如图 11.3 所示),用户可以按自己的意愿来设计。CHESS 项目的象棋说明对话框 AboutDlg 具体设计步骤如下:

- (1) 选择 File | Open Project 命令,利用弹出的 Open Project 对话框打开 CHESS 项目中的 AboutDlog;
- (2) 打开 Standard 组件选项卡,单击 Panel 组件,再在已建的窗体中单击,就出现一窗格组件。
- (3) 打开 Standard 组件选项卡,单击 Label 组件,再在已建的窗体中单击,就出现一标签组件。
- (4) 打开 Additional 组件选项卡,单击 Image 组件,再在已建的窗体中单击,就出现一图像组件。
- (5) 打开 Standard 组件选项卡,单击 Button 组件,再在已建的窗体中单击,就出现一按钮组件。

当然,要符合自己的愿望,还需要更改各组件的属性。

1. 定义象棋说明对话框 AboutDlg 的总体属性

象棋说明对话框 AboutDlg (如图 11.3 所示)的功能是显示版权信息,主要属性如表 11.5 所示。



图 11.3 AboutDlg 象棋说明对话框设计时的状态

表 11.5 象棋说明对话框的属性

属 性	属性值
Left	115
Top	88
BorderStyle	bsDialog
Caption	关于中国象棋
ClientHeight	318
ClientWidth	278
Color	clBtnFace
Font.Charset	DEFAULT_CHARSET
Font.Color	clWindowText
Font.Height	-11
Font.Name	'Default'
Position	poScreenCenter
PixelsPerInch	96
TextHeight	13

2. 定义象棋说明对话框 AboutDlg 各组件属性

自定义对话框 AboutDlg 中的组件有：窗格组件 Panel1，标签组件 ProductName、Maker、Adress、Copyright、Tele，图像组件 RkingIcon、BkingIcon，按钮组件 OKBut，它们的主要属性如表 11.6~表 11.8 所示。

表 11.6 象棋说明对话框中部分组件的属性(1)

组 件	属 性	属性值
Panel1	Left	24
	Top	16
	Width	225

续表

组 件	属 性	属性值
Panel 1	Height	233
	BevelOuter	BvLowered
	TabOrder	0
ProductName	Left	77
	Top	32
	Width	72
	Height	24
	Alignment	TaCenter
	Caption	中国象棋
	Font.Charset	DEFAULT_CHARSET
	Font.Color	ClWindowText
	Font.Height	-19
	Font.Name	MS Sans Serif
Maker	Left	72
	Top	72
	Width	91
	Height	16
	Alignment	TaCenter
	Caption	制作者: NAME
	Font.Charset	DEFAULT_CHARSET
	Font.Color	ClWindowText
	Font.Height	-13
	Font.Name	MS Sans Serif

表 11.7 象棋说明对话框中部分组件的属性(2)

组 件	属 性	属性值
Adress	Left	69
	Top	120
	Width	96
	Height	16
	Alignment	taCenter
	Caption	地址: ADDRES
	Font.Charset	DEFAULT_CHARSET
	Font.Color	clWindowText
	Font.Height	-13
	Font.Name	MS Sans Serif
RkingIcon	Left	25
	Top	24
	Width	33
	Height	33
Copyright	Left	70

续表

组 件	属 性	属性值
Copyright	Top	192
	Width	101
	Height	16
	Alignment	taCenter
	Caption	Copyright ?2000
	Font.Charset	DEFAULT_CHARSET
	Font.Color	clWindowText
	Font.Height	-13
	Font.Name	MS Sans Serif

表 11.8 象棋说明对话框中部分组件的属性 (3)

组 件	属 性	属性值
Tele	Left	71
	Top	160
	Width	98
	Height	13
	Alignment	taCenter
	Caption	电话:TELEPHONE
	Font.Charset	GB2312_CHARSET
	Font.Color	clWindowText
	Font.Height	-13
	Font.Name	幼圆
BkingIcon	Left	169
	Top	24
	Width	33
	Height	33
	Stretch	True
OKBut	Left	104
	Top	264
	Width	75
	Height	25
	Caption	OK
	TabOrder	1
	OnClick	OKButtonClick

附带说一下, AboutDlg 象棋说明对话框只有一个方法, 当用户单击 OK 按钮时将关闭对话框, 源程序请见 CHESS 项目中的 AboutDlog.cpp, 此处不赘述。

11.2.2 改变棋盘颜色对话框 ColorDlg 的主要属性

改变棋盘颜色对话框 ColorDlg 是一个静态对话框(如图 11.4 所示),用户可以按自己的意愿来设计。

CHESS 项目的改变棋盘颜色对话框 ColorDlg 具体设计步骤如下:

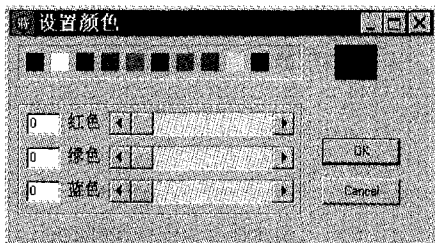


图 11.4 设计时的改变棋盘颜色对话框 ColorDlg

- (1) 选择 File | Open Project 命令, 利用弹出的 Open Project 对话框打开 CHESS 项目中已建 ColorDlg 窗体;
- (2) 打开 Additional 组件选项卡, 单击 Bevel 组件, 再在已建的窗体中单击, 就出现一斜面组件;
- (3) 打开 Standard 组件选项卡, 单击 Label 组件, 再在已建的窗体中单击, 就出现一标签组件;
- (4) 打开 Standard 组件选项卡, 单击 Edit 组件, 再在已建的窗体中单击, 就出现一编辑组件;
- (5) 打开 Standard 组件选项卡, 单击 ScrollBar 组件, 再在已建的窗体中单击, 就出现一滚动组件;
- (6) 打开 Standard 组件选项卡, 单击 Button 组件, 再在已建的窗体中单击, 就出现一按钮组件。

当然, 要符合自己的愿望, 还需要更改各组件的属性。

1. 定义改变棋盘颜色对话框 ColorDlg 总体属性

改变棋盘颜色对话框 ColorDlg(如图 11.4、图 11.5 所示)的功能是供用户输入颜色值以改变棋盘背景色, 主要属性如表 11.9 所示。

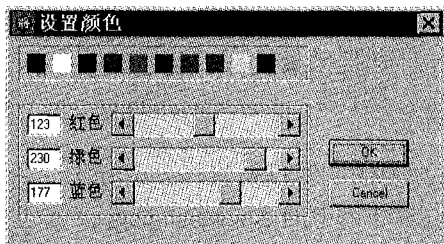


图 11.5 运行时的改变棋盘颜色对话框 ColorDlg

表 11.9 改变棋盘颜色对话框属性

属 性	属性值
Left	273
Top	173
Width	415
Height	227
BorderIcons	[biSystemMenu]
Caption	设置颜色
Color	clBtnFace
Font.Charset	DEFAULT_CHARSET
Font.Color	clWindowText
Font.Height	-11
Font.Name	MS Sans Serif
Position	poMainFormCenter
OnShow	ColorDlgShow
PixelsPerInch	96
TextHeight = 13	13

2. 定义改变棋盘颜色对话框 ColorDlg 各组件属性

改变棋盘颜色对话框 ColorDlg 中的组件有：窗格组件 Bevel1、窗格组件 Bevel2、标签组件 ColorSelection、Label2、Label3、Label4、Label5、Label6、Label7、Label8、Label9、Label11、Label10、Label11、Label12、Label13、Label14，它们的属性如表 11.10~表 11.13 所示；按钮组件 OKBtn、CancelBtn 的属性如表 11.14 所示；滚动组件 ScrollBar_Red、ScrollBar_Green ScrollBar_Blue 的属性如表 11.15 所示；编辑组件 EditRed、EditGreen、EditBlue 的属性如表 11.16 所示。

表 11.10 改变棋盘颜色对话框部分组件属性(1)

组 件	属 性	属性值
Bevel1	Left	8
	Top	64
	Width	273
	Height	105
	Shape	bsFrame
Bevel2	Left	8
	Top	8
	Width	273
	Height	33
ColorSelection	Left	312
	Top	8
	Width	41

续表

组 件	属 性	属性值
Color Selection	Height	33
	AutoSize	False
	Color	clBlack
	ParentColor	False
Label2	Left	56
	Top	72
	Width	33
	Height	17
	AutoSize	False
	Caption	红色
	Font.Color	clWindowText
	Font.Height	-16

表 11.11 改变棋盘颜色对话框部分组件属性(2)

组 件	属 性	属性值
Label3	Left	56
	Top	104
	Width	32
	Height	20
	Caption	绿色
	Font.Charset	DEFAULT_CHARSET
	Font.Color	clWindowText
Label4	Font.Height	-16
	Left	56
	Top	136
	Width	32
	Height	20
	Caption	蓝色
	Font.Charset	DEFAULT_CHARSET
Label5	Font.Color	clWindowText
	Font.Height	-16
	Left	16
	Top	16
	Width	17
	Height	17
Label6	AutoSize	False
	Color	clBlack
	Left	40
	Top	16
	Width	17
	Height	17
	AutoSize	False
	Color	clWhite

表 11.12 改变棋盘颜色对话框部分组件属性(3)

组 件	属 性	属性值
Label7	Left	64
	Top	16
	Width	17
	Height	17
	AutoSize	False
	Color	clMaroon
Label8	Left	88
	Top	16
	Width	17
	Height	17
	AutoSize	False
	Color	clGreen
Label9	Left	112
	Top	16
	Width	17
	Height	17
	AutoSize	False
	Color	ClOlive
Label11	Left	136
	Top	16
	Width	17
	Height	17
	AutoSize	False
	Color	ClPurple
Label10	Left	160
	Top	16
	Width	17
	Height	17
	AutoSize	False
	Color	clTeal

表 11.13 改变棋盘颜色对话框部分组件属性(4)

组 件	属 性	属性值
Label11	Left	184
	Top	16
	Width	17
	Height	17
	AutoSize	False
	Color	clRed
Label12	Left	208

续表

组 件	属 性	属性值
Label12	Top	16
	Width	17
	Height	17
	AutoSize	False
	Color	clYellow
Label3	Left	232
	Top	16
	Width	17
	Height	17
	AutoSize	False
Label4	Color	ClBlue
	Left	256
	Top	16
	Width	17
	Height	17
	AutoSize	False
Label4	Color	clAqua

表 11.14 改变棋盘颜色对话框部分组件属性(5)

组 件	属 性	属性值
OKBtn	Left	300
	Top	96
	Width	75
	Height	25
	Caption	OK
	Default	True
	TabOrder	0
	OnClick	OKClick
CancelBtn	Left	300
	Top	134
	Width	75
	Height	25
	Cancel	True
	Caption	Cancel
	ModalResult	2
	TabOrder	1
CancelBtn	OnClick	CancelBtnClick

表 11.15 改变棋盘颜色对话框部分组件属性(6)

组 件	属 性	属性值
ScrollBar_Red	Left	96
	Top	72

续表

组 件	属 性	属性值
ScrollBar_Red	Width	177
	Height	21
	Max	255
	PageSize	0
	TabOrder	2
	OnChange	ScrollBarChange
ScrollBar_Green	Left	96
	Top	104
	Width	177
	Height	21
	Max	255
	PageSize	0
	TabOrder	3
	OnChange	ScrollBarChange
ScrollBar_Blue	Left	96
	Top	136
	Width	177
	Height	21
	Max	255
	PageSize	0
	TabOrder	4
	OnChange	ScrollBarChange

表 11.16 改变棋盘颜色对话框部分组件属性(7)

组 件	属 性	属性值
EditRed	Left	16
	Top	72
	Width	33
	Height	21
	AutoSelect	False
	AutoSize	False
	Font.Color	clWindowText
	Font.Height	-11
	TabOrder	5
	Text	'0'
	OnExit	EditExit
EditGreen	Left	16
	Top	104
	Width	33
	Height	21
	TabOrder	6
	Text	'0'

续表

组 件	属 性	属性值
EditGreen	OnExit	EditExit
EditBlue	Left	16
	Top	136
	Width	33
	Height	21
	TabOrder	7
	Text	'0'
	OnExit	EditExit

11.2.3 改变棋盘颜色对话框 ColorDlg 的方法

通过向 ColorDlg 窗体中添加方法使得窗体活动起来。这些方法(函数)存放在 CHESS 项目的 ColorDlog.cpp 文件中。其声明放在 ColorDlog.h 中。GetColorValue 是通过在 TColorDlg 类中的私有部分添加用户函数而形成的。而其余函数是通过双击相应组件事件选项卡而形成的(OkClick、CancelBtnClick、ScrollBarChange、ColorDlgShow、EditExit)，它们是 IDE 自动管理形成的，放在__published 部分。

通过添加方法后改变棋盘颜色对话框 ColorDlg 运行状态如图 11.5 所示。

具体制作步骤如下：

- (1) 选择 File | Open Project 命令，利用弹出的 Open Project 对话框打开 CHESS 项目；
- (2) 选择 File | Open 命令，利用弹出的 Open 对话框打开 ColorDlog.h；
- (3) 手工在 TColorDlg 类中的私有部分添加用户函数 GetColorValue，如下所示：

```
class TColorDlg : public TForm
{
__published:    // IDE-managed Components
    TBevel *Bevel2;
    TLabel *Label5;
    TLabel *Label6;
    TLabel *Label7;
    TLabel *Label8;
    TLabel *Label9;
    TLabel *Label1;
    TLabel *Label10;
    TLabel *Label11;
    TLabel *Label12;
    TLabel *Label13;
    TLabel *Label14;
    TBevel *Bevel1;
    TEdit *EditRed;
```



```

    TEdit *EditGreen;
    TEdit *EditBlue;
    TLabel *Label2;
    TLabel *Label3;
    TLabel *Label4;
    TLabel *ColorSelection;
    TScrollBar *ScrollBar_Red;
    TScrollBar *ScrollBar_Green;
    TScrollBar *ScrollBar_Blue;
    TButton *OKBtn;
    TButton *CancelBtn;
    void __fastcall OkClick(TObject *Sender);
    void __fastcall CancelBtnClick(TObject *Sender);
    void __fastcall ScrollBarChange(TObject *Sender);
    void __fastcall ColorDlgShow(TObject *Sender);
    void __fastcall EditExit(TObject *Sender);
private:    // User declarations
    BYTE SqColors[3];
    bool GetColorValue(TObject *Sender);
public:    // User declarations
    __fastcall TColorDlg(TComponent* Owner);
};

```

(4) 选择 View | Project Manager 命令，在弹出的 Project Manager 对话框中双击 ColorDlog 打开 ColorDlog.cpp;

(5) 在打开的文件中手工添加各种函数。

在 ColorDlog.cpp 中手工添加的改变棋盘颜色对话框 ColorDlg 的方法(即函数)如下所示:

1. 函数名称: void __fastcall TColorDlg::ScrollBarChange(TObject *Sender)

说明: 改变棋盘颜色对话框滚动事件处理函数。每个编辑框中的数值随着滚动框的位置改变而变化。

函数定义如下:

```

void __fastcall TColorDlg::ScrollBarChange(TObject *Sender)
{
    EditRed->Text = IntToStr(ScrollBar_Red->Position);
    EditGreen->Text = IntToStr(ScrollBar_Green->Position);
    EditBlue->Text = IntToStr(ScrollBar_Blue->Position);
}

```

```

        ColorSelection->Color = TColor(
            ScrollBar_Red->Position,
            ScrollBar_Green->Position,
            ScrollBar_Blue->Position));
    }
    //-----

```

2. 函数名称: void __fastcall TColorDlg::ColorDlgShow(TObject *Sender)

说明: 改变棋盘颜色对话框显示事件处理函数。初始化每个编辑框中的数值及滚动框的位置。

函数定义如下:

```

void __fastcall TColorDlg::ColorDlgShow(TObject *Sender)
{
    memcpy(SqColors, ColorSquareColors, sizeof(SqColors));

    ScrollBar_Red->Position = SqColors[0];
    EditRed->Text = IntToStr(SqColors[0]);
    ScrollBar_Green->Position = SqColors[1];
    EditGreen->Text = IntToStr(SqColors[1]);
    ScrollBar_Blue->Position = SqColors[2];
    EditBlue->Text = IntToStr(SqColors[2]);
}
//-----

```

3. 函数名称: bool TColorDlg::GetColorValue(TObject *Sender)

说明: 以 try 开头的代码块中的代码被保护了起来。try 代码块之后可以有任意多个包含异常句柄的 catch 代码块。当一个 C++ 应用发出一个异常时, 运行库搜索调用堆栈, 寻找一个类参数与发出的异常中的类型匹配的 catch 代码块。try 代码块之后的 catch 代码块依次被搜索, 直到遇到一次匹配为止。如果未能发现匹配, 用调用函数继续搜索。如果用...作为参数, 可以创建一个与任意异常匹配的 catch 代码块。

如果用户在数量域中敲入一个非数字值, catch (EConvertError &error) 发出整数转换错误。这里用到了指针引用符“&”, 因为 VCL 异常(那些从 Exception 类派生来的异常)必须通过指针引用来被捕捉。当捕捉到了异常后, 显示出错消息, 并把输入聚焦到编辑组件中。

这个代码块不仅捕捉处理错误, 而且在没有将其完全恢复之前将继续弹出错误对话框。

函数定义如下:

```
bool TColorDlg::GetColorValue(TObject *Sender)
{
    int NewVal;
    bool Ok;
    BYTE *CurVal;
    TScrollBar *Scroller;
    TEdit *Sender1 = (TEdit *)Sender;
    Ok = true;
    try
    {
        NewVal = StrToInt(Sender1->Text);
    }
    catch (EConvertError &error)
    {
        Ok=false;
    }

    if (Sender1==EditRed)
    {
        CurVal = &SqColors[0];
        Scroller = ScrollBar_Red;
    }
    else if (Sender1==EditGreen)
    {
        CurVal = &SqColors[1];
        Scroller = ScrollBar_Green;
    }
    else if (Sender1==EditBlue)
    {
        CurVal = &SqColors[2];
        Scroller = ScrollBar_Blue;
    }

    if (Ok)
    {
        Ok = (NewVal > 255) ? false : ( (NewVal < 0 ) ? false : true );
    }
    else
```

```

        NewVal = *CurVal;

        if (!Ok)
        {
            Error("请输入一个介于 0~255 之间的有效数.");
            Sender1->SetFocus();

        }
        else if (NewVal != *CurVal)
        {
            *CurVal = (BYTE)NewVal;
            Scroller->Position=*CurVal;
        }

        ColorSelection->Color = TColor(
            ScrollBar_Red->Position,
            ScrollBar_Green->Position,
            ScrollBar_Blue->Position));
        return Ok;
    }
}
//-----

```

4. 函数名称: void __fastcall TColorDlg::OkClick(TObject *Sender)

说明: 改变棋盘颜色对话框 OK 按钮事件处理函数。每个编辑框中的数值随着滚动框的位置改变而变化。若编辑框中的数值不在 0-255 之间的话, 则出现错误对话框。若正确的话, 不出现错误对话框, 保存输入的数值, 同时, 关闭颜色对话框。

函数定义如下:

```

void __fastcall TColorDlg::OkClick(TObject *Sender)
{
    if( GetColorValue(EditRed) &&
        GetColorValue(EditGreen) && GetColorValue(EditBlue))
    {
        memcpy(ColorSquareColors, SqColors, sizeof(SqColors));
        ColorDlg->Close();
    }
}

```

5. 函数名称: void __fastcall TColorDlg::CancelBtnClick(TObject *Sender)

说明: 颜色对话框 Cancel 按钮事件处理函数。关闭颜色对话框。

函数定义如下:

```
void __fastcall TColorDlg::CancelBtnClick(TObject *Sender)
{
    ColorDlg->Close();
}
//-----
```

6. 函数名称: void __fastcall TColorDlg::EditExit(TObject *Sender)

说明: 颜色对话框中的编辑框退出事件处理函数。若编辑框中的数值不在 0-255 之间的话, 则出现错误对话框。若正确的话, 滚动框的位置随着编辑框中的数值改变而变化。

函数定义如下:

```
void __fastcall TColorDlg::EditExit(TObject *Sender)
{
    if( GetColorValue(EditRed) &&
        GetColorValue(EditGreen) && GetColorValue(EditBlue))
    {
        ScrollBar_Red->Position = StrToInt(EditRed->Text);
        ScrollBar_Green->Position=StrToInt(EditGreen->Text);
        ScrollBar_Blue->Position=StrToInt(EditBlue->Text);
    }
}
//-----
```

11.2.4 改变棋子估值对话框 PieceValueDlg 的主要属性

改变棋子估值对话框 PieceValueDlg 是一个静态对话框(如图 11.6 所示), 用户可以按自己的意愿来设计。CHESS 项目的改变棋子估值 PieceValueDlg 对话框具体设计步骤如下:

- (1) 选择 File|Open Project 命令, 利用弹出的 Open Project 对话框打开 CHESS 项目中已建 PieceValueDlog 窗体;
- (2) 打开 Additional 组件选项卡, 单击 Bevel 组件, 再在已建的窗体中单击, 就出现一斜面组件;
- (3) 打开 Standard 组件选项卡, 单击 Label 组件, 再在已建的窗体中单击, 就出现一标签组件;

