symbian Developer Network

Symbian OS 专题研究

Andreas Jakl

由 Symbian Developer Network 出版

版本 1.0 – 2007 年 8 月

序

本教程介绍使用 C++开发手机应用程序的一些基础知识，所有示例都是面向 Symbian OS 和 Series 60 UI 平台的。

本教程不进行冗长的理论解释，而采用更实用的方式。

在本教程中，读者首先将学习如何创建手机项目；然后，将一个预先写好的类似于 Arkanoid 的小游戏 Mopoid 的游戏逻辑添加到项目中。通过这种有趣的方法，让读者了解在 Symbian OS 上进行开发的一些重要方面，其中包括：

- 定义与使用菜单；
- 处理与显示文本；
- 加载与显示图像；
- 读写文件；
- 使用定时器处理周期性事件；
- 以及其他很多较小但同样重要的主题。

读者可以马上看到自己动手的结果。涉及 Symbian OS 特有的方面（如内存处理）时，将简要地解释其中的原理和用法。

当然，这些主题都只能涉及表面，因为每个主题都足以占据一个教程的篇幅。然而，通过阅读本教程，读者将对如何使用 C++进行手机开发有基本了解，读者可以以此为起点开发自己的项目。

希望读者在本教程中学习愉快，并能够据此创建不错的游戏！

Andreas Jakl

第 0 步：准备工作

下载下列组件：

- ActivePerl：Symbian OS 工具链使用它来编译项目（http://www.activestate.com/）。
- Symbian OS SDK：本教程使用 Series 60 SDK 1.2 for Symbian OS, Nokia Edition，以便最大限度地同老式 Series 60 设备兼容（http://www.symbian.com/developer/sdks_series60.asp）。
- Microsoft Debugging Tools：用于通过 Borland's C++BuilderX 在仿真器中调试应用程序（http://www.microsoft.com/whdc/devtools/debugging/）。
- Visual C++ Toolkit:：仅当没有安装 Visual Studio 6 或.net 时才需要。它是免费的，可为仿真器编译项目。请访问 http://www.microsoft.com/downloads/，然后搜索 Visual C++ Toolkit 以下载这个工具。
- Borland C++BuilderX Mobile Edition (v1.5)：在 http://info.borland.com/survey/cbx15_mobile_edition.html 填写简短的调查后就可免费下载。

按上面的顺序安装这些组件。安装时允许将程序添加到系统的 path 变量中，以备不时之需！建议将 Series 60 SDK 安装到默认位置 C:\Symbian\!

开始

在仿真器中的应用程序崩溃时，有一种方法可让仿真器显示更有帮助的错误消息。要启用这项功能，可在 Series 60 SDK 目录 C:\Symbian\6.1\Series60\Epoc32\Wins\c\system\Bootdata 中创建一个名为 ErrRd 的空文件。这样就可以了。

首次启用 C++BuilderX Mobile Edition 时，需要注册。如果已经在 Borland 开发人员网络上填写调查并注册，它将发送一个文件给您。选择 activation file 选项，然后找到 Borland 发送给您的这个文件。

选择 IDE

Symbian OS 的编译工具实际上是基于命令行的，在没有任何 IDE 时也管用。然而，如果使用 IDE，开发将更轻松。下表列出了几个可供使用的 IDE，都有其优缺点[1]：

Microsoft Visual C++ 6	优点：速度快；编译与调试效率高；有一些用于 Symbian OS 的工具。 缺点：较旧；缺少对 Symbian OS 代码开发的真正支持。
Microsoft Visual Studio .net	优点：速度快；不错的现代 IDE；得到广泛使用。 缺点：Symbian OS SDK 不直接支持它，这使得 Symbian OS 开发更复杂，费用更高。
Metrowerks CodeWarrior	优点：至少集成了部分 Symbian OS。 缺点：不是很便宜；在刚开始时该 IDE 比较难以掌握。
Borland C++BuilderX Mobile Edition[2]	优点：支持菜单与对话框的可视化设计；具有与 Symbian OS 相关的一些特性；目前是免费的！ 缺点：速度较慢，性能不够好；还需进一步集成 Symbian OS。
Eclipse	优点：很不错的 IDE；免费！ 缺点：没有调试支持；尚未真正集成 Symbian OS。

可以看到，开发 Symbian OS C++应用程序没有最佳的方法。本教程选用 Borland C++BuilderX，因为它是免费的，且在开始使用 Symbian OS 时，使用它可以省去很多麻烦，虽然该 IDE 本身还有很大的改进空间。

---

1 所有产品的商标都归其所有者所有。

2 本教程将其称为 C++BuilderX。

请注意：上表只是个人意见。CodeWarrior 也是很不错的选择，尤其考虑到 Nokia 购买了它的移动版本，将来应该会有很大的改进。

选择 Series 60 SDK

较高版本的 SDK 支持 Series 60 SDK for Symbian OS，后者面向较新的 Series 60 手机。到目前为止，所有设备都是向后兼容的，因此为 Symbian OS v6.x 手机（使用 SDK v1.2）开发的应用程序在较新的 Symbian OS v7.0 和 v8.0 手机中也可运行。

虽然使用 SDK v2.0 编译的应用程序（主要面向装有 Symbian OS v7.0 的 Nokia 6600）在较旧的手机上也应该能够运行，但可能有点小问题。如果源代码是用更新的 SDK 编译的，二进制文件将不再向后兼容。

使用哪个版本的 SDK 完全取决于读者。如果只想在较新的手机上进行开发，或者只是想做一些以前不可能实现的事情，则（至少）应使用 SDK v2.0。在这个版本中，图像和声音的加载已经过专门改进；Internet（HTTP）连接也更容易处理。

如果要针对尽可能多的用户开发商业应用程序，则不应将较旧的 Series 60 手机排除在外。选择针对 Series 60 就已经限制了潜在的市场，如果再让程序只适用于最新的手机，销售起来更加困难。

第 1 步：定义 Symbian OS SDK

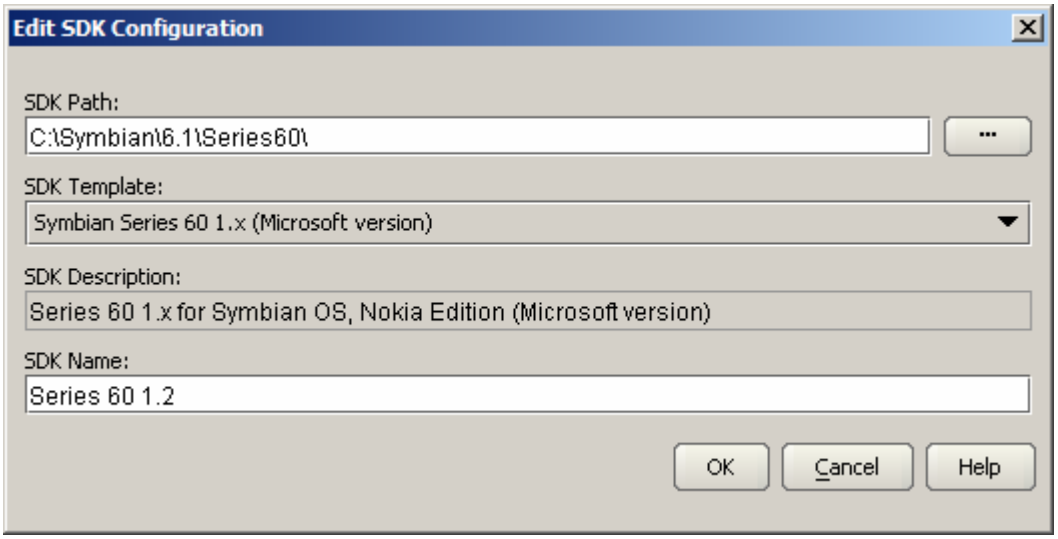


图 1 在 Borland C++BuilderX 中定义 Symbian OS SDK

现在开始工作！打开 Borland C++BuilderX Mobile Edition。选择菜单 Tools→Symbian SDK Configuration，使用安装 SDK 的 SDK 模板（这里为 Symbian Series 60 1.x (Microsoft version)）创建一种新的 SDK 配置。对于 SDK v1.2，不要使用 Borland SDK 版本，否则项目将无法编译！按图 1 设置路径，并选择合适的名称，如 Series 60 1.2。