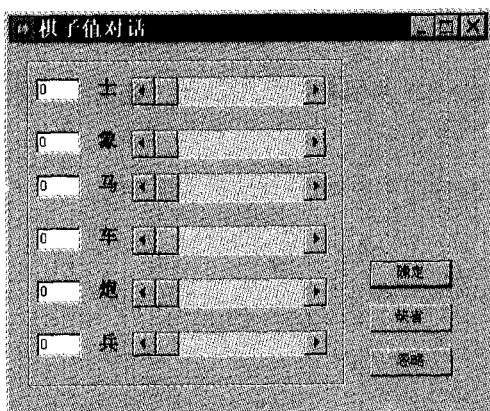


图 11.6 设计时的改变棋子估值对话框 PieceValueDlg

- (4) 打开 Standard 组件选项卡, 单击 Edit 组件, 再在已建的窗体中单击, 就出现一编辑组件;
- (5) 打开 Standard 组件选项卡, 单击 ScrollBar 组件, 再在已建的窗体中单击, 就出现一滚动组件;
- (6) 打开 Standard 组件选项卡, 单击 Button 组件, 再在已建的窗体中单击, 就出现一按钮组件。

当然, 要符合自己的愿望, 还需要更改各组件的属性。

1. 定义改变棋子估值对话框 PieceValueDlg 的总体属性

改变棋子估值对话框 PieceValueDlg(如图 11.6 所示)的功能是供用户输入棋子的评估值, 主要属性如表 11.17 所示。

表 11.17 改变棋子估值对话框属性

属 性	属性值
Left	247
Top	77
Width	445
Height	369
BorderIcons	[biSystemMenu]
Caption	棋子值对话
Color	ClBtnFace
Font.Charset	DEFAULT_CHARSET
Font.Color	ClWindowText
Font.Height	-11
Font.Name	MS Sans Serif
OldCreateOrder	False
Position	PoMainFormCenter

续表

属 性	属性值
OnShow	FormShow
PixelsPerInch	96
TextHeight	13

2. 定义改变棋子估值对话框 PieceValueDlg 各组件属性

改变棋子估值对话框 PieceValueDlg 包括其他一些组件：窗格组件 Bevel1、标签组件 Label3、Label4、Label5、Label6、Label7、Label8 等，它们的属性如表 11.18~表 11.19 所示；改变棋子估值对话框 PieceValueDlg 包括按钮组件 OKBtn、CancelBtn、Default 等，它们的属性如表 11.20 所示；改变棋子估值对话框 PieceValueDlg 包括滚动框组件 Srook、Sknight、Sbishop、Sassist、Sgunner 等，它们的属性如表 11.21~表 11.22 所示；改变棋子估值对话框 PieceValueDlg 包括编辑框组件 Eassist、Ebishop、Eknight、Erook、Egunner、Epawn 等，它们的属性如表 11.23~表 11.24 所示。

表 11.18 改变棋子估值对话框部分组件属性(1)

组 件	属 性	属性值
Bevel1	Left	16
	Top	16
	Width	289
	Height	297
	Shape	BsFrame
Label3	Left	80
	Top	32
	Width	20
	Height	24
	Caption	士
Label4	Left	80
	Top	80
	Width	20
	Height	24
	Caption	象
Label5	Left	80
	Top	120
	Width	20
	Height	24
	Caption	马
Label6	Left	80
	Top	168
	Width	20
	Height	24
	Caption	车

表 11.19 改变棋子估值对话框部分组件属性(2)

组 件	属 性	属性值
Label7	Left	80
	Top	216
	Width	20
	Height	24
	Caption	炮
Label8	Left	80
	Top	264
	Width	20
	Height	24
	Caption	兵

表 11.20 改变棋子估值对话框部分组件属性(3)

组 件	属 性	属性值
OKBtn	Left	328
	Top	200
	Width	75
	Height	25
	Caption	确定
	Default	True
	TabOrder	0
	OnClick	OKBtnClick
CancelBtn	Left	328
	Top	280
	Width	75
	Height	25
	Cancel	True
	Caption	忽略
	TabOrder	2
	OnClick	CancelBtnClick
Default	Left	328
	Top	240
	Width	75
	Height	25
	BiDiMode	BdRightToLeft
	Caption	默认
	TabOrder	1
	OnClick	CmDefault

表 11.21 改变棋子估值对话框部分组件属性(4)

组 件	属 性	属性值
Srook	Left	112
	Top	168
	Width	177
	Height	25
	Max	255
	PageSize	0
	TabOrder	12
	OnChange	ScrollBar_Change
Sknight	Left	112
	Top	120
	Width	177
	Height	25
	Max	255
	PageSize	0
	TabOrder	11
	OnChange	ScrollBar_Change
Sbishop	Left	112
	Top	80
	Width	177
	Height	25
	Max	255
	PageSize	0
	TabOrder	10
	OnChange	ScrollBar_Change

表 11.22 改变棋子估值对话框部分组件属性(5)

组 件	属 性	属性值
Sassist	Left	112
	Top	32
	Width	177
	Height	25
	Max	255
	PageSize	0
	TabOrder	9
	OnChange	ScrollBar_Change
Sgunner	Left	112
	Top	216
	Width	177
	Height	25
	Max	255
	PageSize	0

续表

组 件	属 性	属性值
Sgunner	TabOrder	13
	OnChange	ScrollBar_Change

表 11.23 改变棋子估值对话框部分组件属性(6)

组 件	属 性	属性值
Eassist	Left	24
	Top	32
	Width	41
	Height	21
	TabOrder	3
	Text	0
	OnExit	EditExit
Ebishop	Left	24
	Top	80
	Width	41
	Height	21
	TabOrder	4
	Text	0
	OnExit	EditExit
Eknight	Left	24
	Top	120
	Width	41
	Height	21
	TabOrder	5
	Text	0
	OnExit	EditExit
Erook	Left	24
	Top	168
	Width	41
	Height	21
	TabOrder	6
	Text	0
	OnExit	EditExit

表 11.24 改变棋子估值对话框部分组件属性(7)

组 件	属 性	属性值
Egunner	Left	24
	Top	216
	Width	41
	Height	21

续表

组 件	属 性	属性值
Egunner	TabOrder	7
	Text	0
	OnExit	EditExit
Spawn	Left	112
	Top	264
	Width	177
	Height	21
	Max	255
	PageSize	0
	TabOrder	14
	OnChange	ScrollBar_Change
Epawn	Left	24
	Top	264
	Width	41
	Height	21
	TabOrder	8
	Text	0
	OnExit	EditExit

11.2.5 改变棋子估值对话框 PieceValueDlg 的方法

通过向 PieceValueDlg 窗体中添加方法使得窗体活动起来。这些方法(函数)存放在 CHESS 项目的 PieceValueDlog.cpp 文件中。其声明放在 PieceValueDlog.H 中。GetColorValue 是通过在 TPieceValueDlg 类中的私有部分添加用户函数而形成的。而其余函数是通过双击相应组件事件选项卡而形成的(ScrollBar_Change、EditExit、FormShow、OKBtnClick、CmDefault、CancelBtnClick)，它们是 IDE 自动管理形成的，放在 __published 部分。

通过添加方法后棋子值对话框 PieceValueDlg 运行状态如图 11.7 所示。

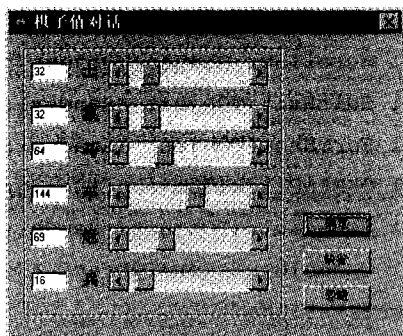


图 11.7 运行时的改变棋子估值对话框 PieceValueDlg

具体制作步骤如下:

- (1) 选择 File | Open Project 命令, 利用弹出的 Open Project 对话框打开 CHESS 项目;
- (2) 选择 File | Open 命令, 利用弹出的 Open 对话框打开 PieceValueDlg.h;
- (3) 手工在 TPieceValueDlg 类中的私有部分添加用户函数 GetColorValue, 如下所示:

```
class TPieceValueDlg : public TForm
{
__published:    // IDE-managed Components
    TButton *OKBtn;
    TButton *CancelBtn;
    TButton *Default;
    TBevel *Bevel1;
    TScrollBar *Srook;
    TScrollBar *Sknight;
    TScrollBar *Sbishop;
    TScrollBar *Sassist;
    TScrollBar *Sgunner;
    TScrollBar *Spawn;
    TEdit *Eassist;
    TEdit *Ebishop;
    TEdit *Eknight;
    TEdit *Erook;
    TEdit *Egunner;
    TEdit *Epawn;
    TLabel *Label3;
    TLabel *Label4;
    TLabel *Label5;
    TLabel *Label6;
    TLabel *Label7;
    TLabel *Label8;

    void __fastcall ScrollBar_Change(TObject *Sender);
    void __fastcall EditExit(TObject *Sender);
    void __fastcall FormShow(TObject *Sender);
    void __fastcall OKBtnClick(TObject *Sender);
    void __fastcall CmDefault(TObject *Sender);
    void __fastcall CancelBtnClick(TObject *Sender);
private:    // User declarations
    int Values[6];
    bool GetColorValue(TObject *Sender);
```

```
public:    // User declarations
    __fastcall TPieceValueDlg(TComponent* Owner);
};
```

(4) 选择 View | Project Manager 命令, 在弹出的 Project Manager 对话框中双击 PieceValueDlg 打开 PieceValueDlg.cpp;

(5) 在打开的文件中手工添加各种函数。

在 PieceValueDlg.cpp 中手工添加的改变棋子估值对话框 PieceValueDlg 的方法(函数)如下所示:

1. 函数名称: bool TPieceValueDlg::GetColorValue(TObject *Sender)

说明: 以 try 开头的代码块中的代码被保护了起来。try 代码块之后可以有任意多个包含异常句柄的 catch 代码块。当一个 C++ 应用发出一个异常时, 运行库搜索调用堆栈, 寻找一个类参数与发出的异常中的类型匹配的 catch 代码块。try 代码块之后的 catch 代码块依次被搜索, 直到遇到一次匹配为止。如果未能发现匹配, 用调用函数继续搜索。如果用...作为参数, 可以创建一个与任意异常匹配的 catch 代码块。

如果用户在数量域中敲入一个非数字值, catch (EConvertError &error) 发出整数转换错误。这里用到了指针引用符 “&”, 因为 VCL 异常(那些从 Exception 类派生来的异常)必须通过指针引用来被捕捉。当捕捉到了异常后, 显示出错消息, 并把输入聚焦到编辑组件中。

这个代码块不仅捕捉处理错误, 而且在没有将其完全恢复之前将继续弹出错误对话框。

函数定义如下:

```
bool TPieceValueDlg::GetColorValue(TObject *Sender)
{
    int NewVal;
    bool Ok;
    int *CurVal;
    TScrollBar *Scroller;
    TEdit *Sender1 = (TEdit *)Sender;
    Ok = true;
    try
    {
        NewVal = Sender1->Text.ToInt();
    }
    catch (EConvertError &error)
    {
```

```
Ok=false;
}
if (Sender1==Eassist)
{
    CurVal = &Values[pvassist];
    Scroller = Sassist;
}
else if (Sender1==Ebishop)
{
    CurVal = &Values[pvbishop];
    Scroller = Sbishop;
}
else if (Sender1==Eknight)
{
    CurVal = &Values[pvknight];
    Scroller = Sknight;
}
else if (Sender1==Erook)
{
    CurVal = &Values[pvrook];
    Scroller = Srook;
}
else if (Sender1==Egunner)
{
    CurVal = &Values[pvgunner];
    Scroller = Sgunner;
}
else if (Sender1==Epawn)
{
    CurVal = &Values[pvpawn];
    Scroller = Spawn;
}

if (Ok)
{
    Ok = (NewVal > 255) ? false : ( (NewVal < 0) ? false : true );
}
else
    NewVal = *CurVal;
```

```

    if (!Ok)
    {
        Error("请输入一个介于 0-255 之间的有效数.");
        Sender1->SetFocus();
    }
    else if (NewVal != *CurVal)
    {
        *CurVal = NewVal;
        Scroller->Position=*CurVal;
    }
    return Ok;
}
//-----

```

2. 函数名称: void __fastcall TPieceValueDlg::FormShow(TObject *Sender)

说明: 改变棋子估值对话框显示事件处理函数。初始化每个编辑框中的数值及滚动框的位置。

函数定义如下:

```

void __fastcall TPieceValueDlg::FormShow(TObject *Sender)
{
    Values[pvassist] = PieceValue[assist] / 16;
    Values[pvbishop] = PieceValue[bishop] / 16;
    Values[pvknight] = PieceValue[knight] / 16;
    Values[pvrook] = PieceValue[rook] / 16;
    Values[pvgunner] = PieceValue[gunner] / 16;
    Values[pvpawn] = PieceValue[pawn] / 16;

    Sassist->Position = Values[pvassist];
    Eassist->Text = IntToStr(Values[pvassist]);
    Sbishop->Position = Values[pvbishop];
    Ebishop->Text = IntToStr(Values[pvbishop]);

    Sknight->Position = BYTE(Values[pvknight]);
    Eknight->Text = IntToStr(Values[pvknight]);
    Srook->Position = Values[pvrook];
    Erook->Text = IntToStr(Values[pvrook]);
    Sgunner->Position = Values[pvgunner];
    Egunner->Text = IntToStr(Values[pvgunner]);
}

```

```

    Spawn->Position = Values[pvpawn];
    Epawn->Text = IntToStr(Values[pvpawn]);

}
//-----

```

3. 函数名称: void __fastcall TPieceValueDlg::OKBtnClick(TObject *Sender)

说明: 改变棋子估值对话框确定按钮事件处理函数。设置好每个编辑框中的数值及滚动框的位置。若编辑框中的数值不在 0-255 之间的话, 则出现错误对话框。若正确的话, 不出现错误对话框, 保存输入的数值, 同时, 关闭棋子值对话框。

函数定义如下:

```

void __fastcall TPieceValueDlg::OKBtnClick(TObject *Sender)
{
    if ( GetColorValue(Eassist) && GetColorValue(Ebishop)
        && GetColorValue(Eknight) && GetColorValue(Erook) &&
        GetColorValue(Egunner) && GetColorValue(Epawn))
    {
        PieceValue[assist] = Values[pvassist] * 16;
        PieceValue[bishop] = Values[pvbishop] * 16;
        PieceValue[knight] = Values[pvknights] * 16;
        PieceValue[rook] = Values[pvrook] * 16;
        PieceValue[gunner] = Values[pvgunner] * 16;
        PieceValue[pawn] = Values[pvpawn] * 16;
        PieceValueDlg->Close();
    }
}
//-----

```

4. 函数名称: void __fastcall TPieceValueDlg::CmDefault(TObject *Sender)

说明: 改变棋子估值对话框默认按钮事件处理函数。输入默认数值, 并初始化每个编辑框中的默认数值及滚动框的默认位置。

函数定义如下:

```

void __fastcall TPieceValueDlg::CmDefault(TObject *Sender)
{
    memcpy(Values, DefaultValues, 6 * sizeof(int));

    Eassist->Text = IntToStr(Values[pvassist]);
}

```



```

Ebishop->Text = IntToStr(Values[pvbishop]);
Eknight->Text = IntToStr(Values[pvknight]);
Erook->Text = IntToStr(Values[pvrook]);
Egunner->Text = IntToStr(Values[pvgunner]);
Epawn->Text = IntToStr(Values[pvpawn]);

Sassist->Position = Values[pvassist];
Sbishop->Position = Values[pvbishop];
Sknight->Position = Values[pvknight];
Srook->Position = Values[pvrook];
Sgunner->Position = Values[pvgunner];
Spawn->Position = Values[pvpawn];
}
//-----

```

5. 函数名称: void __fastcall TPieceValueDlg::CancelBtnClick(TObject *Sender)

说明: 改变棋子估值对话框忽略按钮事件处理函数。关闭棋子值对话框。
函数定义如下:

```

void __fastcall TPieceValueDlg::CancelBtnClick(TObject *Sender)
{
    PieceValueDlg->Close();
}
//-----

```

6. 函数名称: void __fastcall TPieceValueDlg::EditExit(TObject *Sender)

说明: 改变棋子估值对话框中的编辑框退出事件处理函数。若编辑框中的数值不在 0-255 之间的话, 则出现错误对话框。若正确的话, 滚动框的位置随着编辑框中的数值改变而变化。

函数定义如下:

```

void __fastcall TPieceValueDlg::EditExit(TObject *Sender)
{
    if ( GetColorValue(Eassist) && GetColorValue(Ebishop)
        && GetColorValue(Eknight) && GetColorValue(Erook) &&
        GetColorValue(Egunner) && GetColorValue(Epawn))
    {
        Sassist->Position = StrToInt(Eassist->Text);
        Sbishop->Position = StrToInt(Ebishop->Text);
        Sknight->Position = StrToInt(Eknight->Text);
    }
}

```

```

    Srook->Position = StrToInt(Erook->Text);
    Sgunner->Position = StrToInt(Egunner->Text);
    Spawn->Position = StrToInt(Epawn->Text);
}
}
//-----

```

7. 函数名称: void __fastcall TPieceValueDlg::ScrollBar_Change(TObject *Sender)

说明: 改变棋子估值对话框滚动事件处理函数。每个编辑框中的数值随着滚动框的位置改变而变化。

函数定义如下:

```

void __fastcall TPieceValueDlg::ScrollBar_Change(TObject *Sender)
{
    TEdit *Edit1 ;
    TScrollBar *Sender1 = (TScrollBar *)Sender;

    if (Sender1==Sassist)
    {
        Edit1 = Eassist;
        Values[pvassist] = Sender1->Position;
    }
    else if (Sender1==Sbishop)
    {
        Edit1 = Ebishop;
        Values[pvbishop] = Sender1->Position;
    }
    else if (Sender1==Sknight)
    {
        Edit1 = Eknight;
        Values[pvknight] = Sender1->Position;
    }
    else if (Sender1==Srook)
    {
        Edit1 = Erook;
        Values[pvrook] = Sender1->Position;
    }
    else if (Sender1==Sgunner)
    {

```

```

        Edit1 = Egunner;
        Values[pvgunner] = Sender1->Position;
    }
    else if (Sender1==Spawn)
    {
        Edit1 = Epawn;
        Values[pvpawn] = Sender1->Position;
    }

    Edit1->Text = IntToStr(Sender1->Position);

}
//-----

```

11.2.6 定义对局时间对话框 InputMinPerGameDlg 的主要属性

对局时间对话框 InputMinPerGameDlg 是一个静态对话框，读者可以按自己的意愿来设计(如图 11.8 所示)。CHESS 项目的定义对局时间 InputMinPerGameDlg 对话框具体设计步骤如下：

- (1) 选择 File | Open Project 命令，利用弹出的 Open Project 对话框打开 CHESS 项目中已建 InputMinPerGame 窗体；
- (2) 打开 Standard 组件选项卡，单击 Panel 组件，再在已建的窗体中单击，就出现一窗格组件；
- (3) 打开 Additional 组件选项卡，单击 Bevel 组件，再在已建的窗体中单击，就出现一斜面组件；
- (4) 打开 Standard 组件选项卡，单击 Label 组件，再在已建的窗体中单击，就出现一标签组件；
- (5) 打开 Standard 组件选项卡，单击 Edit 组件，再在已建的窗体中单击，就出现一编辑组件；
- (6) 打开 Standard 组件选项卡，单击 Button 组件，再在已建的窗体中单击，就出现一按钮组件。

当然，要符合自己的愿望，还需要更改各组件的属性。

1. 定义对局时间对话框总体属性

对局时间(分钟)值对话框 InputMinPerGameDlg(如图 11.8 所示)的功能是供用户输入每次游戏的时间(分钟)值，主要属性如表 11.25 所示。

表 11.25 定义对局时间对话框属性

属 性	属性值
Left	425
Top	275
BorderIcons	[biSystemMenu]
BorderStyle	bsDialog
Caption	'设置每次游戏分钟数'
ClientHeight	168
ClientWidth	313
Color	clBtnFace
ParentFont	True
OldCreateOrder	True
Position	poScreenCenter
PixelsPerInch	96
TextHeight	13

2. 定义对局时间对话框各组件属性

对局时间(分钟)值对话框 InputMinPerGameDlg 还包括: 窗格组件 Bevel1, LABEL 组件 Label1, 按钮组件 OKBtn、CancelBtn, 窗格组件 Panel1, 编辑组件 Edit1 等, 它们的属性如表 11.26~表 11.27 所示。

表 11.26 定义对局时间部分组件属性(1)

组 件	属 性	属性值
Bevel1	Left	0
	Top	112
	Width	313
	Height	1
	Shape	BsFrame
	Style	BsRaised
Label1	Left	24
	Top	24
	Width	265
	Height	24
	AutoSize	False
	Caption	每次游戏分钟数:
	Color	ClGray
OKBtn	Left	63
	Top	132
	Width	75

续表

组 件	属 性	属性值
OKBtn	Height	25
	Caption	确定
	Default	True
	ModalResult	1
	TabOrder	0
CancelBtn	Left	167
	Top	132
	Width	75
	Height	25
	Cancel	True
	Caption	忽略
	ModalResult	2
	TabOrder	1

要说明的是，按钮 OKBtn 和 CancelBtn 的属性 ModalResult 的取值就是单击该按钮的返回值，很重要。

表 11.27 定义对局时间部分组件属性 (2)

组 件	属 性	属性值
Panel1	Left	24
	Top	66
	Width	265
	Height	30
	BevelOuter	BvLowered
	Caption	Panel1
	TabOrder	3
Edit1	Left	32
	Top	70
	Width	249
	Height	21
	AutoSelect	False
	AutoSize	False
	TabOrder	2
	Text	5.00

11.2.7 定义对局时间对话框 InputMinPerGameDlg 的方法

我们可以按照前面几节介绍的制作改变颜色对话框或改变棋子值对话框的方式来定义对局时间 InputMinPerGameDlg 的方法，但这里我们采用另一种方法来达到目的。即显示

一个静态对话框时,调用其内置方法 `ShowModal` 即可,这样不需要用户再行定义其方法。方法 `ShowModal` 表示如果对话框不关闭,则用户将不能操作其父窗口。要说明的是方法 `Show` 没有此限制,但调用 `Show` 函数不会有 `ModalResult` 返回值,而且还要为每个按钮定义事件回调函数。`ShowModal` 函数运行后将使定义对局时间对话框显示,按钮的返回值 `ModalResult` 可有以下取值:

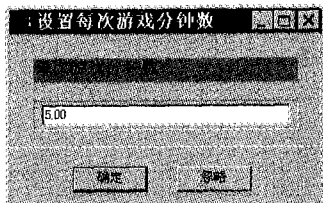


图 11.8 InputMinPerGameDlg 对话框设计时状态

`mrNone`、`mrOk`、`mrCancel`、`mrAbort`、`mrRetry`、`mrYes`、`mrNo`、`mrAll`、`mrYesToAll`、`mrNoToAll`、`mrIgnore`。在本程序中,我们已设定 `InputMinPerGame` 中的“确定”按钮的属性返回值为 `mrOk` 或 1、“忽略”按钮的属性返回值为 `mrCancel` 或 2。我们可以根据 `ModalResult` 的取值来完成不同的功能。

通过以下步骤可以察看 `InputMinPerGame.cpp` 中的代码:

- (1) 选择 `File | Open Project` 命令,利用弹出的 `Open Project` 对话框打开 `CHESS` 项目;
- (2) 选择 `View | Project Manager` 命令,在弹出的 `Project Manager` 对话框中双击 `InputMinPerGame` 打开 `InputMinPerGame.cpp`。

可以看到 `InputMinPerGame.cpp` 中没有用户添加的函数,全由 IDE 自动管理生成。

在 `CHESS` 项目中主窗体【级别】|【总时间】菜单事件将调用 `InputMinPerGameDlg->ShowModal()` 方法,代码如下:

```
void __fastcall TMainForm::TotalTimeClick(TObject *Sender)
{
    char *TotalMoveTime = new char[40];
    int temp;
    COLORTYPE color;
    sprintf(TotalMoveTime, "%.2f", AverageTime);

    if (InputMinPerGameDlg->ShowModal() == mrOk)
    {
        TotalMoveTime = InputMinPerGameDlg->Edit1->Text.c_str();
        double NewMoveTime = atof(TotalMoveTime);
        if (NewMoveTime > 0.)
        {
            if (Level != fullgametime)
            {
                UnCheckLevelMenu(Level);
            }
        }
    }
}
```

```

    CheckMenuItem(TChessMenu->Handle, TotalTime->Command, MF_CHECKED);
}

Level = fullgametime;
AverageTime = NewMoveTime;
PrintCurLevel();

for (temp=0;temp<=1;temp++)
{
    { if (temp==0)
        color = red;
        if (temp==1)
            color= black;
        ChessTime[color].totaltime = (MoveNumber / 2) * AverageTime;
    }
}

MaxLevel = MAXPLY;
}

else
    Error("时间不正确。请重输。");
}

delete TotalMoveTime;

}

```

要引起注意的是，以上并没有定义每个按钮的事件回调函数，而是采用按钮的 `ModalResult` 返回值，这样使编程更具灵活性、简明性。

11.2.8 定义走步时间对话框 `InputSecPerMoveDlg` 的主要属性

走步时间对话框 `InputSecPerMoveDlg` 是一个静态对话框(如图 11.9 所示)，读者可以按自己的意愿来设计。CHESS 项目的定义走步时间 `InputSecPerMoveDlg` 对话框具体设计步骤如下：

- (1) 选择 `File | Open Project` 命令，利用弹出的 `Open Project` 对话框打开 CHESS 项目中已建 `InputSecPerMove` 窗体；
- (2) 打开 `Standard` 组件选项卡，单击 `Panel` 组件，再在已建的窗体中单击，就出现一窗格组件；
- (3) 打开 `Additional` 组件选项卡，单击 `Bevel` 组件，再在已建的窗体中单击，就出现一斜面组件；
- (4) 打开 `Standard` 组件选项卡，单击 `Label` 组件，再在已建的窗体中单击，就出现一标签组件；

- (5) 打开 Standard 组件选项卡, 单击 Edit 组件, 再在已建的窗体中单击, 就出现一编辑组件;
- (6) 打开 Additional 组件选项卡, 单击 BitBtn 组件, 再在已建的窗体中单击, 就出现一位图按钮组件。

当然, 要符合自己的愿望, 还需要更改各组件的属性。



图 11.9 InputSecPerMoveDlg 对话框设计时的状态

1. 定义走步时间对话框总体属性

走步时间(秒)值对话框 InputSecPerMoveDlg(如图 11.9 所示)的功能是供用户输入机器每次走子的时间(秒)值, 主要属性如表 11.28 所示。

表 11.28 定义走步时间对话框属性

属 性	属性值
Left	189
Top	338
BorderIcons	[biSystemMenu]
BorderStyle	bsDialog
Caption	'设置每次移动秒数'
ClientHeight	174
ClientWidth	318
Color	ClBtnFace
ParentFont	True
OldCreateOrder	True
Position	PoScreenCenter
PixelsPerInch	96
TextHeight	13

2. 定义走步时间对话框各组件属性

走步时间(秒)值对话框 InputSecPerMoveDlg 还包括: 窗格组件 Bevel1、LABEL 组件 Label1, 窗格组件 Panel1, 编辑组件 Edit1, 位图按钮 BitBtn1、BitBtn2 等, 它们的属性如表 11.29~表 11.30 所示。

表 11.29 定义走步时间部分组件属性(1)

组 件	属 性	属性值
Bevel1	Left	0
	Top	96
	Width	309
	Height	1
Label1	Left	16
	Top	8
	Width	273
	Height	24
	AutoSize	False
	Caption	每次移动秒数:
	Color	ClGray
	Font.Charset	DEFAULT_CHARSET
	Font.Color	ClWindowText
	Font.Height	-19
	Font.Name	MS Sans Serif
	Font.Style	[]
Panel1	ParentColor	False
	ParentFont	False
	Left	16
	Top	48
	Width	273
	Height	41
	BevelOuter	BvLowered
	TabOrder	3

表 11.30 定义走步时间部分组件属性(2)

组 件	属 性	属性值
Edit1	Left	24
	Top	56
	Width	257
	Height	24
	AutoSize	False
	TabOrder	1
	Text	5.00
BitBtn1	Left	48
	Top	128
	Width	81
	Height	25
	Caption	确定
	Font.Charset	DEFAULT_CHARSET
	Font.Color	ClWindowText

续表

组 件	属 性	属性值
BitBtn1	ModalResult	1
	ParentFont	False
	TabOrder	0
	NumGlyphs	2
BitBtn2	Left	176
	Top	128
	Width	75
	Height	25
	Caption	忽略
	Font.Charset	DEFAULT_CHARSET
	Font.Color	ClWindowText
	ModalResult	2
	TabOrder	2

要说明的是, 按钮 BitBtn1 和 BitBtn2 的属性 ModalResult 的取值就是单击该按钮的返回值, 很重要。

11.2.9 定义走步时间对话框 InputSecPerMoveDlg 的方法

这里我们仍然采用显示一个静态对话框时, 调用其内置方法 ShowModal, 这样不需要用户再行定义其方法。方法 ShowModal 表示如果对话框不关闭, 则用户将不能操作其父窗口。要说明的是方法 Show 没有此限制, 但调用 Show 函数不会有 ModalResult 返回值, 而且还要为每个按钮定义事件回调函数。ShowModal 函数运行后将使定义走步时间对话框显示, 按钮的返回值 ModalResult 可有以下取值:

mrNone、mrOk、mrCancel、mrAbort、mrRetry、mrYes、mrNo、mrAll、mrYesToAll、mrNoToAll、mrIgnore。但在本程序中, 我们设定 InputSecPerMoveDlg 中的“确定”按钮 BitBtn1 的属性返回值为 mrOk 或 1、“忽略”按钮 BitBtn2 的属性返回值为 mrCancel 或 2。我们可以根据 ModalResult 的取值来完成不同的功能。

通过以下步骤可以察看 InputSecPerMove.cpp 中的代码:

- (1) 选择 File | Open Project 命令, 利用弹出的 Open Project 对话框打开 CHESS 项目;
- (2) 选择 View | Project Manager 命令, 在弹出的 Project Manager 对话框中双击 InputSecPerMove 打开 InputSecPerMove.cpp。

可以看到 InputSecPerMove.cpp 中没有用户添加的函数, 全由 IDE 自动管理生成。

在 CHESS 项目中主窗体【级别】|【每次移动时间...】菜单事件将调用 InputSecPerMoveDlg->ShowModal()方法, 代码如下:

```
void __fastcall TMainForm::MoveTimeClick(TObject *Sender)
{
    char *SecsPerMove = new char[40];
    COLORTYPE color;
```

```
int temp;
sprintf(SecsPerMove, "%.2f", AverageTime);

if (InputSecPerMoveDlg->ShowModal()==mrOk)
{
    SecsPerMove = InputSecPerMoveDlg->Edit1->Text.c_str();
    double NewMoveTime = atof(SecsPerMove);
    if (NewMoveTime > 0.)
    {
        if (Level != normal)
        {
            UnCheckLevelMenu(Level);
            CheckMenuItem(TChessMenu->Handle, MoveTime->Command, MF_CHECKED);
        }
        Level = normal;
        AverageTime = NewMoveTime;
        PrintCurLevel();

        for (temp=0;temp<=1;temp++)
        {
            { if (temp==0)
                color = red;
                if (temp==1)
                    color= black;
                ChessTime[color].totaltime = (MoveNumber / 2) * AverageTime;
            }
        }
        MaxLevel = MAXPLY;
    }
    else
        Error("时间不正确。请重输。");
}

delete SecsPerMove;

}
```

要引起注意的是，我们没有定义每个按钮的事件回调函数，而是采用按钮的 **ModalResult** 返回值，这样使编程更具灵活性、简明性。

第 12 章 CHESS 项目非可视化对话框

——动态对话框

有时在运行期间需要从头开始创建对话框，这时动态对话框技术可以派上用场。因为它能灵活地适应变化的情况，而且容易扩展它，使它包括不同的组件及不同组件的参数。

12.1 报告错误对话框 ChessErrorDlg 的主要属性

象棋游戏错误对话框 ChessErrorDlg 是根据窗体属性从头动态创建，其显示状态是不固定的，作用是当程序测试到错误信息时弹出的动态提示对话框，因为动态窗体不使用 DFM 文件，因此必须构造组件和分配事件句柄并在运行期间为 TChessErrorDlg 类设置参数特性，这些事情只能由构造函数负责产生。其函数定义放在 ChessErr.cpp 文件中，函数声明放在 ChessErr.h 文件中，这两个文件均由用户手工定义，IDE 不能自动产生。注意，没有 ChessErr.dfm 文件。

具体制作步骤如下：

- (1) 选择 File | Open Project 命令，利用弹出的 Open Project 对话框打开 CHESS 项目；
- (2) 选择 File | Open 命令，利用弹出的 Open 对话框打开 ChessErr.h 文件；
- (3) 手工定义 TChessErrorDlg 类，在其私有部分加入非可视化的组件及函数声明，在其公有部分声明构造函数，如下所示：

```
class TChessErrorDlg : public TForm
{
private:
    TButton *OKBtn;
    TBevel *Bevel1;
    TPanel *Panel1;
    TStaticText *Label2;
    void __fastcall OKBtnClick(TObject *Sender);
public:
    virtual __fastcall TChessErrorDlg(TComponent* AOwner, AnsiString
        ErrorString);
};
```

- (4) 选择 View | Project Manager 命令，在弹出的 Project Manager 对话框中双击 ChessErr.cpp 选项打开 ChessErr.cpp 文件；
- (5) 在打开的文件中手工添加如下构造函数：

```
__fastcall TChessErrorDlg::TChessErrorDlg(TComponent* AOwner,  
AnsiString ErrorString): TForm(AOwner, NULL)  
{  
    Left = 65;  
    Top = 539;  
    BorderStyle = bsDialog;  
    Caption = "中国象棋错误";  
    ClientHeight = 183;  
    ClientWidth = 292;  
    Color = clBtnFace;  
    ParentFont = True ;  
    OldCreateOrder = True;  
    Position = poScreenCenter;  
    PixelsPerInch = 96;  
  
    TBevel *Bevel1 =new TBevel(this);  
    Bevel1->Parent =this;  
    Bevel1->Height= 1;  
    Bevel1->Width = 292;  
    Bevel1->Left = 0 ;  
    Bevel1->Top = 120;  
  
    TPanel *Panell = new TPanel(this);  
    Panell->Parent =this;  
    Panell->Left = 48;  
    Panell->Top = 24;  
    Panell->Width = 185 ;  
    Panell->Height = 81;  
    Panell->BevelOuter = bvLowered;  
    Panell->Color = clGrayText;  
    Panell->TabOrder = 1 ;  
  
    TStaticText *Label2 = new TStaticText(this);  
    Label2->Parent =Panell;  
    Label2->Alignment=taCenter;  
    Label2->Align=alNone;
```

```
Label2->Color = Panel1->Color;
Label2->AutoSize= true;
Label2->Align=alClient;
Label2->Font->Charset = DEFAULT_CHARSET;
Label2->Font->Color = clBtnFace;
Label2->Font->Height = -14 ;
Label2->ParentColor = False;
Label2->ParentFont = False ;
Label2->Font->Color = clWindowText;
Label2->Font->Height = -17 ;
Label2->Left=20;
Label2->Top = Panel1->Top+Panel1->Height/2-Label2->Height/2-50;
Label2->Width= 108;
Label2->Height = 24;
Label2->Caption=ErrorString;

TButton *OKBtn = new TButton(this);
OKBtn->Parent =this;
OKBtn->Top=140;
OKBtn->Left=111;
OKBtn->Width= 75;
OKBtn->Height =25;
OKBtn->Caption = "OK";
OKBtn->OnClick = OKBtnClick;
OKBtn->Caption = "OK";
OKBtn->TabOrder = 1 ;

};
```

上述构造函数设置了 TChessErrorDlg 类的属性，包括以下几个方面：

- 为动态创建的组件设置 Top 和 Left 参数。否则，组件将出现在窗体的角上。
- 为其他起重要作用的组件设置参数。如果忘记设置原始参数，动态创建的组件将不会出现。
- 分配 OKBtnClick 给组件的事件句柄。单击 OKBtn 时 OKBtnClick 将执行。

下面的一组图显示象棋游戏程序测试到不同错误信息时弹出的错误对话框 ChessErrorDlg，如图 12.1 所示。

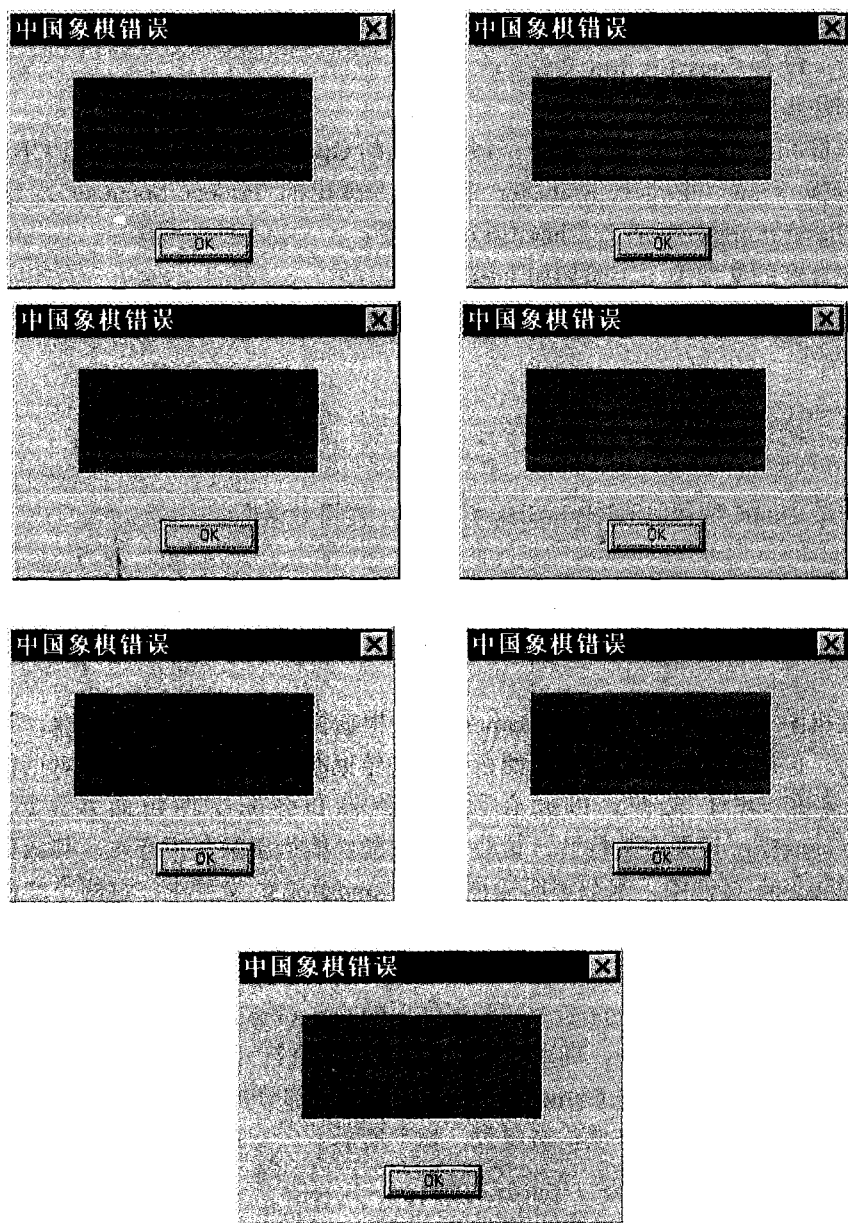


图 12.1 运行 Chess 项目出错时，弹出的对话框

12.2 报告错误对话框 ChessErrorDlg 的方法

报告错误对话框 ChessErrorDlg 的方法很简单，目的是提示用户程序的运行状态出了问题，提醒用户执行正确的操作。由于其上只放置了 OK 按钮以响应用户的请求，因此需要编写 OK 按钮回调函数，当用户单击，则关闭此对话框。其函数定义放在 ChessErr.cpp

文件中, 函数声明放在 ChessErr.h 文件中, 这两个文件均由用户手工定义, IDE 不能自动产生。注意, 没有 ChessErr.dfm 文件。

具体制作步骤如下:

- (1) 选择 File | Open Project 命令, 利用弹出的 Open Project 对话框打开 CHESS 项目;
- (2) 选择 View | Project Manager 命令, 在弹出的 Project Manager 对话框中双击 ChessErr.cpp 选项打开 ChessErr.cpp 文件;
- (3) 在打开的文件中手工添加如下函数:

```
void __fastcall TChessErrorDlg::OKBtnClick(TObject *Sender)
{
    Close();
}
```

当用户单击 OK 按钮时 OKBtnClick 将执行, 关闭对话框。

12.3 结束对弈对话框 EndChessGameDlg 的主要属性

结束象棋游戏对话框 EndChessGameDlg 是根据窗体属性从头动态创建, 其显示状态是不固定的, 作用是询问用户重新开始游戏还是结束游戏弹出的动态提示对话框, 因为动态窗体不使用 DFM 文件, 因此必须构造组件和分配事件句柄并在运行期间为 TEndChessGameDlg 设置参数特性, 这些事情只能由构造函数负责产生。其函数定义放在 EndGameDlog.cpp 文件中, 函数声明放在 EndGameDlog.h 文件中, 这两个文件均由用户手工定义, IDE 不能自动产生。注意, 没有 EndGameDlog.dfm 文件。

具体制作步骤如下:

- (1) 选择 File | Open Project 命令, 利用弹出的 Open Project 对话框打开 CHESS 项目;
- (2) 选择 File | Open 命令, 利用弹出的 Open 对话框打开 EndGameDlog.h 文件;
- (3) 手工定义 TEndChessGameDlg 类, 在其私有部分加入非可视化的组件及函数声明, 在其公有部分声明构造函数, 如下所示:

```
class TEndChessGameDlg : public TForm
{
private:
    TButton *OKBtn;
    TButton *CancelBtn;
    TBevel *Bevel1;
    TPanel *Panel1;
    TLabel *Label1;

    void __fastcall YesBtnClick(TObject *Sender);
    void __fastcall NoBtnClick(TObject *Sender);
public:
```



```
virtual __fastcall TEndChessGameDlg(TComponent* AOwner, AnsiString  
EndGameString);  
};
```