第 2 步：新建项目

创建一个新项目：File→New…→Series 60→Series 60 GUI Application。

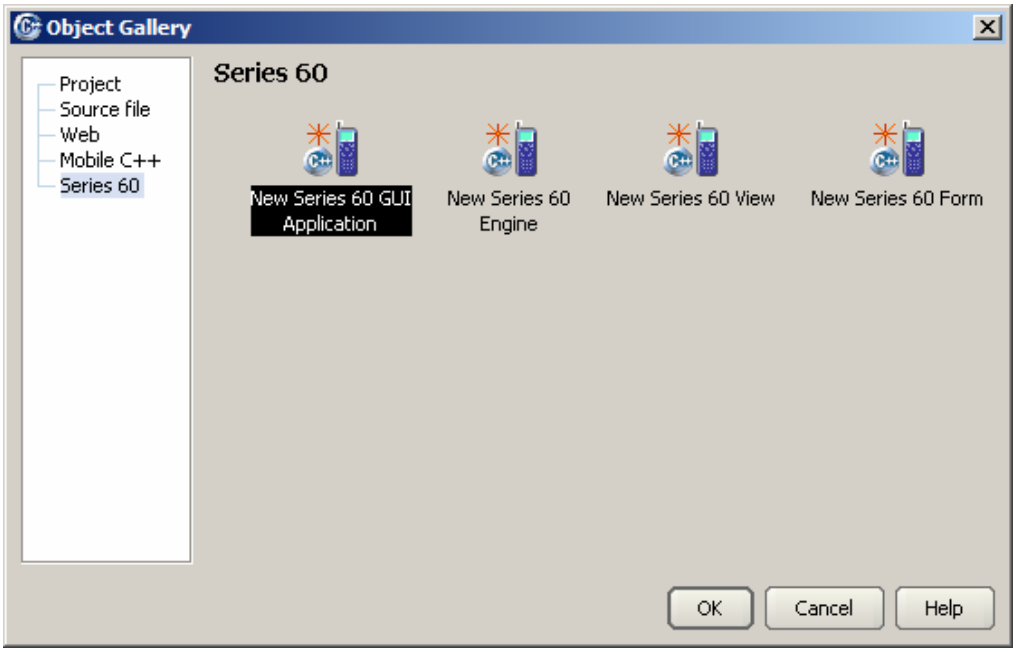


图 2 新建一个 Symbian C++项目

输入 Mopoid 作为项目名（就这个项目而言，不要使用其他名称，否则包含一些预先写好的文件时将不管用）。将它放在 C:/Symbian/dev 目录中，并让 C++BuilderX 创建一个项目子目录(如图 2 所示)，以便将所有项目彼此分开。

接下来输入项目名称 Mopoid，由于要编写一个游戏，因此将 View Type 设置为 Full Screen。

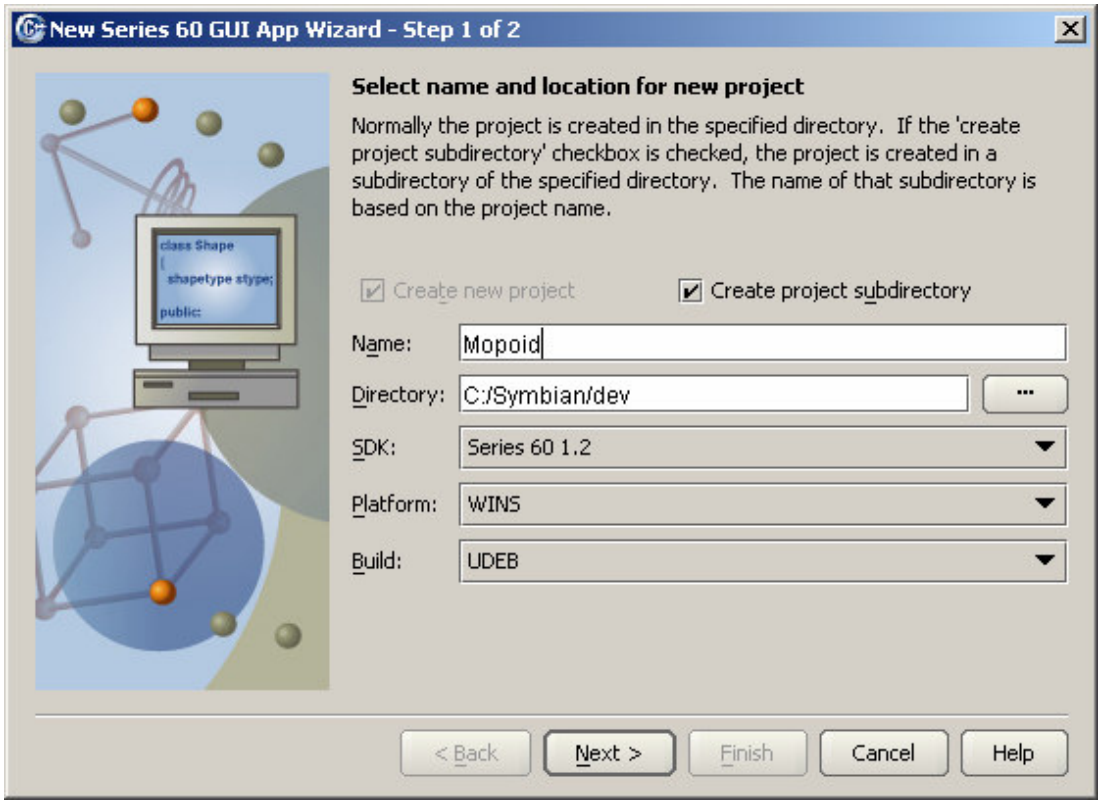


图 3 定义项目属性（第 1 步）

不要使用 C++BuilderX 提供的默认 UID3 值，而将其改为 0x01000000 到 0x0FFFFFFF 之间的值——Symbian 将这些 ID 保留用于测试项目。安装在手机上的每个应用程序都必须有唯一的 UID。因此，如果要向公众发布自己的游戏，应向 mailto:uid@symbiandevnet.com发送邮件，并在邮件中指出您的姓名以及需要多少个 UID（开始时 5 个就足够了），他们将尽快把 UID 发送给您。

图 4 定义项目属性（第 2 步）

第 3 步：测试项目

此时的工作区应类似于图 5。接下来需要测试项目，看配置是否正确。为此，单击 Run Project 按钮（F9）。

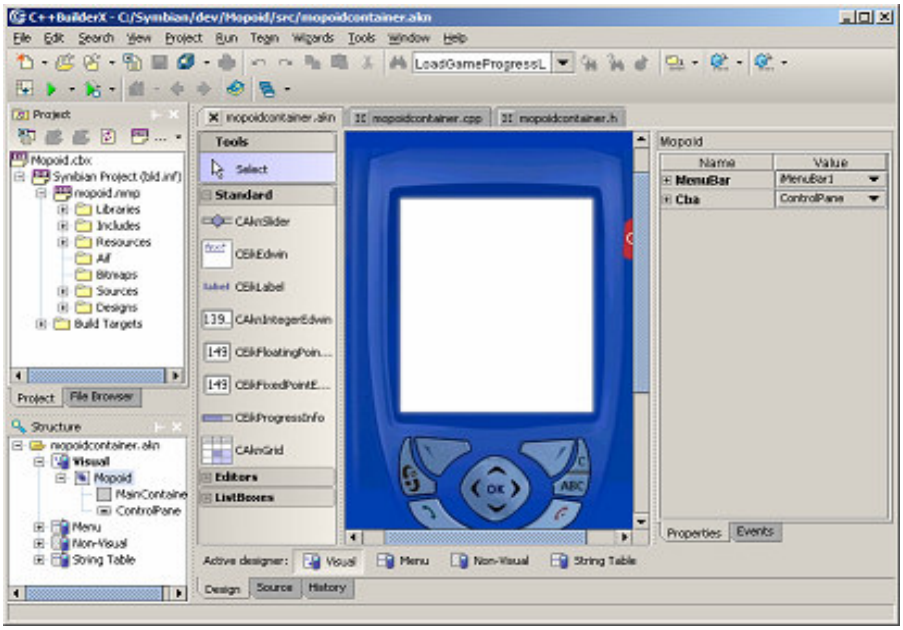


图 5 Borland C++BuilderX 的默认工作区

如果一切正常，过段时间后仿真器将显示出来。如果有常驻内存的病毒扫描程序在运行，可将其关闭，这样该过程将更快些。应用程序位于菜单的底部，因此使用光标键向下移动以选择它。

如果不想每次测试游戏时都这样做，可按左边的软键（Options），选择 Move，并将其移到菜单的左上角。它将始终停留在这个位置，即使关闭并重启仿真器。

运行 Mopoid 应用程序时，将看到一个白色屏幕。这是由于创建的是一个全屏应用程序，因此没有可见的状态栏和菜单栏。按下左边的软键将显示默认菜单，接下来的几步将根据需要对它进行调整。

排除故障

如果计算机安装了 Microsoft VisualStudio.net 或 Microsoft 工具包，Borland C++BuilderX 把该编译器用于 Series 60 SDK v1.2，可能出现下面的错误消息：

```
LNK2019: unresolved external symbol __ftol2
```