- (4) 选择 View | Project Manager 命令, 在弹出的 Project Manager 对话框中双击 EndGameDlog.cpp 选项打开 EndGameDlog.cpp 文件;
- (5) 在打开的文件中手工添加如下构造函数:

```
__fastcall TEndChessGameDlg::TEndChessGameDlg(TComponent* AOwner,  
AnsiString EndGameString): TForm(AOwner, NULL)  
{  
    Left = 65;  
    Top = 539;  
    BorderStyle = bsDialog;  
    Caption = "中国象棋";  
    ClientHeight = 183;  
    ClientWidth = 292;  
    Color = clBtnFace;  
    ParentFont = True ;  
    OldCreateOrder = True;  
    Position = poScreenCenter;  
    PixelsPerInch = 96;  
  
    TBevel *Bevell = new TBevel(this);  
    Bevell->Parent =this;  
    Bevell->Height= 1;  
    Bevell->Width = 292;  
    Bevell->Left = 0 ;  
    Bevell->Top = 120;  
  
    TPanel *Panell = new TPanel(this);  
    Panell->Parent =this;  
    Panell->Left = 48;  
    Panell->Top = 24;  
    Panell->Width = 185 ;  
    Panell->Height = 81;  
    Panell->BevelOuter = bvLowered;  
    Panell->Color = clGrayText;  
    Panell->TabOrder = 1 ;
```

```
TLabel *Label1 = new TLabel(this);
Label1->Parent =Panel1; //保证 Panel1 在底层,Label1 在顶层
Label1->AutoSize= true;
Label1->Alignment=taCenter;
Label1->Color = clGrayText;
Label1->Font->Charset = DEFAULT_CHARSET;
Label1->Font->Color = clWindowText;
Label1->Font->Height = -17 ;
Label1->ParentColor = False;
Label1->ParentFont = False ;
Label1->Height = 24;
Label1->Width= 50;
Label1->Left = Panel1->Left+Panel1->Width/2-Label1->Width/2-50;
Label1->Top = Panel1->Top+Panel1->Height/2-Label1->Height/2-40;
Label1->Caption=EndGameString;

TLabel *Label2 = new TLabel(this);
Label2->Parent =Panel1; //保证 Panel1 在底层,Label2 在顶层
Label2->AutoSize= true;
Label2->Alignment=taCenter;
Label2->Color = clGrayText;
Label2->Font->Charset = DEFAULT_CHARSET;
Label2->Font->Color = clWindowText ;
Label2->Font->Height = -17 ;
Label2->ParentColor = False;
Label2->ParentFont = False ;
Label2->Width= 60;
Label2->Height = 24;
Label2->Left = Panel1->Left+Panel1->Width/2-Label2->Width/2-50;
Label2->Top = Panel1->Top+Panel1->Height/2-Label2->Height/2;
Label2->Caption="还玩吗? ";

TButton *YesBtn = new TButton(this);
YesBtn->Parent =this;
YesBtn->Top=140;
YesBtn->Left=60;
YesBtn->Width= 75;
YesBtn->Height =25;
YesBtn->Caption = "YES";
```

```
YesBtn->OnClick = YesBtnClick;  
YesBtn->Caption = "Yes";  
YesBtn->TabOrder = 0 ;  
  
TButton *NoBtn = new TButton(this);  
NoBtn->Parent =this;  
NoBtn->Top=140;  
NoBtn->Left=170;  
NoBtn->Width= 75;  
NoBtn->Height =25;  
NoBtn->Caption = "NO";  
NoBtn->OnClick = NoBtnClick;  
NoBtn->Caption = "No";  
NoBtn->TabOrder = 1 ;  
  
};
```

上述构造函数设置了 TEndChessGameDlg 类的属性，包括以下几个方面：

- 为动态创建的组件设置 Top 和 Left 参数。否则，组件将出现在窗体的角上。
- 为其他起重要作用的组件设置参数。如果忘记设置原始参数，动态创建的组件将不会出现。
- 分配 YesBtnClick 给组件的事件句柄。当用户单击 Yes 按钮时 YesBtnClick 将执行。
- 分配 NoBtnClick 给组件的事件句柄。当用户单击 No 按钮时 NoBtnClick 将执行。

如图 12.2，图 12.3 所示的是象棋游戏程序测试到不同状态时弹出的询问对话框 EndChessGameDlg。



图 12.2 EndChessGameDlg 运行时的状态

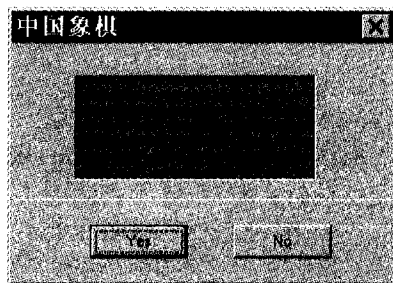


图 12.3 EndChessGameDlg 运行时的状态

12.4 结束对弈对话框 EndChessGameDlg 的主要方法

结束对弈对话框 EndChessGameDlg 的方法很简单，目的是对局结束时提示用户是否愿意继续下一盘的对弈，供用户执行选择操作。其上放置了“**Yes**”、“**No**”两个按钮以响应用户的请求。当用户单击 **Yes** 按钮，程序将发送一个消息调用主窗体类的 FileNew 函数以新建棋盘回到初状态，同时关闭此对话框；当用户单击 **No** 按钮，程序将发送一个消息调用主窗体的 Exit 函数以结束对弈状态，同时关闭此对话框。其函数定义放在 EndGameDlog.cpp 文件中，函数声明放在 EndGameDlog.h 文件中，这两个文件均由用户手工定义，IDE 不能自动产生。注意，没有 EndGameDlog.dfm 文件。

具体制作步骤如下：

- (1) 选择 **File | Open Project** 命令，利用弹出的 **Open Project** 对话框打开 **CHESS** 项目；
- (2) 选择 **View | Project Manager** 命令，在弹出的 **Project Manager** 对话框中双击 **EndGameDlog.cpp** 选项打开 EndGameDlog.cpp 文件；
- (3) 在打开的文件中手工添加如下函数：

```
void __fastcall TEndChessGameDlg::NoBtnClick(TObject *Sender)
{
    PostMessage(MainForm->Handle, IDM_EXIT, 0, 0);
    Close();
}
```

当用户单击 **No** 按钮时 NoBtnClick 将执行，关闭应用程序。

//-----

```
void __fastcall TEndChessGameDlg::YesBtnClick(TObject *Sender)
{
    GameOver = false;
    PostMessage(Application->MainForm->Handle, IDM_FILENEW, 0, 0);
    Close();
}
```

当用户单击 **Yes** 按钮时 YesBtnClick 将执行，建立棋盘回到初始状态。

第 13 章 CHESS 项目的 API 函数 及自定义函数

从事 Windows 应用程序开发，离不开对 Win32 API 函数的调用。同时能否创建新函数而非仅操作已有的函数也是衡量程序员水平高低的重要方面。除了熟悉 Windows API 函数外，学习、借鉴别人的自定义函数也是提高自己的捷径。

13.1 CHESS 项目中 Windows API 函数详解

可视化编程环境操作简单高效、界面友好。采用类库和组件编程，程序员可以把主要精力放在程序整体功能的设计上，从而不必关心具体过程细节的实现。但是，类库和组件只提供了一些带普遍性的功能，对于一些比较复杂和特殊的功能来说，单使用类库和组件是难以实现的。因此要想开发出更灵活、更实用、更具效率的应用程序，必然要涉及到直接使用 Windows API 函数。API 函数是构筑整个 Windows 程序框架的基石，只有充分理解和利用 API 函数，才能深入到 Windows 内部，充分发挥 C++Builder 平台的强大功能和灵活性，才能成功地扩展和突破类库、组件等可视化环境的限制。API 函数一般分为五大类函数：窗口管理接口函数、图形设备接口函数、系统管理接口函数、国际特性接口函数及网络服务接口函数，这里只对 CHESS 项目中用到的一些函数作简要介绍，完整的描述由于涉及技术内容广泛，篇幅庞大，在此不赘述。

Windows API 函数是最底层的函数，不需要用户定义，只需在调用它的文件中插入包含文件(如#include <stdio.h>)，编译程序会自动找到相应的库文件进行连接。CHESS 项目中用到不少 API 函数，目的是扩充 C++Builder 可视化函数的功能，完成可视化函数不能完成的功能。如：载入自定义光标、重画菜单条中的某一项、激活/禁止鼠标输入、操作文本或位图资源等。以下的 API 函数分散于 CHESS 项目不同的文件中，往往由高层的可视化函数来调用。

1. LoadCursor

功能：从一个与应用事例相关的执行文件(.EXE 文件)中载入指定的光标资源。

定位：文件 MainForm1.cpp 中函数 void __fastcall TMainForm::FormShow(TObject *Sender);

函数原型：HCURSOR LoadCursor(HINSTANCE hInstance,LPCTSTR lpCursorName);

返回值：如果成功，返回值是新载入的光标的句柄；如果失败，返回值是 NULL。

注意： LoadCursor 函数仅载入没有载入过的光标资源，否则，它检取已存在的光标资源的句柄。仅当 lpCursorName 参数指向一个光标资源时，该函数才返回一个有效的光标句柄。如果 lpCursorName 不是指向光标而是指向了其他类型的资源(如 icon)，则该函数返回值不是 NULL，尽管它不是一个有效的光标句柄，该函数为当前显示设备光标搜索最贴切的光标资源。光标资源可以是彩色或单色的位图。

示例： 装入自定义光标 IDC_WCUR 到实例 HInstance 中。

```
Screen->Cursors[crWaitCursor] = LoadCursor(HInstance, "IDC_WCUR");
```

说明： HInstance: 标识一个应用程序的全局实例句柄。

IDC_WCUR: 自定义光标资源标识符。

2. MessageBox

功能： 创建、显示、和操作一个消息框。

定位： 文件 GlobalFun.cpp 中函数 static bool SaveChessBoard(char *savefile)。

函数原型： int MessageBox(HWND hWnd,LPCTSTR lpText,LPCTSTR lpCaption,UINT uType);

返回值： 如果没有足够的内存来创建消息框，则返回值为零。如果调用成功，则返回值为下列对话框返回的菜单项目值中的一个：

IDABORT: Abort 按钮被选中。

IDCANCE: Cancel 按钮被选中。

IDIGNORE: Ignore 按钮被选中。

IDNO: No 按钮被选中。

IDOK: Ok 按钮被选中。

IDRETRY: Retry 按钮被选中。

IDYES: Yes 按钮被选中。

如果一个消息框有一个 Cancel 按钮，且如果 Esc 键被按下或 Cancel 键被选择，则函数返回 IDCANCEL 值。如果消息框没有 Cancel 按钮，则按 Esc 键没有作用。

示例：

```
MessageBox(Application->Handle, "没有足够的内存执行操作",  
            "中国象棋", MB_OK | MB_ICONHAND);
```

说明： Application->Handle: 标识应用程序实例句柄。

“没有足够的内存执行操作”：将被显示的消息的字符串。

“中国象棋”：对话框标题的字符串。

MB_OK: 消息框含有一个 OK 按钮图标。

MB_ICONHAND: 停止消息框图标。

3. DrawMenuBar

功能：重画指定菜单的菜单条。

定位：文件 MainForm1.cpp 中函数 `void __fastcall TMainForm::TwoPlayerClick(TObject *Sender);`

函数原型：`bool DrawMenuBar(HWND hWnd)`

返回值：如果调用成功，返回非零值；如果调用失败，返回值是零。

示例：

```
DrawMenuBar(MainForm->Handle);
```

说明： MainForm->Handle: 主窗体实例句柄。

4. ReleaseCapture

功能：从当前线程中的窗口释放鼠标捕获，并恢复通常的鼠标输入处理。

定位：文件 GolbalFun.cpp 中函数 `void DragEnd(bool ValidMove)`。

函数原型：`bool ReleaseCapture(void)`

返回值：如果调用成功，返回非零值；如果调用失败，返回值是零。

注意：应用程序在调用函数 SetCapture 之后调用此函数。

示例：

```
ReleaseCapture();
```

说明： 无参数。

5. SetCapture

功能：在属于当前线程的指定窗口里设置鼠标捕获。

定位：文件 GlobalFun.cpp 中函数 `void DragStart(SQUARETYPE square, POINT& point)`。

函数原型：`HWND SetCapture(HWND hWnd);`

返回值：返回值是上次捕获鼠标的窗口句柄。如果不存在那样的句柄，返回值是 NULL。

注意： 只有前台窗口才能捕获鼠标。如果一个后台窗口想捕获鼠标，则该窗口仅为其光标热点在该窗口可见部分的鼠标事件接收消息。另外，即使前台窗口已捕获了鼠标，用户也可点击该窗口，将其调入前台。当一个窗口不再需要所有的鼠标输入时，创建该窗口的线程应当调用函数 ReleaseCapture 来释放鼠标。此函数不能被用来捕获另一进程的鼠标输入。

示例：

```
SetCapture(MainForm->Handle); //捕获光标，禁止输入
```

说明： MainForm->Handle: 主窗体实例句柄。

6. BitBlt

功能：对指定的源设备环境区域中的像进行位块(bit-block)转换，以传送到目标设备环境。

定位：文件 GlobalFun.cpp 中函数 static void ShowPiece(POINT& p)。

函数原型：bool BitBlt(HDC hdcDest,int nXDest,int nYDest,int nWidth,int nHeight,HDC hdcSrc,int nXSrc,int nYSrc,DWORD dwRop);

返回值：如果成功，那么返回值非零；如果失败，那么返回值为零。

示例：

```
BitBlt(hdc, p.x, p.y , Bitmap.bmWidth, Bitmap.bmHeight,hMemoryDC, 0, 0, SRCAND);
```

说明：hDC: 指向目标设备环境的句柄。

p.x: 指向目标矩形区域左上角的 X 轴逻辑坐标。

p.y: 指向目标矩形区域左上角的 Y 轴逻辑坐标。

Bitmap.bmWidth: 指定目标矩形区域的逻辑宽度。

Bitmap.bmHeight: 指定目标矩形区域的逻辑高度。

hMemoryDC: 指向源设备环境的句柄。

0: 指向源矩形区域左上角的 X 轴逻辑坐标。

0: 指向源矩形区域左上角的 Y 轴逻辑坐标。

SRCAND: 通过使用布尔型的 AND(与)操作符将源矩形区域的颜色与目标矩形区域的颜色合并。

7. CreateCompatibleBitmap

功能：创建与指定的设备环境相关的设备兼容的位图。

定位：文件 GlobalFun.cpp 中函数 static void ShowPiece(POINT& p)。

函数原型：HBITMAP CreateCompatibleBitmap(HDC hdc,int nWidth,int nHeight);

返回值：如果执行成功，那么返回值是位图的句柄；如果执行失败，那么返回值为 NULL。

示例：

```
CreateCompatibleBitmap(hdc, Bitmap.bmWidth, Bitmap.bmHeight);
```

说明：hDC: 设备环境句柄。

Bitmap.bmWidth: 指定位图的宽度，单位为像素。

Bitmap.bmHeight: 指定位图的高度，单位为像素。

8. LoadBitmap

功能：从模块的可执行文件中加载指定的位图资源。

定位：文件 MainForm1.cpp 中函数 void __fastcall TMainForm::FormShow(TObject *Sender)。

函数原型：HBITMAP LoadBitmap(HINSTANCE hInstance, LPCTSTR lpBitmapName);

返回值：如果执行成功，则返回值指向指定位图的句柄；如果执行失败，那么返回值是 NULL。

示例：

```
::PieceBmpArray[pawn-1][black] = LoadBitmap(HInstance, "BPawnBmp");
```

说明： HInstance: 指向应用程序实例的全局句柄。

"BPawnBmp": 要加载的黑兵位图资源名称。

9. CreateSolidBrush

功能：创建一个具有指定颜色的逻辑刷子。

定位：文件 MainForm1.cpp 中函数 void __fastcall TMainForm::FormShow(TObject *Sender)。

函数原型：HBRUSH CreateSolidBrush(COLORREF crColor);

返回值：如果执行成功，则返回值标识一个逻辑实心刷子；如果执行失败，那么返回值是 NULL。

示例： hColorBrush = CreateSolidBrush(

```
RGB(ColorSquareColors[0], ColorSquareColors[1], ColorSquareColors  
[2]));
```

说明： RGB(ColorSquareColors[0], ColorSquareColors[1], ColorSquareColors[2]): 根据红绿蓝颜色确定刷子的颜色。

10. PatBlt

功能：使用当前选入指定设备环境中的刷子绘制给定的矩形区域。

定位：文件 GlobalFun.cpp 中函数 void ClearSquare(SQUARETYPE square)。

函数原型： bool PatBlt(HDC hdc, int nXLeft, int nYLeft, int nWidth, int nHeight, DWORD dwRop);

返回值：如果执行成功，则返回值非零；如果执行失败，那么返回值是 0。

示例： PatBlt(hdc, p.x, p.y, SQUARE_SIZE, SQUARE_SIZE, PATCOPY);

说明： hdc: 设备环境句柄。

p.x: 指向要填充矩形左上角的 X 轴逻辑坐标。

p.y: 指向要填充矩形左上角的 Y 轴逻辑坐标。

SQUARE_SIZE: 指定矩形宽度，按逻辑单位表示。

SQUARE_SIZE: 指定矩形高度，按逻辑单位表示。

PATCOPY: 将指定的模式拷贝到目标位图上。

11. ScreenToClient

功能：把屏幕上指定点的屏幕坐标转换成用户坐标。

定位：文件 MainForm1.cpp 中函数 void __fastcall TMainForm::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift, int X, int Y)。

函数原型：bool ScreenToClient(HWND hWnd, LPPOINT lpPoint);

返回值：如果执行成功，则返回值非零；如果执行失败，那么返回值是 0。

示例：MainForm->ScreenToClient(point); //C++Builder 中的用法

说明：MainForm: 主窗口的句柄，此窗口的用户空间将被用来转换。

point: 含有要转换的屏幕坐标。

12. CreateCompatibleDC

功能：创建一个与指定设备兼容的内存设备上下文环境(DC)。

定位：文件 GlobalFun.cpp 中函数 void PrintPiece(SQUARETYPE square, PIECETYPE piece, COLORTYPE color, DWORD Rop)。

函数原型：HDC CreateCompatibleDC(HDC hdc);

返回值：如果执行成功，则返回内存上下文环境的句柄；如果失败，那么返回值是 NULL。

示例：hMemoryDC = CreateCompatibleDC(hdc);

说明：hDC: 现有设备上下文环境的句柄。

13. DeleteObject

功能：删除一个逻辑笔、画笔、字体、位图、区域或者色板，释放所有与该对象有关的系统资源，在对象被删除之后，指定的句柄也就失效了。

定位：文件 GlobalFun.cpp 中函数 void DrawFrame(HDC hdc, RECT& ChessBoardRect, bool DrawBackground)。

函数原型：bool DeleteObject(HGDIOBJ hObject);

返回值：如果执行成功，则返回非零值；如果指定的句柄无效或者它已被选入设备上下文环境，则返回值是零。

示例：DeleteObject(hColorBrush);

说明：hColorBrush: 颜色画笔的句柄。

14. GetDC

功能：检索一指定窗口的客户区域或整个屏幕的显示设备上下文环境的句柄，以后可以在 GDI 函数中使用该句柄来设备上下文环境中绘图。

定位：文件 MainForm1.cpp 中函数 void __fastcall TMainForm::FormShow(TObject *Sender)。

函数原型：HDC GetDC(HWND hWnd);

返回值: 如果执行成功, 则返回指定窗口客户区的设备上下文环境; 如果失败, 返回值为 NULL。

示例: `hDC = GetDC(MainForm->Handle);`

说明: `MainForm->Handle`: 设备上下文环境被检索的主窗口句柄。

15. ReleaseDC

功能: 释放设备上下文环境(DC)供其他应用程序使用。

定位: 文件 `MainForm1.cpp` 中函数 `void __fastcall TMainForm::FormShow(TObject *Sender)`。

函数原型: `int ReleaseDC(HWND hWnd, HDC hDC);`

返回值: 返回值说明了设备上下文环境是否释放; 如果释放成功, 则返回值为 1; 如果没有释放成功, 则返回值为 0。

示例: `ReleaseDC(MainForm->Handle, hDC);`

说明: `MainForm->Handle`: 指向要释放的设备上下文环境所在的主窗口句柄。

`hDC`: 指向要释放的设备上下文环境句柄。

16. SelectObject

功能: 选择一对象到指定的设备上下文环境中, 该新对象替换先前的同类型的对象。

定位: 文件 `GlobalFun.cpp` 中函数 `void DrawFrame(HDC hDC, RECT& ChessBoardRect, bool DrawBackground)`。

函数原型: `HGDIOBJ SelectObject(HDC hdc, HGDIOBJ hgdiobj);`

返回值: 如果选择对象不是区域并且函数执行成功, 那么返回值是被取代的对象的句柄;

如果选择对象是区域并且函数执行成功, 返回如下一值:

SIMPLEREGION: 区域由单个矩形组成;

COMPLEXREGIN: 区域由多个矩形组成;