要解决这个问题的方法是，在文本编辑器中打开文件C:\Symbian\6.1\Shared\EPOC32\Tools\cl_win.pm，搜索包含/W4 的行，将它改为CLFLAGS = /nologo /Zp4 /W4 /QIfist（即将/QIfist 参数添加到该选项中）。[3]

如果在编译过程中 C++BuilderX 显示如下错误：

```
Can't locate E32env.pm in @INC [...]
```

一种可能的解决方案是将下面两个路径放在 path 环境变量的开头：

```
C:\Symbian\6.1\Shared\epoc32\gcc\bin;C:\Symbian\6.1\Shared\epoc32\tools;
```

如果这样还不能解决问题，确保安装了 Perl 后，对 Symbian OS SDK 进行修复安装，这旨在确保系统中有 Perl 后再安装 SDK。

测试完项目后别忘了关闭仿真器窗口！如果编译时仿真器还在后台运行，将出现多条错误消息！

第 4 步：定义字符串

默认情况下，MopoidContainer.akn 文件应处于活动状态。如果没有，通过左边的项目窗口打开它：Mopoid.cbx→Symbian project (bld.inf)Mopoid.mmp→Designs→MopoidContainer.akn（双击）。

要创建菜单项，首先必须定义要使用的文本。在 Symbian OS 中，文本通常是在资源文件中定义的。这样更易于对应用程序进行本地化，因为手机是全球通用的，所以希望游戏能支持多种语言。切换到 String Table 设计器。

[3] 感谢 Simon Woodside 提供这种技巧——参见 http://simonwoodside.com/weblog/2004/07/18。

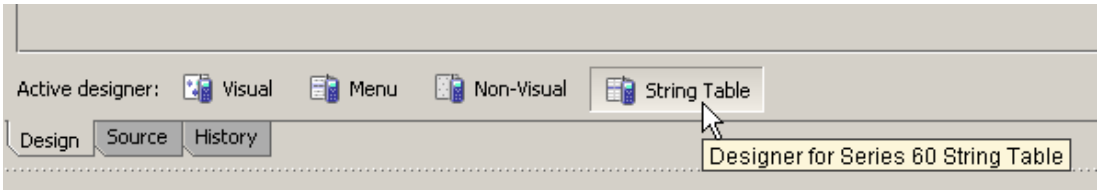
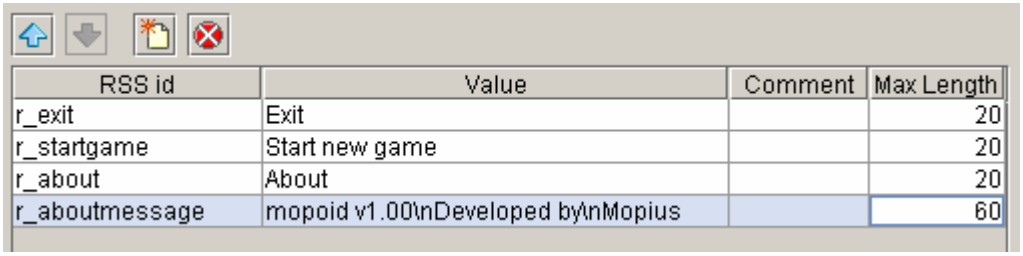


图 6 切换到 String Table 设计器

这个游戏有一个简单的菜单，能够让用户开始玩游戏、显示 About 对话框以及退出游戏。因此，定义下面四个字符串：



RSS id	Value	Comment	Max Length
r_exit	Exit		20
r_startgame	Start new game		20
r_about	About		20
r_aboutmessage	mopoid v1.00\nDeveloped by\nMopius		60

图 7 在应用程序的资源文件中添加新字符串

对于 About 对话框中的消息（r_aboutmessage），首先必须将最大长度（Max Length）增大到 60 左右，以便有足够的空间存储消息。使用\n 告诉系统从哪里开始换行。在单元格中填写好文本后按回车键，以确保 C++BuilderX 保存新输入的文本。

请注意：对于本教程，要求按如上所述正确地输入名称。后面将添加一些预先写好的源文件到项目中，这些源文件使用了这些名称。

第 5 步：定义菜单

接下来在游戏中使用这些字符串。切换到 Menu 设计器，将发现 C++BuilderX 已经创建了两个菜单项。这个游戏还需要一个菜单项，因此，选择左边窗格中的 MenuItem 工具，然后在菜单中单击，在其末尾添加一个新菜单项：

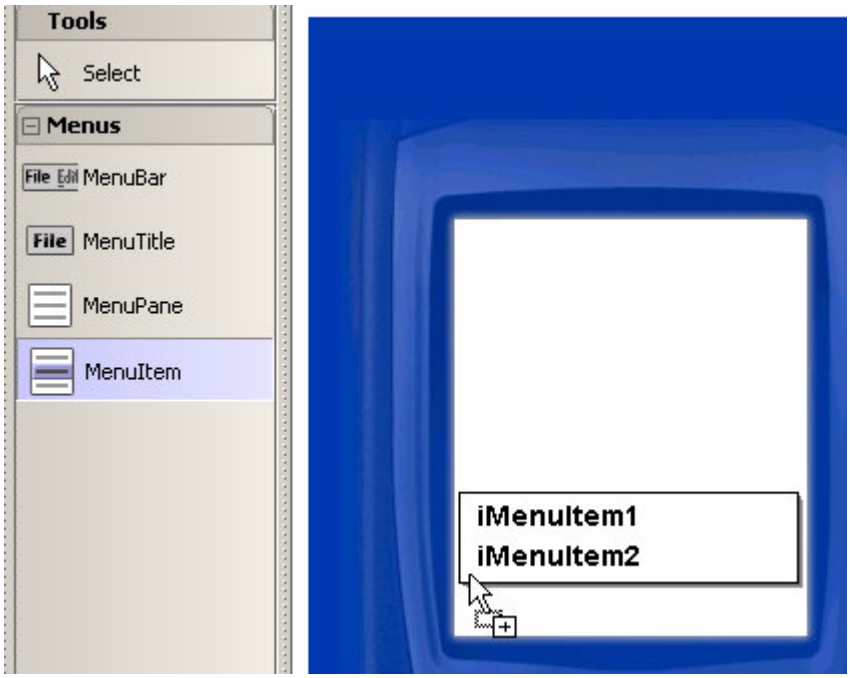


图 8 使用设计工具添加新菜单项

接下来调整菜单项，使其使用前面定义的文本。

单击 iMenuItem1 选中它，在右边的窗格中将其名称改为 iMenuItemNewGame。至于它的文本，只要选择资源 r_startgame 作为其 ID。保留默认的 Command 值 1001。

请注意：修改了值后一定要按回车键，这很重要。如果不按回车键切换到其他地方，输入的文本可能不会被保存。

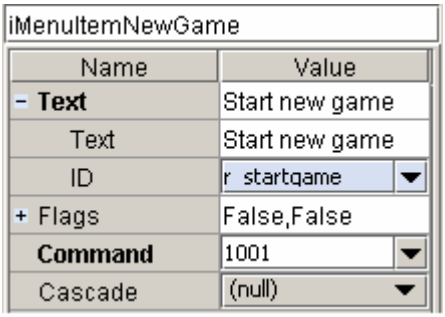


图 9 设置菜单项的属性

接下来，将 iMenuItem2 的名称改为 iMenuItemAbout；选择 r_about 作为其 ID。Command 值 1002 不需要改动。另外，选中第 2 个标志 EEikMenuItemSeparatorAfter，这将在该菜单项下方创建一条直线，将第三个命令（Exit）与其他菜单项分开。

最后一个菜单项用于退出游戏。将其名称 iMenuItem3 改为 iMenuItemExit，并将 r_exit 用作其 ID。这次不再使用标准命令 ID，而使用 EAknCmdExit 命令。如果操作系统要关闭应用程序（如手机没有足够内存时），该命令将发送给应用程序。因此，每个应用程序都必须响应该事件，并立刻关闭应用程序。

如果现在运行游戏，将发现 Exit 菜单项已经管用。以后测试游戏时，应总是使用该命令来退出应用程序，而不是仅仅关闭仿真器。这样做的原因是，仿真器环境将自动地检查内存，如果发现内存泄漏将指出这一点。如果仅仅关闭仿真器，将要过好几个小时才能发现内存泄漏，这样查找原因将更困难。

第 6 步：显示 About 对话框

至此，没有编写一行代码就创建了一个包含菜单的应用程序。为显示 About 对话框，需要开始编写代码。切换到 Non-visual 设计视图：

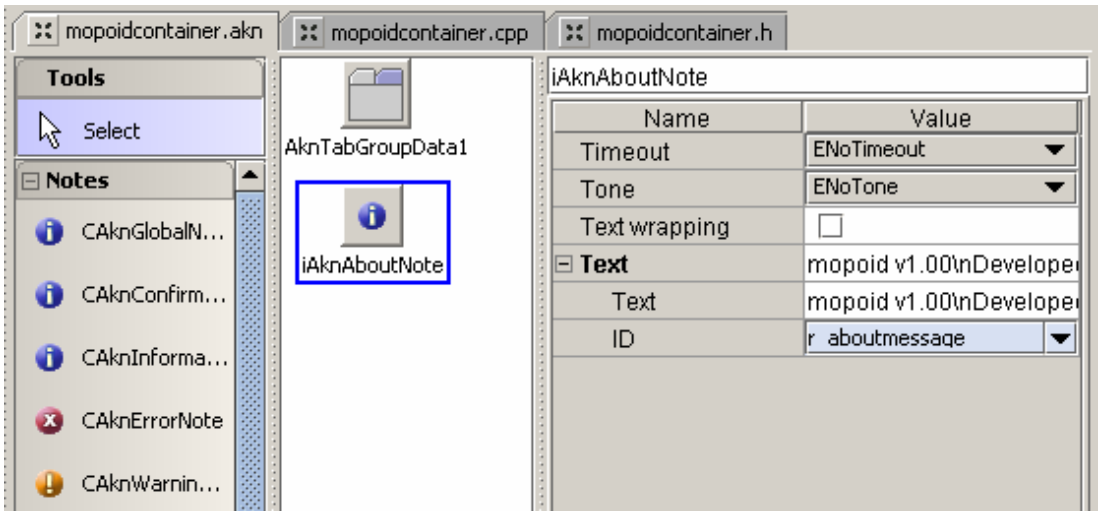


图 10 在应用程序中添加一个 About 对话框

在左边的 Tool 窗格中选择 CAknInformationNote，这是一个简单的小型窗口，将包含指定的文本及一个预定义图标。接下来单击中间的白色区域以创建说明。在右边的属性窗口中，将其名称改为 iAknAboutNote，将其文本 ID 改为 r_aboutmessage。

至此，About 对话框就定义好了。要显示它，必须将其同菜单命令 about 关联起来。切换回 Menu 设计器，选择菜单项 About。

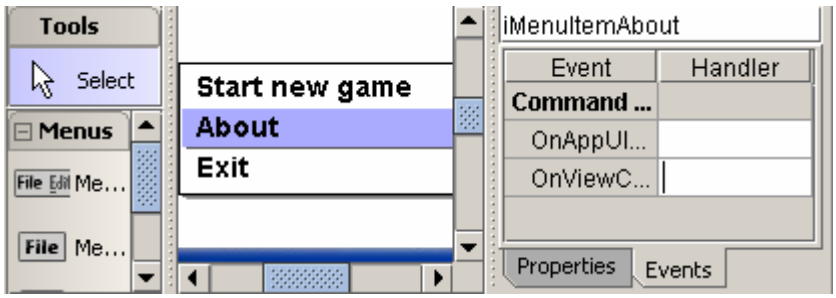


图 11 通过选择菜单项调用 About 对话框

在右边的属性窗口中，切换到 Events。双击 OnViewCommand 旁边的空文本框，C++BuilderX 将自动创建了一个用户选择该菜单项时将调用的函数，并切换到该函数

中。然而，编写代码前，先切换回 MopoidContainer.akn，将 IDE 创建的函数的名称改为 OniMenuItemAboutViewCommandL，这实际上是在函数名末尾添加 L。

稍后将解释这样做的原因。按回车键，C++BuilderX 将切换到该函数中，此时其名称是改正后的。

```
160 TInt CMopoidContainer::ExecuteiAknAboutNoteL()
161 {
162   /* 22.01.05 19:33 */
163   TBuf < 256 > TBuf_256l;
164
165   iCoeEnv->ReadResource( TBuf_256l, RS_R_ABOUTMESSAGE );
166   CAknInformationNote * iAknAboutNote = new( ELeave )CAknInformationNote;
167   iAknAboutNote->SetTimeout( CAknNoteDialog::ENoTimeout );
168   iAknAboutNote->SetTone( CAknNoteDialog::ENoTone );
169   iAknAboutNote->SetTextWrapping( EFalse );
170   return iAknAboutNote->ExecuteLD( TBuf_256l );
171 }
172
173 void CMopoidContainer::OniMenuItemAboutViewCommandL( TInt aCommand )
174 {
175   ExecuteiAknAboutNoteL();
176 }
```

图 12 用于显示 About 对话框的代码

接下来，在 MopoidContainer.cpp 中编写如图所示的第 175 行代码。这行代码调用 C++BuilderX 刚才创建的函数。

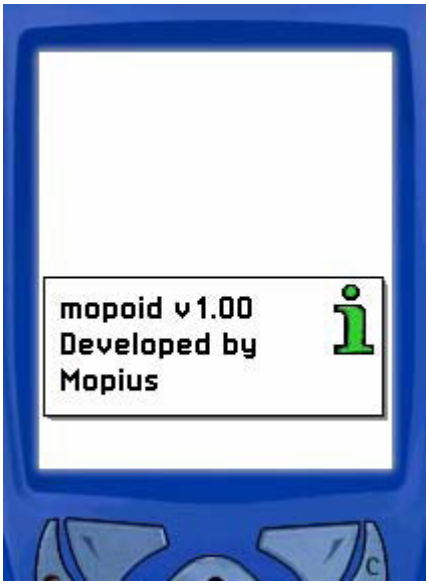


图 13 Series 60 仿真器（SDK v1.2）中的 About 对话框

当开始编程前，About 对话框已经能够正常显示。下面介绍为什么要在函数名末尾添加 L。

读者将注意到，函数 ExecuteiAknAboutNoteL( )的名称末尾也有 L。这是为什么呢？原因是在这个函数中分配了内存（用于创建和显示对话框，另外请注意创建对话框的代码行中的 ELeave），而内存分配过程可能失败。当前，手机的资源很有限。因此，需要处理操作系统不能向应用程序提供所需资源的问题，这很重要。

这些事件及其他错误（如找不到文件）由一个系统专门处理，该系统类似于 Java 或现代 C++中的 try/catch。错误沿调用层次向上传递，直到得到处理。在 Symbian OS 中，为让开发人员能够忽略（leave）错误，在函数名后面添加 L，这已成为一种标准做法。

在这个例子中，如果没有足够的内存，显示 About 消息的函数将忽略这种错误。通常，开发人员不（也不应该）处理这种问题，而让系统显示合适的错误消息。如果开发人员负责处理每种错误，代码将非常庞大。

如果发生错误，错误将自动传递给调用栈中的下一个函数。这意味着OniMenuItemAboutViewCommandL( )函数也可以忽略它。因此，在其名称末尾添加了 L。

到目前为止一切顺利。然而，C++BuilderX 还创建了一个命令分发函数 DispatchViewCommandEvents( )，该函数包含如下注释：NOTE: This routine is managed by the C++BuilderX IDE - DO NOT MODIFY。

注意该函数的名称末尾没有 L。如果手工添加 L，C++BuilderX 将无法识别它。在调用栈中，下一个函数是文件 MopoidView.cpp 中的 HandleCommandL( )（末尾有 L！）。用户选择 About 菜单项时，系统将调用该函数！

由于该 IDE 缺乏灵活性，这里创建了一些不符合 Symbian OS 编码惯例的代码。在这个例子中，将忽略这种情况，因为 L 并非供系统使用，而只是旨在帮助开发人员。然而，这里使用它有助于解释忽略（leave）系统的基础知识以及阐述 C++Builder 自动生成的代码的局限性。

如果出现问题

如果出现问题且需要找出原因，可参阅文件 Mopoid.Step7.zip，它包含至此创建的参考项目。对于接下来的每一步，都有使用参考项目的 zip 文件！

第 7 步：将应用程序安装到手机中

通常仿真器的效果都很不错，但有时候会发生这样的情况：代码在仿真器中工作正常，而在手机上却崩溃了。最突出的例子是静态成员变量，它在仿真器中可行，但在手机中将出现问题！

直接在手机上调试很困难且并非总是可行，当手机只是报告系统错误时很难找出崩溃的原因。然而，如果定期地在手机上运行应用程序，查找问题将更容易。

要让 IDE 知道编译针对的设备，必须将编译过程的目标平台切换为 ARMI（Series 60 手机的处理器为 ARM 处理器），将其类型设置为 UREL（发布版）。要编译项目，选择 Project→Make Project ‘Mopoid.cbx’（Ctrl + F9）。