NULLREGIN: 区域为空。

如果发生错误并且选择对象不是一个区域, 那么返回值为 NULL, 否则返回 `GDI_ERROR`。

示例: `hOldBrush = (HBRUSH)SelectObject(hDC, GetStockObject(NULL_BRUSH));`

说明: `hDC`: 设备上下文环境的句柄。

`GetStockObject(NULL_BRUSH)`: 被选择的对象句柄, 该指定对象由

`GetStockObject` 函数创建。

17. Rectangle

功能: 画一个矩形, 用当前的画笔画矩形轮廓, 用当前画刷进行填充。

定位: 文件 `GlobalFun.cpp` 中函数 `void DrawFrame(HDC hDC, RECT& ChessBoardRect, bool DrawBackground)`。

函数原型：bool Rectangle(HDC hdc,int nLeftRect,int nTopRect,int nRightRect,int nBottomRect);

返回值：如果执行成功，则返回值非零；如果函数执行失败，那么返回值是 0。

示例：Rectangle(hdc, x1, y1, x2, y2);

说明：hdc: 设备环境句柄。

x1: 指定矩形左上角的 X 轴逻辑坐标。

y1: 指定矩形左上角的 Y 轴逻辑坐标。

x2: 指定矩形右下角的 X 轴逻辑坐标。

y2: 指定矩形右下角的 Y 轴逻辑坐标。

18. GetTextMetrics

功能：给指定的缓冲区里填入当前选择字体的度量值。

定位：文件 MainForm1.cpp 中函数 void __fastcall TMainForm::FormShow(TObject *Sender)。

函数原型：bool GetTextMetrics(HDC hdc,LPTEXTMETRIC lptm);

返回值：如果执行成功，则返回值非空；如果函数执行失败，那么返回值是 0。

示例：TEXTMETRIC tm;

```
GetTextMetrics(hdc, &tm);
```

说明：hdc: 设备环境句柄。

tm: 接收度量值的结构 TEXTMETRIC 的指针。

19. TextOut

功能：用当前选择字符、背景颜色和正文颜色将一个字符串写到指定位置。

定位：文件 GlobalFun.cpp 中函数 static void DrawAlphaNum()。

函数原型：bool TextOut(HDC hdc,int nXStart,int nYStart,LPCTSTR lpString,int cbString);

返回值：如果调用成功，返回非零值；如果调用失败，返回值是 0。

示例：TextOut(hdc, XPos, YPos, NumArray[8-i], 2);

说明：hdc: 设备环境句柄。

XPos: 指定用于字符串对齐的基准点的逻辑 X 坐标。

YPos: 指定用于字符串对齐的基准点的逻辑 Y 坐标。

NumArray[8-i]: 数字字符的指针。

2: 一个中文字符的字节数。

20. LineDDA

功能：确定由起点和终点定义的线段上哪些点将高亮度显示。

定位：文件 GlobalFun.cpp 中函数 void DragEnd(bool ValidMove)。

函数原型: bool LineDDA(int nXStart,int nYStart,int nXEnd,int nYEnd,LINEDDAPROC lpLineFunc,LPARAM lpData);

返回值: 如果调用成功, 则返回值不为 0; 如果调用失败, 那么返回值是 0。

示例: LineDDA(CurPoint.x, CurPoint.y, StartPoint.x, StartPoint.y,

 LINEDDAPROC(ReturnHome), NULL);

说明: CurPoint.x: 定义线段起点的 X 坐标。

CurPoint.y: 定义线段起点的 Y 坐标。

StartPoint.x: 定义线段终点的 X 坐标。

StartPoint.y: 定义线段终点的 Y 坐标。

LINEDDAPROC(ReturnHome): 应用程序定义的回调函数指针。

NULL: 应用程序定义的数据指针为空。

21. LineDDAProc

功能: 是与 LineDDA 函数一起使用的应用程序定义的回调函数。

定位: 文件 GlobalFun.cpp 中函数 void DragEnd(bool ValidMove)。

函数原型: VOID CALLBACK LineDDAProc(int X,int Y,LPARAM lpData);

返回值: 无。

示例: LineDDAProc(Point.x, Point.x, lpData);

说明: Point.x: 当前点的 X 坐标。

Point.x: 当前点的 Y 坐标。

lpData: 应用程序定义的数据指针。

22. ineTo

功能: 从当前位置画线到指定的点, 但不包括给定点。

定位: 文件 GlobalFun.cpp 中函数 void ClearSquare(SQUARETYPE square)。

函数原型: bool LineTo(HDC hdc,int nXEnd,int nYEnd);

返回值: 如果调用成功, 则返回值不为 0; 如果调用失败, 那么返回值为 0。

示例: LineTo(hdc, p.x, p.y);

说明: hdc: 设备环境句柄。

p.x: 定义线段终点的 X 坐标。

p.y: 定义线段终点的 Y 坐标。

23. MoveToEx

功能: 将当前位置更新为指定的点, 并有选择的返回原先的位置。

定位: 文件 GlobalFun.cpp 中函数 void ClearSquare(SQUARETYPE square)。

函数原型: bool MoveToEx(HDC hdc,int X,int Y,LPPPOINT lpPoint);

返回值: 如果调用成功, 则返回值不为 0; 如果调用失败, 那么返回值为 0。

示例: MoveToEx(hdc, p.x, p.y, NULL);

说明: hDC: 设备环境句柄。
p.x: 定义新位置的 X 逻辑坐标。
p.y: 定义新位置的 Y 逻辑坐标。
NULL: 不返回原先的位置。

24. SetBkColor

功能: 将当前背景颜色设置为指定的颜色值, 如果设备不能实现指定的颜色值则设置为最接近的物理颜色。

定位: 文件 GlobalFun.cpp 中函数 static void DrawAlphaNum()。

函数原型: COLORREF SetBkColor(HDC hDC, COLORREF crColor);

返回值: 如果调用成功, 返回值指定原来的背景颜色值为 COLORREF 值; 如果调用失败, 返回值为 CLR_INVALID。

示例: SetBkColor(hDC, RGB(192, 192, 192));

说明: hDC: 设备环境句柄。
RGB(192, 192, 192): 指定浅灰色为新的背景颜色。

25. CreatePen

功能: 用于创建一个具有指定的风格宽度和颜色的逻辑画笔。

定位: 文件 GlobalFun.cpp 中函数 void DrawFrame(HDC hDC, RECT& ChessBoardRect, bool DrawBackground)。

函数原型: HPEN CreatePen(int fnPenStyle, int nWidth, COLORREF crColor);

返回值: 如果调用成功, 则返回值为标识一个逻辑笔的句柄; 如果调用失败, 则返回值为 NULL。

示例: hPen = CreatePen(PS_SOLID, 1, RGB(128, 128, 128));

说明: PS_SOLID: 实线笔。
1: 指定笔的宽度为一个像素单位。
RGB(128, 128, 128): 指定笔的颜色为深灰色。

26. PtInRect

功能: 用于确定指定点是否在指定矩形内。

定位: 文件 GlobalFun.cpp 中函数 SQUARETYPE GetValidSquare(POINT p, COLORTYPE player, bool CheckPiece)。

函数原型: bool PtInRect(CONST RECT *lprc, POINT pt);

返回值: 如果指定点在矩形内, 则返回值不为零; 如果指定点不在矩形内, 则返回值为 0。

示例: PtInRect(&Rect, point);

说明: Rect: 被检查矩形区域的指针。
pt: POINT 结构的指定的点。

27. lstrcat

功能：将一个字符串附加在另一个字符串后面。

定位：文件中 GlobalFun.cpp 函数 char *MoveStr(MOVETYPE *move)。

函数原型：LPTSTR lstrcat(LPTSTR lpString1, LPCTSTR lpString2);

返回值：若运行成功，返回值指向缓冲区；若失败，则返回值为 NULL。

示例：lstrcat(str, str1);

说明： str: 一个以 NULL 为终止符字符串的指针。这个缓冲区必须足够大能包含两个字符串。

str1: 一个以 NULL 为终止符字符串的指针，它将追加在由 str 中指定字符串后面的字符串指针。

28. lstrcpy

功能：复制一个字符串到缓冲区。

定位：文件中 GlobalFun.cpp 函数 char *MoveStr(MOVETYPE *move)。

函数原型：lstrcpy(LPTSTR lpString1, LPCTSTR lpString2);

返回值：若运行成功，返回值是缓冲区的指针；若失败，则返回值为 NULL。

示例：lstrcpy(str, str0);

说明： str: 指向接收由参数 str0 指向字符串内容的缓冲区。这个缓冲区必须足够大能包含字符串，还包括最后的 NULL 终止符。

str0: 指向待复制的以 NULL 为终止符的字符串。

13.2 自定义函数的声明及调用

CHESS 项目自定义函数的声明单独作为一个包含文件放在 GLOBAL.h 中，这样可以方便调用，由于冠以 extern 说明为全局变量，它们不单独属于某一个特定的类，所有的类函数均可以调用。如果说 API 函数处于金字塔的最底层，则自定义函数处于第二层，类的成员函数处于第三层。GLOBAL.h 文件只是声明了自定义函数名称，每个名称的具体作用即自定义函数的定义可以分布于不同的模块单元中。CHESS 项目中的自定义函数放在 GLOBAFUN.cpp 文件中。调用时就像 Windows API 函数一样，在需要的类文件中(如 MainForm1.cpp)插入 GLOBAL.h 包含文件就可进行调用，形如：#include "Global.h"。

以下列出 CHESS 项目中的自定义函数。

```
extern void DragStart(SQUARETYPE, POINT&);
extern void DragEnd(bool);
extern void Drag(POINT&);
extern short Attacks(COLORTYPE acolor, SQUARETYPE square);
extern POINT GetSquareXY(SQUARETYPE);
extern void ClearSquare(SQUARETYPE square);
```

```
extern void DrawFrame(HDC, RECT&, bool=false);
extern void ClearChessBoard();
extern void ResetNewPos();
extern void Error(char *);
extern void ColorToPlay(COLORTYPE);
extern void InitDisplay();
extern void ClearDisplay();
extern void ClearInfoWindow();
extern void NewChessGame();

extern void InsertPiece(PIECETYPE p, COLORTYPE c, SQUARETYPE sq);
extern void PrintCurLevel();
extern void InitNode(NODEVAL*);
extern void InitTime(CHESSCLOCKTYPE *clock);
extern void StartTime(CHESSCLOCKTYPE *);
extern void StopTime(CHESSCLOCKTYPE *);
extern void PrintNodes(NODEVAL *, double);
extern void Perform(MOVETYPE *, bool);
extern void TakeBackMove(MOVETYPE *move);
extern void MakeMove(MOVETYPE *);
extern bool EqMove(MOVETYPE *a, MOVETYPE *b);
extern void AddNode(NODEVAL *nodes);

extern short PieceAttacks(PIECETYPE, COLORTYPE, SQUARETYPE, SQUARETYPE);
extern SIXTYTYPE SixtyMoveCnt(void);
extern REPEATTYPE Repetition(short immediate);
extern void PrintBestMove(MOVETYPE *mainpath, MAXTYPE mainvalu);
extern void PrintMove(bool *singlestep, DEPTHTYPE maxdepth, MAXTYPE
evaluation, bool);
extern int StatEvalu(MOVETYPE *move);
extern short KillMovGen(MOVETYPE *move);
extern void CalcPVTable();
extern void EnterKeyMove();
extern void StartAnalysis();
extern void Warning(char *);

extern void PrintMove(int, COLORTYPE, MOVETYPE *, double);
extern void UpdateChessBoard();
extern void InitMovGen();
```



```
extern void MovGen();
extern void Message(char *);
extern void CalcPieceTable();
extern void RemoveBestPath();
extern void ResetMoves();
extern void InitChessTime();
extern void RemoveHint();
extern void ClearPVTable();
extern void AdjustMoves();
extern void SearchMove(int maxlevel);
extern void OpeningLibMsg();
extern char *MoveStr(MOVETYPE *);
extern void RemoveMessage();
extern void EnterMove(MOVETYPE *move);

extern void FlashMove(MOVETYPE *move);
extern void StoreMoves();
extern void CalcAttackTable();
extern void SaveGame(char *savefile);
extern void RestoreGame(char *);
extern bool Undo();
extern bool Redo();
extern void PrintChessBoard();
extern void SearchHintMove();
extern void DisplayHint();
extern void Talk();
extern SQUARETYPE GetValidSquare(POINT, COLORTYPE, bool);
extern void DrawInvertedBitmap(SQUARETYPE);
extern void DrawNormalBitmap(SQUARETYPE);
extern void InitCommStuff();
extern bool MoveCheck(SQUARETYPE, SQUARETYPE);
extern void ProgramMove();
extern WORD MessageToPost;
extern void QuitProgram();
extern void HideAttacks();
extern void DisplayTime();
extern void InitNode(NODEVAL *nodes);
extern void RemoveKillMove(void);
```

13.3 CHESS 程序中部分自定义函数的使用说明

CHESS 项目中的自定义函数放在 16 个函数定义源程序之一的 GlobalFun.cpp 文件中,具体说明每一个自定义函数的作用,它们只能由更低层的 API 库函数来合成,不属于某一特定的类,但类函数可以调用,在需要的类文件中(如 EditForm2.cpp)手工插入 GLOBAL.h 包含文件就可进行调用,形如:

```
#include "Global.h".
```

而与 CHESS 项目文件相关的 CHESS.cpp 中的文件调用全部由 IDE 自动管理,例如:

```
#include <vcl.h>
#pragma hdrstop
```

USEUNIT("GlobalFun.cpp");//由 IDE 自动管理

编译程序会根据此依赖关系进行连接。

以下列出 CHESS 项目中的自定义函数。

1. 函数名称: void ClearChessBoard()

说明: 清除棋盘和初始化棋盘。

函数定义如下:

```
void ClearChessBoard()
{
    SQUARETYPE Square;
    for (Square = 0; Square <= 0x98; Square++)
    {
        ChessBoard[Square].piece = empty;
        ChessBoard[Square].color = red;
    }
}
```

2. 函数名称: void ClearIndex(void)

说明: 清除棋盘索引和棋子表。

函数定义如下:

```
void ClearIndex(void)
{
    SQUARETYPE square;
    COLORTYPE col;
```

```

INDEXTYPE index;
int temp =0;

for (square = 0; square <= 0x98; square++)
    ChessBoard[square].index = 16;
    /*
在 C++Builder 中枚举常量加一个常数，不能成为后一个常量。
只得采用整数间接来达到目的。
*/
for (temp = 0; temp <= 1; temp++)
{if (temp==0)
    col = red;
    if (temp==1)
        col = black; }
for (index = 0; index < 16; index++)
    PieceTable[col][index].ipiece = empty;
OfficerNumber[red] = PawnNumber[red] = -1;
OfficerNumber[black] = PawnNumber[black] = -1;
}

```

3. 函数名称: void CalcPieceTable(void)

说明: 根据变化的情况计算棋子表(10 行 9 列)。

函数定义如下:

```

void CalcPieceTable(void)
{
    SQUARETYPE square;
    PIECETYPE piecel;
    int i;
    int temp;
    ClearIndex();
    /*
在 C++Builder 中枚举常量加一个常数，不能成为后一个常量。
只得采用整数间接来达到目的。
*/
for ( temp = 0; temp <= 6; temp++)
{
    {
        if (temp==0)

```

```

        piecel=king;
        if (temp==1)
            piecel=rook;
            if (temp==2)
                piecel=gunner;
                if (temp==3)
                    piecel=knight;
                    if (temp==4)
                        piecel=assist;
                        if(temp==5)
                            piecel=bishop;
                            if (temp==6)
                                piecel=pawn;
                                }

        if (piecel == pawn)
        {
            OfficerNumber[red] = PawnNumber[red];
            OfficerNumber[black] = PawnNumber[black];
        }
        square = 0;

        do
        {
            if (ChessBoard[square].piece == piecel)
            {
                PawnNumber[ChessBoard[square].color]++;

                PieceTable[ChessBoard[square].color][PawnNumber[ChessBoard[square].color]].
                ipiece

                    = piecel;

                PieceTable[ChessBoard[square].color][PawnNumber[ChessBoard[square].color]].
                isquare

                    = square;

                    ChessBoard[square].index =
                short (PawnNumber[ChessBoard[square].color]);
            }

            square ^= 0x77;

```

```

        if (!(square & 4))
        {
            if (square >= 0x70)
                square = (square + 0x11) & 0x73;
            else
                square += 0x10;
        }
    } while (square);
    for (i=0;i<=9;i++)
    {
        square=8+0x10*i;
        if (ChessBoard[square].piece == piece1)
        {
            PawnNumber[ChessBoard[square].color]++;

PieceTable[ChessBoard[square].color][PawnNumber[ChessBoard[square].color]].
ipiece
                = piece1;

PieceTable[ChessBoard[square].color][PawnNumber[ChessBoard[square].color]].
isquare
                = square;
            ChessBoard[square].index = short (PawnNumber[ChessBoard[square].
color]);
        }
    }

    for ( i=0;i<=7;i++)
    {
        square=0x80+i;
        if (ChessBoard[square].piece == piece1)
        {
            PawnNumber[ChessBoard[square].color]++;

PieceTable[ChessBoard[square].color][PawnNumber[ChessBoard[square].color]].
ipiece
                = piece1;

```

```

PieceTable[ChessBoard[square].color][PawnNumber[ChessBoard[square].color]].
isquare
        = square;
        ChessBoard[square].index = short (PawnNumber[ChessBoard[square].
color]);
    }
}

for ( i=0;i<=7;i++)
{
    square=0x90+i;
    if (ChessBoard[square].piece == piece1)
    {
        PawnNumber[ChessBoard[square].color]++;

PieceTable[ChessBoard[square].color][PawnNumber[ChessBoard[square].color]].
ipiece
        = piece1;

PieceTable[ChessBoard[square].color][PawnNumber[ChessBoard[square].color]].
isquare
        = square;
        ChessBoard[square].index = short (PawnNumber[ChessBoard[square].
color]);
    }
}
}
}

```

4. 函数名称: inline void MovePiece(SQUARETYPE newpos1, SQUARETYPE oldpos)

说明: 移动棋子到棋盘上新的位置。

函数定义如下:

```

inline void MovePiece(SQUARETYPE newpos1, SQUARETYPE oldpos)
{
    BOARDTYPE b;
    b = ChessBoard[newpos1];
    ChessBoard[newpos1] = ChessBoard[oldpos];
}

```

```

    ChessBoard[oldpos] = b;
    PieceTable[ChessBoard[newpos1].color][ChessBoard[newpos1].index].is
quare = newPos1;
}

```

5. 函数名称: inline void DeletePiece(SQUARETYPE insquare)

说明: 擦除棋子。这个函数在吃子时使用。要求擦除的方框内必须有一个棋子。

函数定义如下:

```

inline void DeletePiece(SQUARETYPE insquare)
{
    ChessBoard[insquare].piece = empty;
    PieceTable[ChessBoard[insquare].color][ChessBoard[insquare].index].
ipiece = empty;
}

```

6. 函数名称: inline void InsertPTabPiece(PIECETYPE inpiece, COLORTYPE incolor,

SQUARETYPE insquare)

说明: 在棋子表中插入棋子。退回到吃子前时使用此函数。

函数定义如下:

```

inline void InsertPTabPiece(PIECETYPE inpiece, COLORTYPE incolor,
    SQUARETYPE insquare)
{
    ChessBoard[insquare].piece
PieceTable[incolor][ChessBoard[insquare].index].ipiece
        = inpiece;
    ChessBoard[insquare].color = incolor;
    PieceTable[incolor][ChessBoard[insquare].index].isquare = insquare;
}

```

7. 函数名称: void Perform(MOVETYPE *move, bool resetmove)

说明: 如果允许回退的话, 在棋盘上退回一步。更改棋盘和棋子表(包括玩家和对方)。移动棋子、删除棋子、插入棋子和更改棋盘类型时都用到此函数。

函数定义如下:

```

void Perform(MOVETYPE *move, bool resetmove)
{

```

```
    if (resetmove)
    {
        MovePiece(move->oldpos, move->newpos1);
        if (move->content != empty)
            InsertPTabPiece(move->content, Opponent, move->newpos1);
    }
    else
    {
        if (move->content != empty)
            DeletePiece(move->newpos1);
        MovePiece(move->newpos1, move->oldpos);
    }
}
```

8. 函数名称: bool EqMove(MOVETYPE *a, MOVETYPE *b)

说明: 比较两次移动是否一样。

函数定义如下:

```
bool EqMove(MOVETYPE *a, MOVETYPE *b)
{
    if (a->movpiece == b->movpiece)
        if (a->newpos1 == b->newpos1)
            if (a->oldpos == b->oldpos)
                if (a->content == b->content)
                    if (a->spe == b->spe)
                        return true;
    return false;
}
```

9. 函数名称: void ClearInfoWindow()

说明: 清除信息窗口中的所有信息。

函数定义如下:

```
void ClearInfoWindow()
{
    InfoForm->Reset();
}
```


10. 函数名称: void ColorToPlay(COLORTYPE color)

说明: 显示当前对弈方。

函数定义如下:

```
void ColorToPlay(COLORTYPE color)
{
    if (color == red)
        InfoForm->SetTurnText("红方");
    else
        InfoForm->SetTurnText("黑方");
}
```

11. 函数名称: void Error(char *str)

说明: 错误报告。有多媒体音响时喇叭会发音。

函数定义如下:

```
void Error(char *str)
{
    if (SoundOn)
        MessageBeep(0);
    strcpy(buffer, str);
    SendMessage(MainForm->Handle, IDM_ERROR, 0, 0);
}
```

12. 函数名称: void Warning(char *str)

说明: 警告报告。有多媒体音响时喇叭会发音。

函数定义如下:

```
void Warning(char *str)
{
    if (SoundOn)
        MessageBeep(0);
    Message(str);
}
```

13. 函数名称: char *MoveStr(MOVETYPE *move)

说明: 把棋子移动用记谱术语表达出来。

函数定义如下:

```
char *MoveStr(MOVETYPE *move)
{
    static char str[10];
    static char *str0 = " ";
    static char *str1 = " ";
    static char *str2 = " ";
    static char *str3 = " ";
    int tmpint;
    strcpy(str, " ");
    if (move->movpiece != empty)
    {
        str[0] = PieceLetter[move->movpiece];
        str[1] = char('a' + move->oldpos % 16);
        str[2] = char('1' + move->oldpos / 16);
        if (move->content == empty)
            str[3] = '-';
        else
            str[3] = 'x';
        str[4] = char('a' + move->newpos1 % 16);
        str[5] = char('1' + move->newpos1 / 16);

        if ((str[0] == 'K') && (ChessBoard[move->oldpos].color == red))
            strcpy(str0, "帅");
        if ((str[0] == 'K') && (ChessBoard[move->oldpos].color == black))
            strcpy(str0, "将");
        if ((str[0] == 'A') && (ChessBoard[move->oldpos].color == red))
            strcpy(str0, "仕");
        if ((str[0] == 'A') && (ChessBoard[move->oldpos].color == black))
            strcpy(str0, "士");
        if ((str[0] == 'B') && (ChessBoard[move->oldpos].color == red))
            strcpy(str0, "相");
```

```
        if ((str[0] == 'B') && (ChessBoard[move->oldpos].color ==
black))

        lstrcpy(str0, "象");
    if (str[0] == 'N')
        lstrcpy(str0, "马");
        if (str[0] == 'R')
            lstrcpy(str0, "车");
    if ((str[0] == 'G') && (ChessBoard[move->oldpos].color == red))
        lstrcpy(str0, "炮");
        if ((str[0] == 'G') && (ChessBoard[move->oldpos].color ==
black))

        lstrcpy(str0, "炮");//黑方的炮实为异体字, 此处用正体字代
    if ((str[0] == 'P') && (ChessBoard[move->oldpos].color == red))
        lstrcpy(str0, "兵");
    if ((str[0] == 'P') && (ChessBoard[move->oldpos].color == black))
        lstrcpy(str0, "卒");
        if ((str[1] == 'a') && (ChessBoard[move->oldpos].color == red))
            lstrcpy(str1, "九");
        if ((str[1] == 'b') && (ChessBoard[move->oldpos].color == red))
            lstrcpy(str1, "八");
        if ((str[1] == 'c') && (ChessBoard[move->oldpos].color == red))
            lstrcpy(str1, "七");
        if ((str[1] == 'd') && (ChessBoard[move->oldpos].color == red))
            lstrcpy(str1, "六");
        if ((str[1] == 'e') && (ChessBoard[move->oldpos].color == red))
            lstrcpy(str1, "五");
        if ((str[1] == 'f') && (ChessBoard[move->oldpos].color == red))
            lstrcpy(str1, "四");
        if ((str[1] == 'g') && (ChessBoard[move->oldpos].color == red))
            lstrcpy(str1, "三");
        if ((str[1] == 'h') && (ChessBoard[move->oldpos].color == red))
            lstrcpy(str1, "二");
        if ((str[1] == 'i') && (ChessBoard[move->oldpos].color == red))
            lstrcpy(str1, "一");
        if ((str[1] == 'a') && (ChessBoard[move->oldpos].color == black))
            lstrcpy(str1, "1");
        if ((str[1] == 'b') && (ChessBoard[move->oldpos].color == black))
            lstrcpy(str1, "2");
```

```

        if ((str[1] == 'c') && (ChessBoard[move->oldpos].color ==
black))

        lstrcpy(str1, "3");
        if ((str[1] == 'd') && (ChessBoard[move->oldpos].color == black))
            lstrcpy(str1, "4");
        if ((str[1] == 'e') && (ChessBoard[move->oldpos].color == black))
            lstrcpy(str1, "5");
        if ((str[1] == 'f') && (ChessBoard[move->oldpos].color == black))
            lstrcpy(str1, "6");
        if ((str[1] == 'g') && (ChessBoard[move->oldpos].color == black))
            lstrcpy(str1, "7");
        if ((str[1] == 'h') && (ChessBoard[move->oldpos].color == black))
            lstrcpy(str1, "8");
        if ((str[1] == 'i') && (ChessBoard[move->oldpos].color == black))
            lstrcpy(str1, "9");

        if ((move->newpos1 > move->oldpos) && (ChessBoard[move-
>oldpos].color == red))
            if ((move->newpos1 >> 4) != (move->oldpos >> 4))
                lstrcpy(str2, "进");
        if ((move->newpos1 < move->oldpos) && (ChessBoard[move->oldpos].color
== black))
            if ((move->newpos1 >> 4) != (move->oldpos >> 4))
                lstrcpy(str2, "进");
                if ((move->newpos1 < move->oldpos) && (ChessBoard[move-
>oldpos].color == red))
                    if ((move->newpos1 >> 4) != (move->oldpos >> 4))
                        lstrcpy(str2, "退");
        if ((move->newpos1 > move->oldpos) && (ChessBoard[move->oldpos].color
== black))
            if ((move->newpos1 >> 4) != (move->oldpos >> 4))
                lstrcpy(str2, "退");
            if (move->newpos1 > move->oldpos)
        if ((move->newpos1 >> 4) == (move->oldpos >> 4))
            lstrcpy(str2, "平");
            if (move->newpos1 < move->oldpos)
        if ((move->newpos1 >> 4) == (move->oldpos >> 4))
            lstrcpy(str2, "平");

```

```
tmpint = 0;
if (((move->newpos1) >>4) == ((move->oldpos) >>4))
    tmpint = 9 - move->newpos1 % 16 ;
if (ChessBoard[move->oldpos].color == red)
    if ((move->movpiece == king ) || (move->movpiece == rook) ||
        (move->movpiece == gunner) || (move->movpiece == pawn ))
    {
        if (tmpint == 1)
            lstncpy(str3 , "一");
        if (tmpint == 2)
            lstncpy(str3 , "二");
        if (tmpint == 3)
            lstncpy(str3 , "三");
        if (tmpint == 4)
            lstncpy(str3 , "四");
        if (tmpint == 5)
            lstncpy(str3 , "五");
        if (tmpint == 6)
            lstncpy(str3 , "六");
        if (tmpint == 7)
            lstncpy(str3 , "七");
        if (tmpint == 8)
            lstncpy(str3 , "八");
        if (tmpint == 9)
            lstncpy(str3 , "九");
    }

tmpint = 0;
if (((move->newpos1) % 16) == ((move->oldpos) % 16))
    tmpint = abs((move->newpos1 >>4) - (move->oldpos >> 4));
if (ChessBoard[move->oldpos].color == red)
    if ((move->movpiece == king ) || (move->movpiece == rook) ||
        (move->movpiece == gunner) || (move->movpiece == pawn ))
    {
        if (tmpint == 1)
            lstncpy(str3 , "一");
        if (tmpint == 2)
            lstncpy(str3 , "二");
```

```
        if (tmpint == 3)
            lstrcpy(str3, "三");
        if (tmpint == 4)
            lstrcpy(str3, "四");
        if (tmpint == 5)
            lstrcpy(str3, "五");
        if (tmpint == 6)
            lstrcpy(str3, "六");
        if (tmpint == 7)
            lstrcpy(str3, "七");
        if (tmpint == 8)
            lstrcpy(str3, "八");
        if (tmpint == 9)
            lstrcpy(str3, "九");
    }

    tmpint = 0;

    if ((move->newpos1 >> 4) == (move->oldpos >> 4))
        tmpint = (move->newpos1 % 16 + 1);
    if (ChessBoard[move->oldpos].color == black )
        if ((move->movpiece == king ) || (move->movpiece == rook) ||
            (move->movpiece == gunner) || (move->movpiece == pawn ))
        {
            if (tmpint == 9)
                lstrcpy(str3, "9");
            if (tmpint == 8)
                lstrcpy(str3, "8");
            if (tmpint == 7)
                lstrcpy(str3, "7");
            if (tmpint == 6)
                lstrcpy(str3, "6");
            if (tmpint == 5)
                lstrcpy(str3, "5");
            if (tmpint == 4)
                lstrcpy(str3, "4");
            if (tmpint == 3)
                lstrcpy(str3, "3");
            if (tmpint == 2)
                lstrcpy(str3, "2");
```

```
    if (tmpint == 1)
        strcpy(str3, "1");
    }

    tmpint = 0;
    if ((move->newpos1 % 16) == (move->oldpos % 16))
        tmpint = abs((move->newpos1 >>4) - (move->oldpos >> 4));
    if (ChessBoard[move->oldpos].color == black )
    if ((move->movpiece == king ) || (move->movpiece == rook) ||
        (move->movpiece == gunner) || (move->movpiece == pawn ))
    {
        if (tmpint == 9)
            strcpy(str3, "9");
        if (tmpint == 8)
            strcpy(str3, "8");
        if (tmpint == 7)
            strcpy(str3, "7");
        if (tmpint == 6)
            strcpy(str3, "6");
        if (tmpint == 5)
            strcpy(str3, "5");
        if (tmpint == 4)
            strcpy(str3, "4");
        if (tmpint == 3)
            strcpy(str3, "3");
        if (tmpint == 2)
            strcpy(str3, "2");
        if (tmpint == 1)
            strcpy(str3, "1");
    }
    tmpint = 0;
    if ((move->newpos1 >>4) != ((move->oldpos >>4) && ((move->newpos1 %
16) != ((move->oldpos) % 16))
        tmpint = 9 - move->newpos1 % 16;
    if (ChessBoard[move->oldpos].color == red )
    if ((move->movpiece == assist ) || (move->movpiece == bishop) ||
        (move->movpiece == knight ))
    {
        if (tmpint == 1)
```

```
        strcpy(str3, "一");
    if (tmpint == 2)
        strcpy(str3, "二");
    if (tmpint == 3)
        strcpy(str3, "三");
    if (tmpint == 4)
        strcpy(str3, "四");
    if (tmpint == 5)
        strcpy(str3, "五");
    if (tmpint == 6)
        strcpy(str3, "六");
    if (tmpint == 7)
        strcpy(str3, "七");
    if (tmpint == 8)
        strcpy(str3, "八");
    if (tmpint == 9)
        strcpy(str3, "九");
    }
    tmpint = 0;
    if ((move->newpos1 >> 4) != ((move->oldpos) >> 4) && ((move->newpos1)
% 16) != ((move->oldpos) % 16))
        if (ChessBoard[move->oldpos].color == black)
            if ((move->movpiece == assist) || (move->movpiece == bishop)
                || (move->movpiece == knight ))
                tmpint = (move->newpos1 % 16 + 1);

        {
            if (tmpint == 1)
                strcpy(str3, "1");
            if (tmpint == 2)
                strcpy(str3, "2");
            if (tmpint == 3)
                strcpy(str3, "3");
            if (tmpint == 4)
                strcpy(str3, "4");
            if (tmpint == 5)
                strcpy(str3, "5");
            if (tmpint == 6)
                strcpy(str3, "6");
```



```
        if (tmpint == 7)
            strcpy(str3, "7");
        if (tmpint == 8)
            strcpy(str3, "8");
        if (tmpint == 9)
            strcpy(str3, "9");
    }
    strcpy(str, str0);
    strcat(str, str1);
    strcat(str, str2);
    strcat(str, str3);
}
}
return (str);
}
```

14. 函数名称: void PrintMove(int movenumber, COLORATYPE programcolor, MOVETYPE *move, double time)

说明: 在信息窗口显示着法记录
函数定义如下:

```
void PrintMove(int movenumber, COLORATYPE programcolor, MOVETYPE *move,
double time)
{
    int minutes;
    minutes = (int)(time / 60.0);

    sprintf(buffer, "%2.2d:%#04.1f%3.3d. %s", minutes, time - minutes * 60.0,
movenumber / 2 + 1, MoveStr(move));
    if (programcolor == red)
        InfoForm->SetRedInfoText(buffer);
    else
        InfoForm->SetBlackInfoText(buffer);
}
```

15. 函数名称: void DrawFrame(HDC hDC, RECT& ChessBoardRect, bool DrawBack ground)

说明: 使用 Windows API 函数画一个三维边框。

函数定义如下:

```
void DrawFrame(HDC hDC, RECT& ChessBoardRect, bool DrawBackground)
{
    int x1, y1, x2, y2;
    POINT pArray[3];
    HPEN hPen, hOldPen;
    HBRUSH hOldBrush;

    x1 = ChessBoardRect.left;
    x2 = ChessBoardRect.right;
    y1 = ChessBoardRect.top;
    y2 = ChessBoardRect.bottom ;
    if (DrawBackground == false)
        hOldBrush = (HBRUSH)SelectObject(hDC, GetStockObject(NULL_BRUSH));
    else
        hOldBrush = (HBRUSH)SelectObject(hDC, GetStockObject(LTGRAY_BRUSH));

    hOldPen = (HPEN)SelectObject(hDC, GetStockObject(WHITE_PEN));

    Rectangle(hDC, x1, y1, x2 , y2);
    SelectObject(hDC, hPen = CreatePen(PS_SOLID, 1, RGB(192, 192, 192)));
    Rectangle(hDC, x1+1, y1+1, x2 -1, y2 -1);
    DeleteObject(SelectObject(hDC, GetStockObject(WHITE_PEN)));

    pArray[0].x = x1 + 2;
    pArray[1].y = pArray[0].y = y2 - 3;
    pArray[2].x = pArray[1].x = x2 - 3;
    pArray[2].y = y1 + 2;

    Polyline(hDC, pArray, 3);
    hPen = CreatePen(PS_SOLID, 1, RGB(128, 128, 128));
    SelectObject(hDC, hPen);
    pArray[0].x = x1;
    pArray[1].y = pArray[0].y = y2-1;
    pArray[2].x = pArray[1].x = x2-1;
    pArray[2].y = y1;
```

```
Polyline(hDC, pArray, 3);
pArray[1].x = pArray[0].x = x1 + 2;
pArray[0].y = y2 - 3;
pArray[2].y = pArray[1].y = y1 + 2;
pArray[2].x = x2 - 3;
Polyline(hDC, pArray, 3);
SelectObject(hDC, hOldBrush);
DeleteObject(SelectObject(hDC, hOldPen));
}
```

16. 函数名称: void PrintCurLevel()

说明: 显示当前级别信息。

函数定义如下:

```
void PrintCurLevel()
{
    extern bool MultiMove;

    if (MultiMove)
        strcpy(buffer, "两人对弈");
    else
    {
        switch (Level)
        {
            case normal:
                sprintf(buffer, "%1.0f 秒/移动", AverageTime);
                break;
            case fullgametime:
                sprintf(buffer, "%2.2f 分钟/局", AverageTime);
                break;
            case easygame:
                strcpy(buffer, "容易");
                break;
            case infinite :
                strcpy(buffer, "无限制");
                break;
            case plysearch :
                sprintf(buffer, "回合深度 = %d", MaxLevel);
                break;
        }
    }
}
```

```

        case matesearch:
            strcpy(buffer, "将死搜索");
            break;
        case matching :
            strcpy(buffer, "匹配用户时间");
            break;
    }
}
InfoForm->SetLevelText(buffer);
}

```

17. 函数名称: POINT GetSquareXY(SQUARETYPE square)

说明: 获取棋格坐标 X, Y。

函数定义如下:

```

POINT GetSquareXY(SQUARETYPE square)
{
    POINT p;

    if (Turned)
    {
        square = (char)0xff ^ square - 0x99 ;
    }

    p.x = ((square & 0xf)) * SQUARE_SIZE + BORDERSIZE + MYFRAME_SIZE;
    p.y = (9 - ((square >> 4))) * SQUARE_SIZE + BORDERSIZE +
MYFRAME_SIZE/16+SQUARE_SIZE;

    return p;
}

```

18. 函数名称: void DrawChessBoard()

说明: 在屏幕上显示棋盘 10 行 9 列, 如 0x78 表示第 8 行第 9 列

函数定义如下:

```

void DrawChessBoard()
{
    unsigned char no;
    HDC hDC;

```

```

const SQUARETYPE printno[90] = { 0, 0x10, 0x20, 0x30, 0x40, 0x50, 0x60,
0x70, 0x80, 0x90,
                                0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98,
                                0x88, 0x78, 0x68, 0x58, 0x48, 0x38, 0x28, 0x18,
                                8,    7,    6,    5,    4,    3,    2,    1,
                                0x11, 0x21, 0x31, 0x41, 0x51, 0x61, 0x71, 0x81,
                                0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
                                0x77, 0x67, 0x57, 0x47, 0x37, 0x27, 0x17,
                                0x16, 0x15, 0x14, 0x13, 0x12,
                                0x22, 0x32, 0x42, 0x52, 0x62, 0x72,
                                0x73, 0x74, 0x75, 0x76,
                                0x66, 0x56, 0x46, 0x36, 0x26,
                                0x25, 0x24, 0x23,
                                0x33, 0x43, 0x53, 0x63,
                                0x64, 0x65,
                                0x55, 0x45, 0x35,
                                0x34, 0x44, 0x54};

```

```

ChessBoardRect.left = BORDERSIZE;
ChessBoardRect.top = 2.5* BORDERSIZE + 5;
ChessBoardRect.right = BORDERSIZE + (2 * MYFRAMESIZE) + (9 *
SQUARE_SIZE);
ChessBoardRect.bottom = BORDERSIZE + (2 * MYFRAMESIZE) + (11 *
SQUARE_SIZE)-3;
hDC = GetDC(MainForm->Handle);
DrawFrame(hDC, ChessBoardRect);
ReleaseDC(MainForm->Handle, hDC);
for (no = 0; no < 90; no++)
    ClearSquare(printno[no]);
}

```

19. 函数名称: void PrintPiece(SQUARETYPE square, PIECETYPE piece, COLOR
TYPE color, DWORD Rop)

说明: 显示棋子。

函数定义如下:

```
void PrintPiece(SQUARETYPE square, PIECETYPE piece, COLORTYPE color, DWORD
Rop)
{
    HBITMAP hBitmap, hOldBmp, hMaskBmp;
    BITMAP Bitmap;
    HDC hMemoryDC, hDC;
    POINT p;

    if (piece == empty)
        return;

    hBitmap = PieceBmpArray[piece-1][color];
    hMaskBmp = MaskArray[piece-1];

    hDC = GetDC(MainForm->Handle);
    hMemoryDC = CreateCompatibleDC(hDC);
    GetObject(hBitmap, sizeof(BITMAP), (LPSTR) &Bitmap);
    hOldBmp = (HBITMAP)SelectObject(hMemoryDC, hMaskBmp);
    p = GetSquareXY(square);

    BitBlt(hDC, p.x, p.y, Bitmap.bmWidth, Bitmap.bmHeight,
        hMemoryDC, 0, 0, SRCAND);
    SelectObject(hMemoryDC, hBitmap);
    BitBlt(hDC, p.x, p.y, Bitmap.bmWidth, Bitmap.bmHeight,
        hMemoryDC, 0, 0, Rop);
    SelectObject(hMemoryDC, hOldBmp);
    DeleteDC(hMemoryDC);
    ReleaseDC(MainForm->Handle, hDC);
}
```

20. 函数名称: void InitDisplay()

说明: 初始化显示。

函数定义如下:

```
void InitDisplay()
{
    SQUARETYPE square;
    for (square = 0; square <= 0x98; square++)
```

```

        if (((square>>4)<10) && ((square % 16)<9) && ((square | 0xff)>=0))
            if ((ChessBoard[square].piece != Display[square].piece) ||
                (ChessBoard[square].piece != empty) &&
                (ChessBoard[square].color !=
                 Display[square].color))
            {
                Display[square].piece = ChessBoard[square].piece;
                Display[square].color = ChessBoard[square].color;
            }
    }
}

```

21. 函数名称: static void FrameSquare(SQUARETYPE square, bool Defended)

说明: 在受攻击的棋子边上画一个矩形框。如果受到攻击又没有保护的话, 显示红框, 如果受到攻击但有保护的话, 显示黑框。

函数定义如下:

```

static void FrameSquare(SQUARETYPE square, bool Defended)
{
    POINT p;
    HPEN hOldPen;
    HBRUSH hOldBrush;
    HDC hDC;
    hDC = GetDC(MainForm->Handle);

    p = GetSquareXY(square);
    hOldBrush = (HBRUSH)SelectObject(hDC, GetStockObject(NULL_BRUSH));
    if (Defended)
        hOldPen = (HPEN)SelectObject(hDC, GetStockObject(BLACK_PEN));
    else
        hOldPen = (HPEN)SelectObject(hDC, CreatePen(PS_SOLID, 1, RGB(192, 0,
0)));

    Rectangle(hDC, p.x+1, p.y+1, p.x+SQUARE_SIZE-4, p.y+SQUARE_SIZE-1);
    SelectObject(hDC, hOldBrush);
    if (Defended)
        SelectObject(hDC, hOldPen);
    else
        DeleteObject(SelectObject(hDC, hOldPen));
}