图 14 修改编译目标

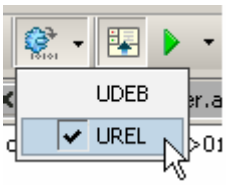


图 15 为发布而不是调试进行编译

完成编译后，目录 C:\Symbian\dev\Mopoid\group 中将包含一个文件 Mopoid_ARMI_UREL.sis。将这个文件传输到手机上，然后安装它。

要发送该文件，可使用手机自带的 PC Suite，也可通过蓝牙或 IrDA 连接（如果有的话）来发送，方法是右击该文件并选择 Send to→Bluetooth device。有关如何在手机上安装文件，请阅读手机的用户手册。

如果一切正常，继续开发时别忘了切换回编译配置 WINS / UDEB！

第 8 步 将游戏引擎加入到项目中

开发整个游戏需要很长时间，本教程旨在介绍 Symbian OS 开发的基础知识，而不是花大量时间创建游戏例程。因此，这里将导入一些预先写好的文件到项目中，然后再添加一些重要而有趣的特性。

请注意：也可以创建具有更优美的面向对象结构的游戏。然而，编写这里提供的文件时，旨在让读者更容易理解与 Symbian OS 相关的问题，而不是提供结构完美的程序。

下面的很多步骤在开发项目时很重要。在开发项目时，很可能需要添加图像和声音文件、添加已有的文件等。

解压缩文件

将文件 Mopoid.Step8.Update.zip 解压缩到目录 C:\Symbian\dev\中。保持压缩文件的目录结构不变！覆盖所有文件。如果没有出现覆盖文件的警告，则说明没有将文件解压缩到正确的目录中。

分发游戏数据

压缩文件中包含一个新目录 data，该目录包含用于游戏的图像和声音。在仿真器中执行应用程序时，程序将在自己的目录中查找文件。它调用.\data\dobitmaps.bat 文件自动创建一个对应的目录，并将所有数据文件复制到该目录。如果 Series 60 SDK 不是安装在默认位置（C:\Symbian）或使用的 SDK 版本不是 v1.2，则首先必须调整该批处理文件中的目录名。

将源文件加入到项目中

接下来要让 IDE 添加一些新的源文件到项目中。为此，右击项目窗口中的 Sources 图标，然后选择 Add sources...。

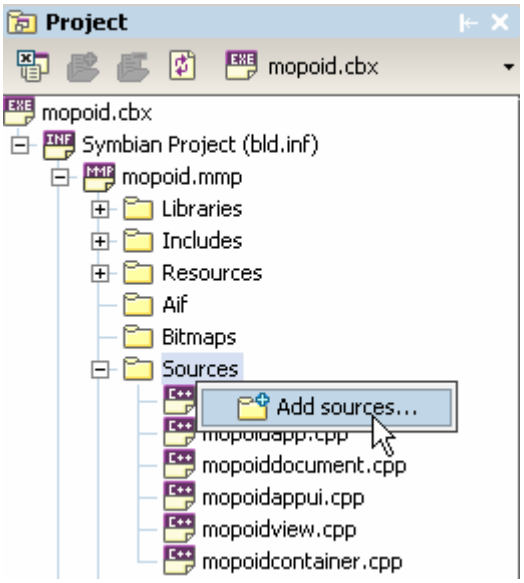


图 16 添加其他源文件

选中./src/目录中还没有包含到项目中的所有文件，将它们添加到项目中：

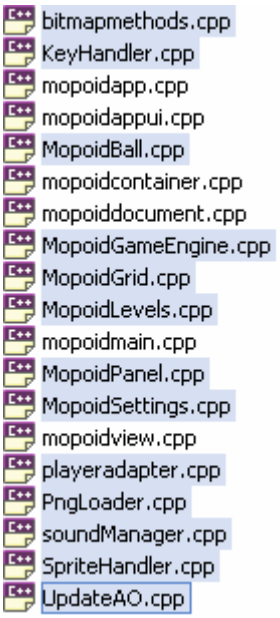


图 17 要添加的文件

将其他库添加到项目定义文件中

最终的游戏要执行很多功能：加载并播放声音、显示 PGN 图像以及访问文件。这些功能要用到 SDK 提供的一些库，开发人员只需将它们添加到项目中即可。在项目定义窗口中双击 Mopoid.mmp，这是 Symbian OS 项目定义文件，它包含对项目所需的所有资源文件及库的引用（及其他内容）。

向下滚动到文件末尾，并在其他所有库定义后面添加下列几行：

```
LIBRARY efsrv.lib            // For loading data files
LIBRARY bitgdi.lib           // For drawing
LIBRARY mediaclientimage.lib // For loading png files
LIBRARY mediaclientaudio.lib // For loading and playing audio
LIBRARY estor.lib            // For writing to & reading from files
```

定义新文本

真正的游戏需要很多文本。按第 4 步介绍的那样在字符串表中定义下列字符串。在以冒号结尾的文本项后面添加一个空格，因为将在文本后面加上数字！

名称	文本
r_score	Score:
r_level	Level:
r_pause	Game Passed
r_gameover	Game Over
r_finished	You made it!
r_lifelost	Life Lost!
r_lives	Lives:
r_pressjoystick	Press Joystick
r_highscore	High Score:
r_enterlevel	Entering Level:
r_title	mopoid

开始游戏

开始新游戏的命令必须与游戏引擎中启动游戏的函数关联起来。像为 About 对话框添加函数一样，为 Start New Game 菜单项创建一个新的事件函数，但这次不要将函数名修改成包含有后缀 L——开始游戏的函数不能忽略错误。将下面的代码插入到新函数 OniMenuItemNewGameViewCommand( )中：

```
iGameEngine->StartNewGame( );
```

添加图像与声音文件

为这个游戏准备了多个.png 文件和.wav 文件，以使其更吸引人。必须将这些文件添加到项目中，这样它们才会被复制到手机中！为此，使用蓝色齿轮将编译配置切换为 ARMI UREL。现在，项目窗格将包含一个 Package File 项。由于 C++BuildX 有一个小错误，添加文件时要格外小心。

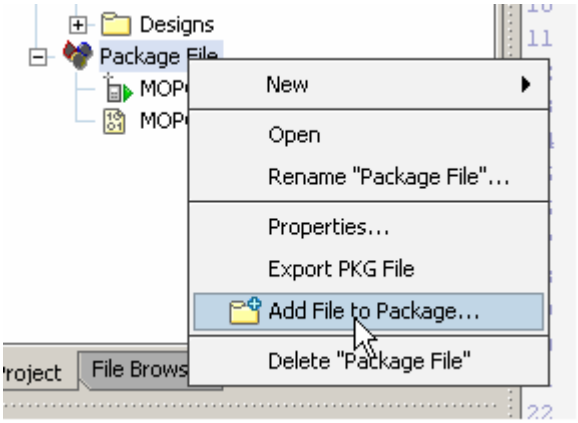


图 18 将其他文件添加到包中

右击 Package File 并选择 Add File to Package...，这将打开一个对话框。选择 Mopoid 项目的/data/目录中的第一个文件（ball.png）。暂还不要单击 OK，否则必须将该文件从包中删除再重新添加。

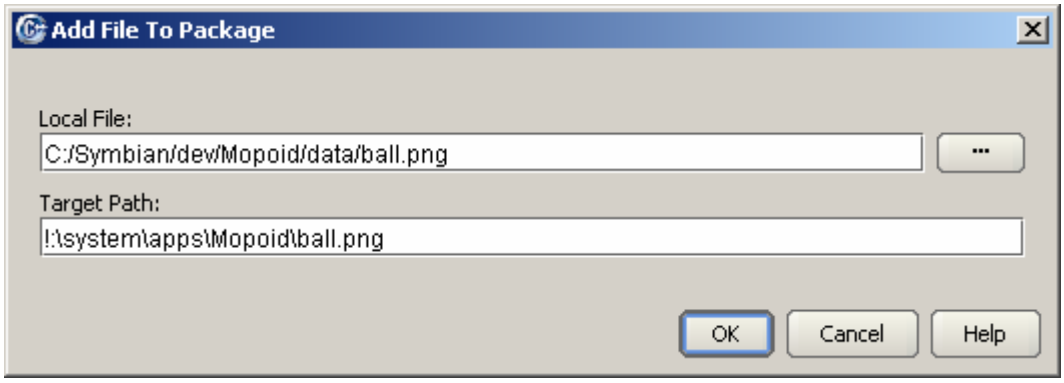


图 19 在目标路径中添加文件名

将文件名（\ball.png）添加到第二个文本框的目标路径中，完成后单击 OK。

目标路径指定了文件在手机中的位置和名称。如果没有指定文件名，手机将无法复制文件！如果 IDE 能够自动地添加文件名就好了。

行首的！表示用户可选择应用程序的安装位置，后面将更详细地介绍这一点。

C++BuilderX 1.5 有个错误：添加文件后，不能再修改目标路径。虽然该 IDE 允许开发人员修改它，但单击 OK 后，它不会保存所做的修改，且不会给出提示。因此，在添加文件时别忘了添加正确的文件名！