```

```
ReleaseDC(MainForm->Handle, hDC);  
}
```

22. 函数名称: void HideAttacks(void)

说明: 对受到攻击的棋子隐藏边框。

函数定义如下:

```
void HideAttacks(void)  
{  
    SQUARETYPE square;  
  
    for (square = 0; square <= 0x98; square++)  
        if (((square>>4)<10) && ((square % 16)<9) && ((square | 0xff)>=0))  
        {  
            if (ChessBoard[square].attacked == true)  
            {  
                ChessBoard[square].attacked = false;  
                ClearSquare(square);  
                PrintPiece(square, ChessBoard[square].piece,  
                    ChessBoard[square].color, SRCINVERT);  
            }  
        }  
}
```

23. 函数名称: void ShowAttacks()

说明: 对受到攻击的棋子显示边框。

函数定义如下:

```
void ShowAttacks()  
{  
    SQUARETYPE square;  
    for (square = 0; square <= 0x98; square++)  
        if (((square>>4)<10) && ((square % 16)<9) && ((square | 0xff)>=0))  
        {  
            if (Attacks(ComputerColor, square) && ChessBoard[square].color !=  
ComputerColor && ChessBoard[square].piece != empty)  
            {  
                ChessBoard[square].attacked = true;  
                if (Attacks((COLORTYPE)!ComputerColor, square))
```



```

        FrameSquare(square, true);
    else
        FrameSquare(square, false);
    }
    else if (ChessBoard[square].attacked == true)
    {
        ChessBoard[square].attacked = false;
        ClearSquare(square);
        PrintPiece(square, ChessBoard[square].piece,
            ChessBoard[square].color, SRCINVERT);
    }
}
}

```

24. 函数名称: void UpdateChessBoard()

说明: 根据选项重画棋盘。

函数定义如下:

```

void UpdateChessBoard()
{
    SQUARETYPE square;
    for (square = 0; square <= 0x98; square++)
        if (((square>>4)<10) && ((square % 16)<9) && ((square | 0xff)>=0))
            if ((ChessBoard[square].piece != Display[square].piece) ||
                (ChessBoard[square].piece != empty) && (ChessBoard[square].
color !=
                Display[square].color))
            {
                if (Display[square].piece != empty)
                    ClearSquare(square);
                Display[square].piece = ChessBoard[square].piece;
                Display[square].color = ChessBoard[square].color;
                if (ChessBoard[square].piece != empty)
                    PrintPiece(square, ChessBoard[square].piece,
                        ChessBoard[square].color, SRCINVERT);
            }
    if (Level == easygame && !Editing)
        ShowAttacks();
}

```

25. 函数名称: void PrintChessBoard()

说明: 打印棋盘。包括棋盘、棋子、记录棋子位置的行列数字。

函数定义如下:

```
void PrintChessBoard()
{
    SQUARETYPE square;

    DrawChessBoard();
    for (square = 0; square <= 0x98; square++)
        if (((square>>4)<10) && ((square % 16)<9) && ((square | 0xff)>=0))
        {
            if (Display[square].piece != empty)
                PrintPiece(square, Display[square].piece,
                    Display[square].color, SRCINVERT);
        }
    DrawAlphaNum();
    if (Level == easygame && !Editing)
        ShowAttacks();
}
```

26. 函数名称: SQUARETYPE GetValidSquare(POINT p, COLORTYPE player, bool CheckPiece)

说明: 确定鼠标位置所在方格中的棋子可否移动。

函数定义如下:

```
SQUARETYPE GetValidSquare(POINT p, COLORTYPE player, bool CheckPiece)
{
    POINT point;
    SQUARETYPE square;
    RECT srect;

    for (square = 0; square <= 0x98; square++)
    {
        if (((square>>4)<10) && ((square % 16)<9) && ((square | 0xff)>=0))
        {
            point = GetSquareXY(square);
```

```
    sqrect.left = point.x;
    sqrect.top = point.y;
    sqrect.right = sqrect.left + SQUARE_SIZE;
    sqrect.bottom = sqrect.top + SQUARE_SIZE;
    if (PtInRect(&sqrect, p))
    {
        if ((Display[square].color == player && Display[square].piece
            != empty) || !CheckPiece)
            return square;
    }
}
return -1;
}
```

27. 函数名称: void DrawNormalBitmap(SQUARETYPE square)

说明: 画一个正常的位图。

函数定义如下:

```
void DrawNormalBitmap(SQUARETYPE square)
{
    ClearSquare(square);
    PrintPiece(square, Display[square].piece, Display[square].color,
SRCINVERT);
}
```

28. 函数名称: void DrawInvertedBitmap(SQUARETYPE square)

说明: 画一个反相的位图。

函数定义如下:

```
void DrawInvertedBitmap(SQUARETYPE square)
{
    PrintPiece(square, Display[square].piece, Display[square].color,
NOTSRCERASE);
}
```

29. 函数名称: void PrintNodes(NODEVAL *nodes, double time)

说明: 在信息窗口中显示搜索树的节点。

函数定义如下:

```
void PrintNodes(NODEVAL *nodes, double time)
{
    double nodereal;
    char buffer[80];

    nodereal = (nodes->nodebase * MAXINT) + nodes->nodeoffset;
    if (time)
    {
        sprintf(buffer, "%7.1f", nodereal/time);
        InfoForm->SetSecondsText(buffer);
    }
    sprintf(buffer, "%7.0f ", nodereal);
    InfoForm->SetNodeText(buffer);
}
```

30. 函数名称: void PrintBestMove(MOVETYPE *mainpath, MAXTYPE mainevalu)

说明: 在信息窗口中显示最好走法。

函数定义如下:

```
void PrintBestMove(MOVETYPE *mainpath, MAXTYPE mainevalu)
{
    DEPTHTYPE dep = 0;

    *buffer = 0;

    if (ShowBestPath == false)
        return;
    while (dep < 7 && (mainpath[dep].movpiece != empty))
    {
        strcat(buffer, MoveStr(&mainpath[dep++]));
        strcat(buffer, " ");
    }
    InfoForm->SetBestPathText(buffer);
    sprintf(buffer, "%7.2f", mainevalu / 256.0);
    InfoForm->SetValueText(buffer);
}
```

31. 函数名称: void RemoveBestPath()

说明: 清除信息窗口中的最好路线。

函数定义如下:

```
void RemoveBestPath()
{
    InfoForm->SetBestPathText("");
}
```

32. 函数名称: static void DrawBump(HDC hDC, int x, int y)

说明: 画一个凸起来的框。

函数定义如下:

```
static void DrawBump(HDC hDC, int x, int y)
{
    int x2, y2;
    HPEN hOldPen;
    HBRUSH hOldBrush;
    POINT pArray[3];

    x2 = x + 2 * CHARSIZE + 4;
    y2 = y-- + LINESIZE + 1;
    x-=2;

    hOldBrush = (HBRUSH)SelectObject(hDC, GetStockObject(LTGRAY_BRUSH));
    hOldPen = (HPEN)SelectObject(hDC, GetStockObject(WHITE_PEN));

    Rectangle(hDC, x, y, x2, y2);

    SelectObject(hDC, CreatePen(PS_SOLID, 1, RGB(128, 128, 128)));
    pArray[0].x = ++x;
    pArray[1].y = pArray[0].y = y2;
    pArray[2].x = pArray[1].x = x2;
    pArray[2].y = ++y;
    Polyline(hDC, pArray, 3);

    SelectObject(hDC, hOldBrush);
    DeleteObject(SelectObject(hDC, hOldPen));
}
```

33. 函数名称: static void DrawAlphaNum()

说明: 显示数字和文字。注意中文是以 2 字节方式存储的。

函数定义如下:

```
static void DrawAlphaNum()
{
    int i;
    int XPos, YPos;
    HDC hDC;

    hDC = GetDC(MainForm->Handle);

    XPos = SQUARE_SIZE / 2 - CHARSIZE/2+SQUARE_SIZE/3;
    YPos = SQUARE_SIZE - 8 ;

    SetBkColor(hDC, RGB(192, 192, 192));

    for (i = 0; i < 9; i++)
    {
        DrawBump(hDC, XPos, YPos);
        if (Turned)
        {
            TextOut(hDC, XPos, YPos, NumArray[8-i], 2);
        }
        else
            TextOut(hDC, XPos, YPos, NumArray[i], 2);
        XPos += SQUARE_SIZE;
    }

    XPos = SQUARE_SIZE / 2 - CHARSIZE/2+SQUARE_SIZE/3;
    YPos = BORDERSIZE + (9 * SQUARE_SIZE) + (2 * MYFRAMESIZE) +
    1+2*SQUARE_SIZE;

    for (i = 0; i < 9; i++)
    {
        DrawBump(hDC, XPos , YPos + SQUARE_SIZE / 8 );
        if (Turned)
```

```

    TextOut(hDC, XPos, YPos + SQUARE_SIZE / 8, CharArray[i], 2);
    else
    TextOut(hDC, XPos, YPos + SQUARE_SIZE / 8, CharArray[8-i], 2);
    XPos += SQUARE_SIZE;
}
ReleaseDC(MainForm->Handle, hDC);
}

```

34. 函数名称: static bool SaveChessBoard(char *savefile)

说明: 存储棋盘。按自定义棋盘数据结构以二进制方式保存。

函数定义如下:

```

static bool SaveChessBoard(char *savefile)
{
    FILE *gameboard;
    SAVERESTORE *Save;

    Save = new SAVERESTORE;
    if (!Save)
    {
        MessageBox(Application->Handle, "没有足够的内存执行操作",
            "中国象棋", MB_OK | MB_ICONHAND);
        return 0;
    }
    if ((gameboard = fopen(savefile, "wb")) == NULL)
    {
        sprintf(buffer, "不能打开 %s 用于写", savefile);
        MessageBox(Application->Handle, buffer, "中国象棋", MB_OK |
MB_ICONHAND);
        delete Save;
        return 0;
    }

    Save->Player = Player;
    Save->Opponent = Opponent;
    Save->ProgramColor = ProgramColor;
    Save->Turned = Turned;
    Save->MultiMove = MultiMove;
    Save->AutoPlay = AutoPlay;
}

```

```
Save->SingleStep = SingleStep;
Save->Level = Level;
Save->AverageTime = AverageTime;
Save->MaxLevel = MaxLevel;
Save->MoveNumber = MoveNumber;
Save->ChessClockTotalTime = ChessClock.totaltime;
Save->BlackTotalTime = ChessTime[black].totaltime;
Save->RedTotalTime = ChessTime[red].totaltime;
memcpy(Save->PieceValue, PieceValue, 8 * sizeof(int));
fwrite(Save, sizeof(SAVERESTORE), 1, gameboard);
fwrite(ChessBoard, sizeof(ChessBoard), 1, gameboard);
fclose(gameboard);
delete Save;
return 1;
}
```

35. 函数名称: void LoadChessBoard(char *loadfile)

说明: 加载棋盘。按自定义棋盘数据结构以二进制方式打开。
函数定义如下:

```
void LoadChessBoard(char *loadfile)
{
    FILE *load;
    SAVERESTORE *Restore;

    Restore = new SAVERESTORE;

    if (!Restore)
    {
        MessageBox(MainForm->Handle, "没有足够的内存执行操作",
            "中国象棋", MB_OK | MB_ICONHAND);
        return;
    }

    if ((load = fopen(loadfile, "rb")) == 0)
        Error("文件没有发现");
    else
    {
        fread(Restore, sizeof(SAVERESTORE), 1, load);
    }
}
```



```

        fread(ChessBoard, sizeof(ChessBoard), 1, load);
        fclose(load);
        Player = Restore->Player;
        Opponent = Restore->Opponent;
        ProgramColor = Restore->ProgramColor;
        Turned = Restore->Turned;
        MultiMove = Restore->MultiMove;
        AutoPlay = Restore->AutoPlay;
        SingleStep = Restore->SingleStep;
        Level = Restore->Level;
        AverageTime = Restore->AverageTime;
        MaxLevel = Restore->MaxLevel;
        MoveNumber = Restore->MoveNumber;
        ChessClock.totaltime = Restore->ChessClockTotalTime;
        ChessTime[black].totaltime = Restore->BlackTotalTime ;
        ChessTime[red].totaltime = Restore->RedTotalTime;
        memcpy(PieceValue, Restore->PieceValue, 8 * sizeof(int));
        ClearDisplay();
        InitDisplay();
        MainForm->Invalidate();

        PrintCurLevel();
        ResetNewPos();
    }
    delete Restore;
}

```

36. 函数名称: void InitChessTime()

说明: 初始化象棋时钟。

函数定义如下:

```

void InitChessTime()
{
    InitTime(&ChessTime[red]);
    InitTime(&ChessTime[black]);
}

```

37. 函数名称: void StopChessTime()

说明: 停止象棋时钟。

函数定义如下:

```
void StopChessTime()
{
    if (Running)
    {
        StopTime(&ChessTime[RunColor]);
        MainForm->Timer1->Enabled=false;
        Running = false;
    }
}
```

38. 函数名称: void StartChessTime(COLORTYPE color)

说明: 停止运行的象棋时钟, 开始另一方的时钟。

函数定义如下:

```
void StartChessTime(COLORTYPE color)
{
    RunColor = color;
    Running = true;
    StartTime(&ChessTime[RunColor]);
    MainForm->Timer1->Enabled =true;
}
```

39. 函数名称: void ResetMoves()

说明: 重置移动表。

函数定义如下:

```
void ResetMoves()
{
    Depth = -1;
    MoveTable[-1] = ZeroMove;
}
```

40. 函数名称: void RemoveHint()

说明: 清除提示行。

函数定义如下:

```
void RemoveHint()
{
    HintPath[0] = ZeroMove;
    HintEvaluat = 0;
}
```

41. 函数名称: void InitNode(NODEVAL *nodes)

说明: 初始化搜索树节点。

函数定义如下:

```
void InitNode(NODEVAL *nodes)
{
    nodes->nodebase = 0;
    nodes->nodeoffset = 0;
}
```

42. 函数名称: bool IllegalMove(MOVETYPE *move)

说明: 检查当 Programcolor == player 时在给定的位置棋子移动是否合法。

函数定义如下:

```
bool IllegalMove(MOVETYPE *move)
{
    bool illegal;

    Perform(move, 0); //向前看一步
    illegal = Attacks(Opponent, PieceTable[Player][0].isquare);
    Perform(move, 1); //退回一步
    return illegal;
}
```

43. 函数名称: void MakeMove(MOVETYPE *move)

说明: 当 Programcolor == player 时, 移动棋子并更改变量。

函数定义如下:

```
void MakeMove(MOVETYPE *move)
{
    Depth++;
    MoveNumber++;
}
```

```
Perform(move, 0); //向前看一步
ProgramColor = Opponent;
Opponent = Player;
Player = ProgramColor;
}
```

44. 函数名称: void PrintComment(void)

说明: 显示游戏状态(如将军、将死、平局等)。

函数定义如下:

```
void PrintComment(void)
{
    extern char buffer[];
    short check, possiblemove, checkmate;
    int nummoves;

    Message("");
    checkmate = 0;
    Depth++;
    possiblemove = 0;
    InitMovGen();
    do
    {
        MovGen();
        if (Next.movpiece != empty)
            if (!IllegalMove(&Next))
                possiblemove = 1;
    }while (Next.movpiece != empty && !possiblemove);

    Depth--;
    check = Attacks(Opponent, PieceTable[Player][0].isquare); /* 计算将军 */
    /* 当将死或和棋时, 应当不会有可能移动的棋子 */
    if (!possiblemove)
    {
        if (check)
        {
            checkmate = 1;
            EndMessage("将死!");
        }
    }
}
```

```

        else
            EndMessage("死局!");
    }
    else
        if (HintEvaluat >= MATEVALUE - DEPTHFACTOR * 16)
        {
            nummoves = (MATEVALUE - HintEvaluat + 0x40) / (DEPTHFACTOR * 2);
            sprintf(buffer, " 将死在%d 次移动后%c", nummoves, (nummoves > 1) ?
'!': '!');
            Message(buffer);
        }

    if (check && !checkmate)
        Message("将军!");
    else /* 重复移动 60 回合规则 */
    {
        if (SixtyMoveCnt() >= 120)
        {
            EndMessage("六十回合规则");
        }

        else /* 子力差别太大, 举手投降 */
            if (-25500 < HintEvaluat && HintEvaluat < -0x1550)
            {
                switch (Opponent)
                {
                    case red :
                        EndMessage("红方认输");
                        break;
                    case black :
                        EndMessage("黑方认输");
                }
            }
    }

    if (possiblemove)
    if (check)
        MessageBox(NULL, "将军!", "中国象棋", MB_OK | MB_ICONEXCLAMATION);

```

```
}
```

45. 函数名称: void EnterMove(MOVETYPE *move)

说明: 进入移动。

函数定义如下:

```
void EnterMove(MOVETYPE *move)
{
    StopChessTime();
    PrintMove(MoveNumber, ProgramColor, move, ChessTime[RunColor].
totaltime);
    MakeMove(move);
    UpdateChessBoard();
    PrintComment();
    StartChessTime(ProgramColor);
}
```

46. 函数名称: void RemoveMove(MOVETYPE *move)

说明: 删除移动。

函数定义如下:

```
void RemoveMove(MOVETYPE *move)
{
    StopChessTime();
    PrintMove(MoveNumber, ProgramColor, move, ChessTime[RunColor].
totaltime);
    TakeBackMove(move);
    UpdateChessBoard();
    PrintComment();
    StartChessTime(ProgramColor);
}
```

47. 函数名称: void EnterKeyMove()

说明: 用户干预进行棋子移动。

函数定义如下:

```
void EnterKeyMove()
{
    MoveTable[Depth+1] = KeyMove;
```

```
PlayerMove = KeyMove;
RemoveHint();
DragEnd(true);
EnterMove(&MoveTable[Depth+1]);
}
```

48. 函数名称: void AdjustMoves()

说明: 退回一步, 移动表相应发生变化。

函数定义如下:

```
void AdjustMoves()
{
    int i;

    for (i = Depth; i >= BACK; i--)
        MoveTable[i - (Depth+1)] = MoveTable[i];
    Depth = -1;
}
```

49. 函数名称: void StoreMoves()

说明: 把退回的每一步保存起来。

函数定义如下:

```
void StoreMoves()
{
    int i;
    Depth--;
    for (i = BACK; i <= Depth; i++)
        MoveTable[i] = MoveTable[i+1];
    MoveTable[BACK] = ZeroMove;
}
```

50. 函数名称: bool MoveCheck(SQUARETYPE startsq, SQUARETYPE endsq)

说明: 检查棋子移动是否合法。

函数定义如下:

```
bool MoveCheck(SQUARETYPE startsq, SQUARETYPE endsq)
{
    Depth++;
```

```
KeyMove = ZeroMove;
InitMovGen();
do
{
    MovGen();
    if (Next.newpos1 == endsq && Next.oldpos == startsq)
    {
        KeyMove = Next;
        break;
    }
} while (Next.movpiece != empty);
if (KeyMove.movpiece == empty)
{
    Warning("不可能的移动!");
    Depth--;
    return false;
}
if (IllegalMove(&KeyMove))
{
    Warning("非法移动。将军!");
    Depth--;
    return false;
}
Depth--;
if (!ComputerThinking)
{
    AdjustMoves();
    EnterKeyMove();
    StoreMoves();
}
return true;
}
```

51. 函数名称: void StartAnalysis()

说明: 计算出计算机作出反应的时间。

函数定义如下:

```
void StartAnalysis()
{
```



```
int timecontrol;

Analysis = 1;
Opan = 0;
Screen->Cursor = TCursor(crWaitCursor); //动态分配光标给应用程序
switch (Level)
{
    case easygame :
    case normal :
        /* 留出一点时间备用。留出 4 次移动的时间确保程序不要错过时间 */
        timecontrol = (((MoveNumber >> 1) + 20) / 20) * 20;
        if (timecontrol <= 40) timecontrol = 40;
        WantedTime = (AverageTime * timecontrol -
                      ChessTime[ProgramColor].totaltime) /
                      (timecontrol + 4 - (MoveNumber >> 1));
        /* 因为开局和中局通常是一局棋中最重要的部分，
           所以开始时程序认为应多化一倍的时间*/
        if ((MoveNumber >> 1) <= 40)
            WantedTime = 5.0 + (WantedTime - 5.0) *
                          ((80 - (MoveNumber >> 1)) / 40);
        break;
    case fullgametime :
        /* 假定游戏能走 40 步，所以把时间除以 44 */
        WantedTime = (AverageTime * 60.0 -
                      ChessTime[ProgramColor].totaltime) / 44;

        /* 因为开局和中局通常是一局棋中最重要的部分，
           所以开始时程序认为应多化一倍的时间*/
        if ((MoveNumber >> 1) <= 40)
            WantedTime = 5.0 + (WantedTime - 5.0) *
                          ((80 - (MoveNumber >> 1)) / 40);
        break;
    case matching :
        /* 根据对方的动作来确定时间 */
        if (MoveNumber >= 2)
            WantedTime = ChessTime[Opponent].totaltime / (MoveNumber >>
1);
        else
            WantedTime = 5.0;
```

```
        WantedTime += (ChessTime[Opponent].totaltime -
                        ChessTime[ProgramColor].totaltime) * 0.25;
    default :
        WantedTime = 1000000.0;
    }
}
```

52. 函数名称: void TakeBackMove(MOVETYPE *move)