对所有.png 文件、.wav 文件和 levels.dat 文件执行上述操作。完成后项目窗格应如下所示：

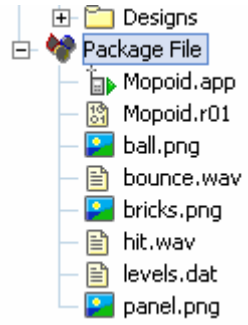


图 20 所有文件都包含在项目中

测试

切换回 WINS / UDEB。现在，游戏应该能够正常运行。然而，屏幕是黑的，其中只有一行文本。接下来几步将添加缺少的所有内容！

排除故障

这里可能出现很多问题，只有严格按照前面的步骤做，项目才能与预先准备的源文件正确组合。如果遇到问题，项目无法运行，可使用 Mopoid.Step8.zip 中的参考项目！

第 9 步：加载图像

Symbian OS C++本身支持.mbm 文件。有一组 RLE——压缩的位图文件。位图文件的优点是加载速度快，容易处理，但它们将占据手机的大量空间。游戏使用大量图像，采用位图制式的游戏占据的空间将很大。

因此，更好的选择是使用.png 或.jpg 文件。但不幸的是，没有预定义的加载例程可用，而手工加载相当复杂。另外，解压缩这样的文件需要一些时间，因此解压缩是异步进行的，在独立的进程中。在 Symbian OS v7.0 中，还有另一种加载图像的方法，但如果希望游戏的用户尽可能多，必须使用旧函数（就现在而言）。

开发与解释加载图像的整个过程有单个教程，因此这里提供了一个开发好的类：PngLoader.cpp。Nokia 提供了另一个文件（bitmapmethods.cpp）用于处理一些常见的任务。还增加了一些函数，使这个类更有用。只要将这两个文件添加到自己的项目中即可！

在 Mopoid 项目中，有一个类负责加载和存储所有图像。如果其他地方需要图像，可将该图像的 ID 传递给 CSpriteHandler 类的一个函数，以获取该图像。

C 类与 T 类

类名开头的 C 表示什么呢？简单地说，这意味着该类总是在堆中创建，且它在大多数情况下还拥有其他（堆）对象。C 类的析构函数被调用时，必须销毁它拥有的对象。

第二重要的类型是 T 类，它基本上类似于简单数据类型，这样的类不能有析构函数。例如，Symbian 中的基本数据类型都有前缀 T（TInt、TReal 等）。

ConstructL

要查看位图管理器类的源代码，可打开 SpriteHandler.cpp 文件。读者将注意到，构造函数为空，需要在 ConstructL( )函数中编写代码，该函数将由 NewL( )函数调用。

这样做的原因在于手机的内存量。开发典型的 PC 应用程序时，大多数情况下都不需要考虑可用内存的问题，但手机完全不同，其内存可能耗尽，导致无法创建类实例。

然而，如果在构造函数中分配内存时失败，已分配的内存将成为孤岛，这很糟糕。因此，在 C++构造函数中执行的代码决不能会忽略错误！然而，sprite 处理程序应在类创建后立即提供图形。

解决方案很简单：使用两阶段构造函数。将所有涉及到内存分配的代码（至少）移到类 ConstructL( )中，这样就可以像通常那样创建对象，然后在下一行中调用该函数。这样，将有机会正确地处理发生的错误。

如果需要多次创建类实例，则必须多次编写代码来实现这些步骤（分配内存和调用 ConstructL( )）。为避免这一点，通常使用静态的 NewL( )或 NewLC( )函数处理两阶段构造函数。一般而言，如果要创建的对象将被赋给成员变量，则必须调用 NewL( )。NewLC( )将对象放在清理栈中，这对自动变量很有用。后面将更详细地讨论这一点，现在先来实现位图的加载！

加载位图

首先，必须指定定义要加载的图像文件的文件名，为此可使用如下代码：

```
_LIT(KBmpPanel, "panel.png");
```

这行代码创建一个命名对象（KBmpPanel），该对象将字符串 panel.png 直接存储到应用程序二进制文件中。在 Symbian OS C++中，字符串的处理与在标准 C++中不同。同样，其原因是移动设备的内存有限。在 Symbian OS C++中，字符串称为描述符，这习惯起来很困难。但这样做有一点好处，那就是描述符占用的内存比标准 C++字符串少。

加载位图后，需要将它存储在某个地方。要查看位图，必须查看 sprite 处理程序类的头文件。不知出于何种原因，C++BuilderX 没有在项目结构中显示这些文件。开发人员必须打开源文件，在左边的文件结构窗格中打开它的 include 文件夹，从中找到 include 文件：

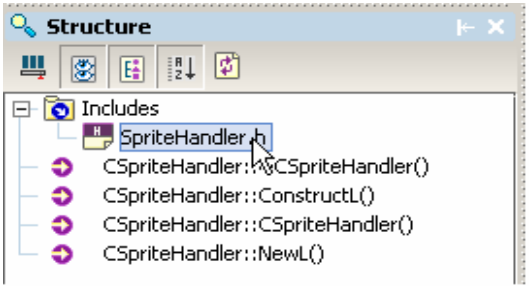


图 21 如何找开 include 文件

在类声明中，包含下述私有成员变量的声明：

```
TFixedArray<CFbsBitmap*, MopoidShared::ENumSprites> iSprites;
```

为什么这个变量命名为 iSprites 而不是 sprites 或 m_sprites？这是另一种 Symbian OS C++编码约定：所有成员（实例）变量都将 i 作为前缀，而参数变量名应以 a 打头。

该变量是什么类型呢？这是一个常规的定长 C++数组，包含范围检查和其他有用的函数（用于遍历数组、获取数组的长度、删除每个元素等）。

回到 Spritehandler.cpp 文件。接下来用位图来填充该数组，这通过调用 CPngLoader 类中的一个静态方法实现的：

```
iSprites[ESpritePanel] = CPngLoader::LoadImageL(KBmpPanel, EColor4K);
```

注意，这里让类将图像加载为 EColor4K。这意味着图像将是 4096 色的，其色深与较早的 Series 60 手机（如 Nokia N-Gage 或 Nokia 7650）相同。较新的手机支持 EColor64K（或更高的色深）。然而，对于简单游戏 Mopoid，其图像不需要这么多颜色。ESpritePanel 是 ID 的文本表示，被称为枚举，是在 MopoidSharedData.h 中定义的。

图 22 面板图像

图 23 面板遮罩（mask）

我们希望面板是圆角的，因此还需要一个遮罩。为此，最简单的方法是使用另一个包含黑白遮罩图像的图形文件。然而，前面说过，加载.png 文件需要很长时间，因此自己生成遮罩是更好的选择。使用下面的方法（该方法也包含本教程提供的源文件中）：

```
iSprites[ESpritePanelM] =

NBitmapMethods::CreateMaskL(iSprites[ESpritePanel], EFalse, 0);
```

这个函数根据源图像左上角像素的颜色创建了一个遮罩位图。如果不希望颜色与左上角像素相同的所有像素都是透明的，可使用第二个和第三个参数来指定要设置为透明的颜色。

接下来，像加载面板图像一样加载球形图像，其文件名为 ball.png，ID 的枚举名为 ESpriteBall 和 ESpriteBallM。

砖块的处理略有不同。它们的形状是矩形，因此不需要遮罩。然而，砖块使用了不同的颜色，因此不是逐个加载图像，而是将所有图像组合为一个文件，加载后再将它们分开。通过使用这种方法，图像文件占用的磁盘空间将更少（整体而言），且加载速度将快得多，虽然应用程序暂时需要更多内存。

图 24 包含 3 个砖块图形的文件

使用下列代码加载砖块图像：

```
_LIT(KBmpBrick, "bricks.png");

CPngLoader::LoadAndSplitImageL(KBmpBrick,

&iSprites[ESpriteBrickNormal], 3, EColor4K);
```

这段代码加载图像文件，将它分为三个图像，并将这些图像存储在 iSprites 数组中，存储位置从 ESpriteBrickNormal 开始。

删除图像

运行应用程序时，程序应加载了图像，虽然您看不到。还没有编写显示图像的代码，现在退出应用程序，将看到如下警告：

图 25 该错误消息指出发生了内存泄漏

这条错误消息指出应用程序发生了内存泄漏。过后将很难找出导致内存泄漏的代码，因此，务必在仿真器中退出应用程序，而不是直接关闭仿真器窗口。

在这个例子中，销毁 CSpriteHandler 对象时没有释放分配给图形文件的内存。幸运的是，由于 iSprites 的 TFixedArray 类，这很容易实现。将下列代码添加到该类的析构函数中即可：

```
iSprites.DeleteAll( );
```