说明: 回退一步并修改变量。

函数定义如下:

```
void TakeBackMove(MOVETYPE *move)
{
    ProgramColor = Opponent;
    Opponent = Player;
    Player = ProgramColor;
    Perform(move, 1); //退回一步
    MoveNumber--;
    Depth--;
}
```

53. 函数名称: void AddNode(NODEVAL *nodes)

说明: 相应改变棋路树的节点数。

函数定义如下:

```
void AddNode(NODEVAL *nodes)
{
    if (nodes->nodeoffset >= MAXINT)
    {
        nodes->nodebase++;
        nodes->nodeoffset = 0;
    }
    else
        nodes->nodeoffset++;
}
```

54. 函数名称: void Wait(int tenths)

说明: 必须留出足够的时间来画图, 这样使画图过程显得较光滑。

函数定义如下:

```
void Wait(int tenths)
{
    clock_t NumTicksToWait;

    /*
    时钟每秒 18.2 次, 大约两次滴嗒相当于 0.1 秒
    */
    NumTicksToWait = (tenths * 2) + clock();
    while (NumTicksToWait > clock()) ;
}
```

55. 函数名称: void FlashMove(MOVETYPE *move)

说明: 当程序移动一个棋子时, 在原位置和目标位置来回闪动一下。

函数定义如下:

```
void FlashMove(MOVETYPE *move)
{
    MakeMove(move);
    UpdateChessBoard();
    Wait(4);
    TakeBackMove(move);
    UpdateChessBoard();
    Wait(4);
}
```

56. 函数名称: void DisplayHint()

说明: 显示提示信息。对应每一个深度有一个最大值, 相应有一个棋子走法, 即最好走法。

函数定义如下:

```
void DisplayHint()
{
    DEPTHTYPE dep = 0;
    buffer[0] = '\0';
    Message(buffer);
    while (HintPath[dep].movpiece != empty)
    {
        strcat(buffer, MoveStr(&HintPath[dep]));
    }
}
```

```
    strcat(buffer, " ");
    Message(buffer);
    MakeMove(&HintPath[dep]);
    UpdateChessBoard();
    Wait(6);
    dep++;
}
while (dep > 0)
{
    dep--;
    TakeBackMove(&HintPath[dep]);
}
UpdateChessBoard();
}
```

57. 函数名称: void InsertPiece(PIECETYPE p, COLORTYPE c, SQUARETYPE sq)

说明: 插入棋子, 更改棋盘表。

函数定义如下:

```
void InsertPiece(PIECETYPE p, COLORTYPE c, SQUARETYPE sq)
{
    ChessBoard[sq].piece = p;
    ChessBoard[sq].color = c;
}
```

58. 函数名称: void ClearPVTable()

说明: 清除棋子值表。

函数定义如下:

```
void ClearPVTable()
{
    COLORTYPE color;
    PIECETYPE piece;
    SQUARETYPE square;
    int colortemp, piecetemp;
    /*
在 C++Builder 中枚举常量加一个常数, 不能成为后一个常量。
只得采用整数间接来达到目的。
*/
}
```

```

    for (colortemp = 0; colortemp <= 1; colortemp++)
    {
        { if (colortemp==0)
            color = red;
          if (colortemp==1)
            color = black;
        }

        /*
在 C++Builder 中枚举常量加一个常数，不能成为后一个常量。
只得采用整数间接来达到目的。
*/
    for (piecetemp = 0; piecetemp <= 6; piecetemp++)
    {
        {
            if (piecetemp==0)
                piece=king;
            if (piecetemp==1)
                piece=rook;
            if (piecetemp==2)
                piece=gunner;
            if (piecetemp==3)
                piece=knight;
            if (piecetemp==4)
                piece=assist;
            if (piecetemp==5)
                piece=bishop;
            if (piecetemp==6)
                piece=pawn;
        }

        for (square = 0; square <= 0x98; square++)
            PVTable[color][piece][square] = 0;
    }
}

```

59. 函数名称: void ResetGame()

说明: 重新开始新的游戏。在棋盘上摆好棋子。

函数定义如下:

```
void ResetGame()
{
    int i;
    ClearChessBoard();
    Running = false;
    for ( i = 0; i < 9; i++)
    {
        InsertPiece(Pieces[i], red, i);
        InsertPiece(Pieces[i], black, i + 0x90);
    }
    for (i=0; i<5; i++)
    {
        InsertPiece(pawn, red, 2*i+0x30);
        InsertPiece(pawn, black, 2*i+0x60);
    }
    InsertPiece(gunner, red, 0x21);
    InsertPiece(gunner, red, 0x27);
    InsertPiece(gunner, black, 0x71);
    InsertPiece(gunner, black, 0x77);
    CalcPieceTable();
    Player = red;
    ClearDisplay();
    InitDisplay();
    ColorToPlay(Player);
    Opponent = black;
}
```

60. 函数名称: void NewChessGame()

说明: 重新开始新的游戏。初始化棋钟, 开始游戏。

函数定义如下:

```
void NewChessGame()
{
    SingleStep = InLibrary = false;
    GameOver = false;
    ResetGame();
    PrintCurLevel();
}
```

```

ResetMoves();
if (!*OpeningSequ)
    UseLib = 0;
else
    UseLib = 200;
MoveTable[-1].content = king;
InitChessTime();
ProgramColor = red;
MoveNumber = 0;
RemoveHint();
ClearPVTable();
PlayerMove = ZeroMove;
if (Logging && !AutoPlay)
*OutputFile << "\n No   Player Program      Hint      Value Level      Nodes
Time\n";
    InitNode(&Nodes);
    ChessClock.totaltime = 0.0;
    Control = (AutoPlay) ? gamemove : readmove;
}

```

61. 函数名称: static void StartUp()

说明: 程序开始运行时, 读取初始化文件中的数据来初始化变量。

函数定义如下:

```

static void StartUp()
{
    long color;
    randomize();

    Level = (LEVELTYPE)GetPrivateProfileInt("CHESS", "Level",
        (int)easygame, INIFile);

    if (GetPrivateProfileString("CHESS", "AverageTime", "5.0", buffer,
        80, INIFile))
        sscanf(buffer, "%f", &AverageTime);

    if (GetPrivateProfileString("CHESS", "ColorSquare", "", buffer, 80,
        INIFile))
    {

```

```

        sscanf(buffer, "%ld", &color);
        ColorSquareColors[0] = (BYTE)(color & 0xff);
        ColorSquareColors[1] = (BYTE)(((color & 0xff00L) >> 8));
        ColorSquareColors[2] = (BYTE)(((color & 0xff0000L) >> 16));
    }

    MaxLevel = (BYTE)GetPrivateProfileInt("CHESS", "MaxLevel", MAXPLY,
    INIFile);
    SoundOn = (bool)GetPrivateProfileInt("CHESS", "SoundOn", 1, INIFile);

    if (!SoundOn) // 程序开始运行时, 默认状态声音开关是打开的
        MainForm->Sound->Checked=true,
    extern void CalcAttackTable();
    CalcAttackTable();
    MultiMove = false;
    AutoPlay = false;
    Turned = false;
    ResetOpening();
    OutputFile = new ofstream("Chess.log");
    if (!OutputFile)
    {
        MessageBox(NULL, "不能打开日志文件.\n 禁止登录!", "中国象棋", MB_OK |
        MB_ICONEXCLAMATION);
        Logging = false;
    }
    else
    {
        *OutputFile << endl;
        *OutputFile << "          China CHESS by SiCong\n";
        *OutputFile << "          =====\n" <<
endl;
    }
}

```

62. 函数名称: void OutputNode(NODEVAL *nodes)

说明: 输出节点。

函数定义如下:

```
void OutputNode(NODEVAL *nodes)
```



```

{
    double nodereal;
    if (!Logging)
        return;
    char buffer[20];
    nodereal = (nodes->nodebase * MAXINT) + nodes->nodeoffset;
    sprintf(buffer, "%12.1f", nodereal);
    *OutputFile << buffer;
}

```

63. 函数名称: void ReturnAnalysis()

说明: 对找到的着法进行分析。对找到的好棋在棋盘上显示出来, 对非法的着法表示拒绝, 同时向信息框中输出信息。

函数定义如下:

```

void ReturnAnalysis()
{
    int myx;
    char str[8];
    char buffer[40];

    MoveTable[0] = MainPath[0]; /* 保存到提示信息中 */
    for (myx = 1; myx <= MAXPLY; myx++)
        HintPath[myx - 1] = MainPath[myx];
    dep = MAXPLY;
    HintEvaluat = MainEvaluat;
    if (MoveTable[0].movpiece == empty)
    {
        HintPath[0] = ZeroMove; /* 不可能的移动 */
        if (AutoPlay)
        {
            NewChessGame();
            PrintChessBoard();
            StartMove();
        }
        return;
    }

    FlashMove(&MoveTable[Depth+1]); /* 来回闪动两下后就执行走步 */
}

```

```

EnterMove(&MoveTable[Depth+1]);
if (SoundOn)
    MessageBeep(0);
StoreMoves();
if (Logging && !AutoPlay)
{
    sprintf(buffer, "%3d. ", (MoveNumber + 1) / 2);
    *OutputFile << buffer;
    strcpy(str, MoveStr(&MoveTable[0]));
    if ((PlayerMove.movpiece == empty) && (Opponent == red))
        sprintf(buffer, "%8.8s ", str);
    else
        sprintf(buffer, "%s%8.8s", MoveStr(&PlayerMove), str);
    *OutputFile << buffer;
    sprintf(buffer, "      (%s)%9.2f%3d:%2d", MoveStr(&MainPath[1]),
        MainEvaluat / 256.0, MaxDepth, LegalMoves);
    *OutputFile << buffer;
    OutputNode(&Nodes);
    sprintf(buffer, "%8.11f\n", ChessClock.totaltime);
    *OutputFile << buffer;
}
PlayerMove = ZeroMove;
ColorToPlay(Player);
if (AutoPlay)
{
    if ((MoveNumber >= 120) || (SixtyMoveCnt() >= 60) ||
        (Repetition(0) >= 3) || (MainEvaluat <= -0x1550))
    {
        NewChessGame();
        PrintChessBoard();
    }
    StartMove();
    return;
}
if (Level != easygame && !GameOver)
    ThinkAwhile();
}

```

64. 函数名称: static void ThinkAwhile()

说明: 在对方想着法的同时开始分析对策。程序假定对方下一步是根据提示着法走棋, 计算出相应的着法。

函数定义如下:

```
static void ThinkAwhile()
{
extern HCURSOR hArrowCursor;

    Screen->Cursor = TCursor(crArrow); //分配光标给应用程序
    MainForm->Menu=MainForm->TChessMenu; //动态设置主窗体主菜单
    if ((HintPath[0].movpiece == empty) || MultiMove)
        return;
    Analysis = false;
    Opan = true;
    AdjustMoves();
    /*程序假定对方根据提示着法走棋设置环境*/
    MoveTable[Depth+1] = HintPath[0];
    MakeMove(&MoveTable[Depth+1]);
    StoreMoves();
    AdjustMoves();
    Depth = 0;          /* 一旦用户干预就停止分析 */
    SearchMove(MaxLevel);
    Depth = -1;
    Opan = false;
    /*如果对方确定是根据提示着法走棋了, 那么就执行计划*/
    if (Analysis)
    {
        ReturnAnalysis();
        return;
    }
    TakeBackMove(&MoveTable[Depth]); /* 恢复环境 */
    if (GotValidMove)
    {
        AdjustMoves();
        EnterKeyMove();
        StoreMoves();

        MainForm->Menu=MainForm->TChessThinkMenu; //动态设置主窗体主菜单
        Screen->Cursor = TCursor(crWaitCursor); //动态分配光标给应用程序
```

```
    }  
    return;  
}
```

65. 函数名称: void StartMove()

说明: 开始移动。注意此处消息函数的参数的使用。

函数定义如下:

```
void StartMove()  
{  
    MSG msg;  
    StartAnalysis();  
    AdjustMoves();  
    CalcLibNo();          /* 试图从棋谱库中找到着法 */  
    Depth = 0;  
    if (LibNo > 0)  
    {  
        OpeningLibMsg();  
        InLibrary = true;  
        FindOpeningMove();  
        if (::PeekMessage(&msg, 0, 0, 0, PM_REMOVE)) //初始化 MSG 消息结构  
        {  
            /*点按“停止”菜单, 注意其 ID 号为 MainForm->Stop->Command */  
            if (msg.message == WM_COMMAND && (msg.wParam -  
                (MainForm->Stop->Command)) == 0)  
                return;  
            ::TranslateMessage(&msg);  
            ::DispatchMessage(&msg);  
        }  
    }  
    else  
    {  
        if (InLibrary)  
        {  
            InLibrary = false;  
            RemoveMessage();  
        }  
        SearchMove(MaxLevel);  
    }  
}
```

```
Depth = -1;
ReturnAnalysis();
}
```

66. 函数名称: void ProgramMove()

说明: 程序移动。

函数定义如下:

```
void ProgramMove()
{
    do
    {
        GotValidMove = false;
        ColorToPlay(Player);
        StartMove();
    }
    while (GotValidMove);
}
```

67. 函数名称: bool Redo()

说明: 判断是否可以重下。

函数定义如下:

```
bool Redo()
{
    MakeMove(&MoveTable[Depth+1]);
    EnterMove(&MoveTable[Depth+1]);
    RemoveHint();
    UpdateChessBoard();
    RemoveBestPath();
    ColorToPlay(Player);
    if (Depth >= -1)
        return false; // 不能再重下
    return true; // 可以再重下
}
```

68. 函数名称: bool Undo()

说明: 判断是否可以悔棋。

函数定义如下:

```
bool Undo()
{
    TakeBackMove(&MoveTable[Depth]);
    extern void RemoveMove(MOVETYPE *);
    RemoveMove(&MoveTable[Depth]);
    RemoveHint();
    UpdateChessBoard();
    RemoveBestPath();
    ColorToPlay(Player);
    if (MoveTable[Depth].movpiece == empty)
        return false; // 不能再悔棋
    return true; // 可以悔棋
}
```

69. 函数名称: void QuitProgram()

说明: 退出程序时往初始化文件中写数据。

函数定义如下:

```
void QuitProgram()
{
    long color;

    if (Logging)
        OutputFile->close();
    color = RGB(ColorSquareColors[0], ColorSquareColors[1],
        ColorSquareColors[2]);
    sprintf(buffer, "%ld", color);
    WritePrivateProfileString("CHESS", "ColorSquare", buffer, INIFile);
    sprintf(buffer, "%d", Level);
    WritePrivateProfileString("CHESS", "Level", buffer, INIFile);
    sprintf(buffer, "%lf", AverageTime);
    WritePrivateProfileString("CHESS", "AverageTime", buffer, INIFile);
    sprintf(buffer, "%d", (int)MaxLevel);
    WritePrivateProfileString("CHESS", "MaxLevel", buffer, INIFile);
    sprintf(buffer, "%d", (int)SoundOn);
    WritePrivateProfileString("CHESS", "SoundOn", buffer, INIFile);
}
```

70. 函数名称: void SearchHintMove()

说明: 从搜索中发现提示移动。

函数定义如下:

```
void SearchHintMove()
{
    /* 如果提示行是空的, 就从棋谱库中搜索一个着法, 如果没有找到,
    就执行一个回合的搜索
    */
    if (HintPath[0].movpiece == empty)
    {
        AdjustMoves();
        CalcLibNo();
        Depth = 0;
        if (LibNo > 0)
            FindOpeningMove();
        else
        {
            Analysis = 1;
            Opan = 0;
            SearchMove(1);
        }
        Depth = -1;
        HintPath[0] = MainPath[0];
        HintPath[1] = ZeroMove;
        HintEvaluat = -MainEvaluat;
    }
}
```

71. 函数名称: void CGetTime(int *hour, int *minute, int *second, int * sec100)

说明: 获取系统时钟(时、分、秒、毫秒)。

函数定义如下:

```
void CGetTime(int *hour, int *minute, int *second, int * sec100)
{
    SYSTEMTIME systime;
    GetLocalTime(&systime);
    *hour = systime.wHour;
```

```
*minute = systime.wMinute;  
*second = systime.wSecond;  
*sec100 = systime.wMilliseconds * 10;  
  
}
```

72. 函数名称: void DisplayTime()

说明: 显示棋钟时间。

函数定义如下:

```
void DisplayTime()  
{  
    int hour1, min1, sec1, s100;  
    static bool Colon = true;  
    double CurTime = ChessTime[RunColor].totaltime;  
    CGetTime(&hour1, &min1, &sec1, &s100);  
    //s100 乘以 0.01 本应该是 1 秒, 可运行时 1 秒变为 5 秒。  
    //为了避免出现时间错误, 作者只好乘以零, 干脆忽略该项。  
    CurTime += ((s100 - ChessTime[RunColor].sec100) * 0.01 * 0 +  
                (sec1 - ChessTime[RunColor].second) +  
                (min1 - ChessTime[RunColor].minute) * 60.0 +  
                (hour1 - ChessTime[RunColor].hour) * 3600.0);  
    min1 = int(CurTime / 60.0);  
    sprintf(buffer, "%2.2d%s%02d", min1, (Colon) ? ":" : " ", int(CurTime  
- min1 * 60.0));  
    InfoForm->SetTimerText(buffer);  
    Colon = !Colon;  
}
```

73. 函数名称: void InitTime(CHESSCLOCKTYPE *clock)

说明: 初始化棋钟数据结构。

函数定义如下:

```
void InitTime(CHESSCLOCKTYPE *clock)  
{  
    memset(clock, 0, sizeof(CHESSCLOCKTYPE));  
}
```


74. 函数名称: void StopTime(CHESSCLOCKTYPE *clock)

说明: 停止棋钟。

函数定义如下:

```
void StopTime(CHESSCLOCKTYPE *clock)
{
    int hour1, min1, sec1, s100;
    CGetTime(&hour1, &min1, &sec1, &s100);
    if (hour1 < clock->hour) hour1+=24;
    clock->totalhours += (hour1 - clock->hour);
    clock->totmins += (min1 - clock->minute);
    clock->totalsecs += (sec1 - clock->second);
    clock->tots100 += (s100 - clock->sec100);
    clock->number++;
    //s100 乘以 0.01 本应该是 1 秒, 可运行时 1 秒变为 5 秒。
    //为了避免出现时间错误, 作者只好乘以零, 干脆忽略该项。
    clock->totaltime = clock->totaltime + ((s100 - clock->sec100) * 0.01
    * 0 +
        (sec1 - clock->second) + (min1 - clock->minute) * 60.0 +
        (hour1 - clock->hour) * 3600.0);
    clock->hour = hour1;
    clock->minute = min1;
    clock->second = sec1;
    clock->sec100 = s100;
}
```



读者回执卡

欢迎亲立即填妥寄回

您好！感谢您购买本书，请您抽出宝贵的时间填写这份回执卡，并将此页剪下寄回我公司读者服务部。我们会在以后的工作中充分考虑您的意见和建议，并将您的信息加入公司的客户档案中，向您提供全程的一体化服务。读者服务包括：

- ★ 免费获得我公司的新书资料；
- ★ 可免费参加我公司组织的技术交流会及讲座；
- ★ 可优惠邮购我公司的所有图书；
- ★ 可参加不定期的促销活动，免费获取赠品；

读者基本资料

姓 名 _____ 性 别 ☐男 ☐女 年 龄 _____
 电 话 _____ 职 业 _____ 文化程度 _____
 E-mail _____ 邮 编 _____
 通讯地址 _____

请在您认可处打√（5至9题可多选）

1. 您购买的图书名称是什么：_____
2. 您对电脑的掌握程度：
 - ☐不懂
 - ☐基本掌握
 - ☐熟练应用
 - ☐精通某一领域
3. 您学习此书的主要目的是：
 - ☐工作需要
 - ☐个人爱好
 - ☐获得证书
4. 您希望通过学习达到何种程度：
 - ☐基本掌握
 - ☐熟练应用
 - ☐专业水平
5. 您想学习的其他电脑知识有：
 - ☐电脑入门
 - ☐操作系统
 - ☐办公软件
 - ☐多媒体设计
 - ☐编程知识
 - ☐图像设计
 - ☐网页设计
 - ☐互联网知识
6. 影响您购买图书的因素：
 - ☐书名
 - ☐作者
 - ☐出版机构
 - ☐印刷、装帧质量
 - ☐内容简介
 - ☐网络宣传
 - ☐图书定价
 - ☐书店宣传
 - ☐封面、插图及版式
 - ☐知名作家（学者）的推荐或书评
 - ☐其他
7. 您比较喜欢哪些形式的学习方式：
 - ☐看图书
 - ☐上网学习
 - ☐用教学光盘
 - ☐参加培训班
8. 您可以接受的图书的价格是：
 - ☐20 元以内
 - ☐30 元以内
 - ☐50 元以内
 - ☐100 元以内
9. 您从何处获知本公司产品信息：
 - ☐报纸、杂志
 - ☐广播、电视
 - ☐同事或朋友推荐
 - ☐网站
10. 您对本书的满意度：
 - ☐很满意
 - ☐较满意
 - ☐一般
 - ☐不满意
11. 您对我们的建议：_____

请剪下本页填写清楚，放入信封寄回，谢谢！

1 0 0 0 8 4

北京市100084—157信箱

读者服务部

收

贴 票 邮 处

邮政编码：□□□□□□