第 10 步：显示图像

仅加载图像文件并没有太大用处——还必须将图像显示在屏幕上。在这个应用程序中，游戏引擎类负责准备后缓冲中的图像，这意味着所有图像都将被绘制到一个屏幕大小的位图中，然后整个位图被复制到屏幕中，从而避免闪烁。

首先来看 CGameEngine 类的 ConstructL( )方法。有一大段计算加载图像大小的代码，解除了注释。如果读者感兴趣，也可看看代码，但就现在而言它们并不重要。

帧本身是在 DrawFrame( )方法中绘制的，该方法位于同一类的源代码后面。注意，该函数被定义为 const，这意味着它不能修改游戏引擎类的成员数据。

处理透明度

请看下面这行代码：

```
iBackBufferBmpGc->SetBrushStyle( CGraphicsContext::ENullBrush );
```

绘制位图时，Symbian OS 通过图像的图形上下文提供了“画笔”和“画刷”，图形上下文负责绘图操作。例如，绘制矩形时，将使用画笔的设置绘制轮廓，并使用画刷进行填充。

这里没有使用这些绘图函数，但必须自己处理这种行为！如果定义一个纯色画刷，并绘制一个包含透明部分的位图，则透明部分将使用当前画刷的颜色及其他属性进行填充！这就是将画刷设置为空的原因，这样位图背景将透过透明部分显示出来。

绘制位图

首先将绘制一个球。第一行代码已经有了，它计算球形的位置，并将结果存储在 TPoint 变量中。这将自动提供 x 坐标与 y 坐标，因此很有用。

在第一行后面编写下列代码：

```
iBackBufferBmpGc->BitBltMasked(ballSpritePos,iSpriteHandler->Get
Sprite(MopoidShared::ESpriteBall),iBall.iSize,iSpriteHandler->
GetSprite(MopoidShared::ESpriteBallM), EFalse);
```

这行代码调用后缓冲位图的图形上下文的 BitBltMasked( )函数，该函数将被遮罩后的源位图复制为目标位图。第一个参数是位置，第二个参数是源图像，第三个参数是要复制的图像大小，第四个参数是要使用的遮罩，第五个参数指出是否将遮罩反相。

要了解 Symbian OS API 提供的函数的参数，可查阅 Symbian OS 帮助。为此，可在“开始”菜单的 Symbian 6.1 SDKs -> Series 60 -> Documentation 中查找 SDK Help。

现在，稍微向下滚动，对面板图像做相同的处理。将其位置设置为 iPanel.iPos 的值；sprite 的枚举 ID 为 ESpritePanel 和 ESpritePanelM；大小为 iPanel.iSize。

前面看到过，砖块不必是透明的。通常，应避免绘制大型透明区域，因为这会降低性能。

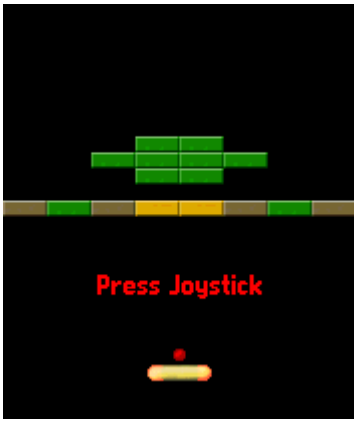


图 26 Mopoid 游戏现在能够显示图像

绘制位图而不使用遮罩的函数（BitBlt( )）与此类似，甚至更容易使用。它只接受两个参数：位置和源位图。将下面这行代码放在 for 循环中标记的地方——for 循环遍历砖块网格以绘制所有可见的砖块。

```
iBackBufferBmpGc->BitBlt(iGrid.ConvertGridToXY(x,y),
iSpriteHandler->GetSprite(spriteId));
```

现在，运行这个应用程序时，它看起来将相当不错。

然而，它还不支持交互。要支持交互，必须处理按键！

第 11 步：处理按键

在 Symbian OS 中处理按键非常简单啊。框架将自动调用容器类（CMopoidContainer）的 OfferKeyEventL( )方法。C++BuilderX IDE 创建的代码将调用转发给 HandleKeyEvents( )函数（在这个例子不需要这么做，因为在该应用程序中没有其他地方需要处理按键）。

HandleKeyEvents( )函数接受两个参数：const TKeyEvent& aKeyEvent 和 TEventCode aType。第一个参数（aKeyEvent）包含有关哪个键被按下的信息；第二个参数（aType）指出发生的是哪种事件。

在 Mopoid 游戏中，我们希望在用户按住方向键时平板不断移动（在这个例子中，用户必须使用游戏杆）。为实现这种行为，必须在按键发生时设置一个标志，在按键松开时解除对该标志的设置。

预先准备了一个特殊类（TKeyHandler），用于存储用户当前按下的方向键。该类由游戏引擎类所有，游戏引擎类还负责将移动事件发送给平板。

开发人员必须完成的任务相当简单，只需编写响应 key down 事件的代码。预先写好的源代码包含其他部分。搜索负责处理 key down 事件（EEventKeyDown）的 if 语句，其中有一个 switch 语句负责处理各个按键，每个键通过扫描码区分。

接下来处理两个另外两种情况：游戏玩家将游戏杆移到左边时调用的 EStdKeyLeftArrow 以及移到右边时调用的 EStdKeyRightArrow。

要将事件发送给前面提到的按键处理程序，必须调用：

```
iGameEngine->iKeyHandler.LeftPressed( );
```

当然，这与将游戏杆移到右边略有不同。如果查看 HandleKeyEvents( )函数，将发现它返回一个布尔值，该返回值告诉系统按键是否得到处理。如果没有处理，按键事件将被发送给下一个可能运行在后台的应用程序（Symbian OS 是一个多任务操作系统！）。

因此，必须让系统知道应用程序已处理了按键，不必将其发送给其他程序。为此，需要相应地设置函数开头定义的标志：

```
handled = ETrue;
```

注意，Symbian OS C++使用自己的布尔值定义，其数据类型为 TBool，取值为 ETrue 或 EFalse。

在 switch 语句中，别忘了在每一个 case 语句末尾添加 break 语句！

编写好代码后，在仿真器中进行测试。这时应该能够移动平板，球能够弹跳，因此可以玩游戏！这一步的代码都是标准 C 代码，但如果读者遇到问题，可查看 Mopoid.Step11.zip 中的 MopoidContainer.cpp 文件。

第 12 步：显示文本

这个游戏还缺少一个重要部分：它不显示有关玩家的分数、生命值和关卡等状态信息。下面来添加这些状态信息，打开 CGameEngine 类的 DrawFrame( )方法。

设置字体

幸运的是，预定义的函数负责显示文本，它们甚至能够将文本对齐。但首先必须命令位图的图形上下文使用哪种字体和颜色，为此在注释// Hud 后面加入下列代码：

```
iBackBufferBmpGc->SetPenStyle(CGraphicsContext::ESolidPen);
iBackBufferBmpGc->SetPenColor(KRgbRed);
iBackBufferBmpGc->UseFont(CEikonEnv::Static(  )->AnnotationFont(
));
TBuf<30> tempStr;
```

下面逐步地分析这段代码。第一行应该很熟悉，它定义画笔样式。前面解释过，画笔用于绘制轮廓。画笔也用于字体的轮廓。在这个例子中，将画笔样式设置为实线，因此我们实际看到的是绘制的文本。

下一行定义颜色，这里使用了一种默认值：纯红色。如果要定义 RGB 颜色，可使用下面的代码：

```
iBackBufferBmpGc->SetPenColor(TRgb(255, 128, 0));
```

下一行选择了一种标准系统字体。Symbian OS 提供了更复杂的字体机制以及创建和使用自定义字体的选项。对于简单的 Mopoid 游戏，我们坚持使用最简单的方法，因此这里使用 AnnotationFont。这种字体使用中等字号，为粗体。

第四行（也是最后一行）创建一个描述符（Symbian OS 中的字符串），其最大长度为 30 个字符。TBuf 类继承了 TDes 类，后者提供了几个操作字符串数据的函数，将使用这些函数来设置文本的内容。注意，TBuf 是在栈上创建的，手机中的栈很有限，因此不要将栈用于存储长度超过 256 个字符的字符串。对于其他情况，应使用基于堆的描述符（HBufC）。

从资源文件中读取文本

接下来读取资源文件中定义的一个字符串（通过 C++BuilderX IDE 的字符串表）。这由 CEikonEnv 类的一个静态方法负责，它直接将字符串写入到（前面定义的）描述符中。该字符串资源名为 RS_R_SCORE，通常它应为 R_SCORE，但 C++BuilderX 在该名称前添加了 RS_。

```
CEikonEnv::Static( )->ReadResource(tempStr, RS_R_SCORE);
```

格式化文本与显示文本

从字符串表中读取的字符串只包含静态文本 Score:，因此必须将当前玩家的得分添加到该字符串的末尾。这由描述符提供的 AppendNum( )函数处理：

```
tempStr.AppendNum(iSettings.iScore);
```

现在，可以将文本显示到屏幕上了。可以使用下面的代码将文本显示在指定的 x/y 处：

```
iBackBufferBmpGc->DrawText(tempStr, TPoint(5, 25));
```

请注意：y 坐标 25 指定文本的下边缘，因此 5/25 是文本左下角的坐标，而不是左上角的坐标！

还需要显示最高得分，读者可自己编写相应的代码。将文本资源 RS_R_HIGHSCORE 读取到同一个临时描述符中，这将覆盖当前存储的内容——这些内容已不再需要了，它已经显示到屏幕中。将通过 iSettings.iHighScore 获得的数字放到文本末尾，然后将文本显示在 5/38 处（当前分数的下一行）。

对齐文本

至此，游戏将在屏幕左边显示当前分数和最高分数。

还需要在右边显示一些信息：

关卡	文本：RS_R_LEVEL
	值：iSettings.iLevel
	位置：y 坐标为 28，x 坐标为

	离屏幕右边缘 3
生命值	文本：RS_R_LIVES
	值：iSettings.iLives
	位置：y 坐标为 38，x 坐标为离屏幕右边缘 3

像前面一样读取文本并设置其格式，但显示方式不同。幸运的是，Symbian OS 提供了一个用于对齐文本的函数：

```
iBackBufferBmpGc->DrawText(tempStr, TRect(0, 0, SCREEN_WIDTH,
SCREEN_HEIGHT), 25, CGraphicsCon 文本：:ERight, 3);
```

TRect(...)定义一个用于对齐文本的矩形。在这里，该矩形为整个屏幕。为优化该程序，可在函数开头定义 Screen TRect，然后在需要 TRect 的所有函数调用中使用该变量。

SCREEN_WIDTH 和 SCREEN_HEIGHT 是我们定义的常量。Series 60 使用的是标准用户界面，因此所有 Series 60 手机的屏幕大小都相同（176x208）。将来，这可能更灵活，Series 60 手机也可能使用更大的屏幕。第三个参数指定 y 坐标，第四个参数指定对齐方式，最后一个参数指定 x 偏移量。

状态消息的处理与此类似。要激活状态消息，可以解除对接下来的代码的注释。如果浏览这段代码，将发现它使用了 TitleFont( )，这种字体比 AnnotationFont 大。它还将状态消息居中。

如果现在运行该游戏，其外观将如下所示：

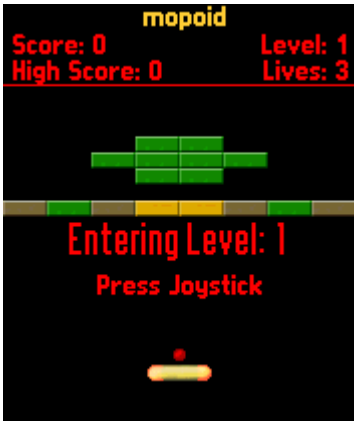


图 27 包含文本的 Mopoid 游戏

很不错，不是吗？为使该游戏看起来真的很酷，还需要做很多工作。

第 13 步：读写文件

本节对读者以后开发项目非常重要。在 Mopoid 游戏中，只保存最高分，以演示如何读写文件。

准备工作

打开 MopoidGameEngine.cpp 并切换到文件 MopoidGameEngine.h。在类声明的前面，声明文件版本号和文件名：

```
#define MOPOID_FILE_VERSION_NUMBER 1
_LIT(KMopoidDataFile, "settings.dat");
```

Symbian OS C++提供了一些用于访问文件的 API。首先，必须包括相应的头文件，在包含其他 include 行之前添加下述代码行：

```
#include <s32file.h>
```

这将包含所需的功能。前面已经将所需的库（estor.lib、efsrv.lib）添加到项目定义（.mmp）中。

设置文件名

在头文件中，只存储了数据文件的文件名——路径必须由应用程序动态添加。如果希望数据文件位于游戏安装目录中，可通过添加应用程序路径来完成这项工作。回到 MopoidGameEngine.cpp 文件并向下滚动到 SaveGameProgressL( )。在这个函数中，首先应执行下列代码：

```
TFileName fullName(KMopoidDataFile);
CompleteWithAppPath(fullName);
#ifdef __WINS__
```

```
fullName[0] = 'C';
#endif
```

如上所示，游戏在仿真器中运行时，必须做一些处理工作。该环境指出应用程序安装在 Z 盘，而 Z 盘为 ROM（在手机中，系统本身就安装在这里）。显然，它将编译到这个（虚拟）盘中。这个盘是不可写的，因此需要必须将盘符改为标准 C 盘。在设备中，可获取应用程序的实际安装路径和盘符。

打开文件

有多种方式可写入文件。这里将使用最灵活的方式，它基于放入存储区（即文件）的流。这使得可以将对象直接序列化到文件中，这在保存复杂游戏的状态时很有帮助。

这里使用直接文件存储区，它不允许修改数据。这对我们来说无关紧要，因为我们将要替换文件的所有内容。下列代码打开文件存储区：

```
CFileStore* store = CDirectFileStore::ReplaceLC(
CEikonEnv::Static( )->FsSession( ), fullName,  EFileWrite);
store->SetTypeL(KDirectFileStoreLayoutUid);
```

注意，这里使用了 CEikonEnv 环境中的文件服务器会话，它负责读写文件。对于整个系统而言，文件服务器只有一个，开发人员不能创建这样的实例，而只能连接它。然而，这是比较耗时的操作，因此这里将复用系统用于读取源文件的文件服务器会话。

创建流

存储区可包含多个流，但其中一个必须是根流。就保存最高分而言，只需要一个流。多个流对于保存各个玩家的数据很有用。创建流时将返回其 ID，以后将该流设置为根流时将用到该 ID。

```
RStoreWriteStream stream;
TStreamId id = stream.CreateLC(*store);
```

写入数据

最后，需要将数据写入文件，将把写入两个整数值写入到流中。在 Symbian OS 中，Tint 是 32 位的变量，因此我们将使用流的 WriteInt32L( )函数来写入数据。

```
// Write file version number
stream.WriteInt32L(MOPOID_FILE_VERSION_NUMBER);
// Write game progress
stream.WriteInt32L(iSettings.iHighScore);
```

并非必须将文件版本写入流中，但事实证明这样做很有用。假设出现如下情况：客户在手机中安装了 Mopoid 游戏；后来您发布了新版本，它也保存玩家的姓名。客户下载它并用它替换手机中的游戏后，旧数据文件将不会删除！

然后，用户启动新的游戏版本，该游戏发现了旧的数据文件，并试图载入它以获取分数和玩家姓名。然而，由于该数据文件是旧游戏的，它不包含玩家姓名。通过比较版本号更容易正确地处理这种情况：例如，可创建一个导入函数。

关闭文件

现在差不多完成了，只要确保写入数据并关闭文件：

```
// Commit the changes to the stream
stream.CommitL( );
CleanupStack::PopAndDestroy( ); // stream

// Set the stream in the store and commit the store
store->SetRootL(id);
store->CommitL( );
CleanupStack::PopAndDestroy( ); // store
```

清理栈

在刚才写的函数中，使用了 Symbian OS 清理（cleanup）栈，这是使用 C++进行 Symbian OS 开发的非常重要的一部分。要全面理解它比较困难，但有很多介绍该主题的文献。因此，这里只简要地总结一下：

想象这种一种情形：方法在堆中创建了一个对象，且有一个指针变量指向它。如果调用该方法时，在结束前出现故障，该方法不能正确返回。框架将自动清理所有自动变量。

然而，如果只有指向对象的指针，指针将被清除，而它指向的对象将留在内存中，没有被任何指针引用。这很糟糕。

因此，通过自动变量使用的对象必须放到清除栈中。如果发生方法非正常结束，框架将自动将清除栈中的所有对象清除。

通常，像下面这样使用清除栈：

```
MyObject x* = new (ELeave) MyObject( );
CleanupStack::PushL(x);
x->DoSomethingDangerousL( );
CleanupStack::PopAndDestroy( );
```

在前面的文件示例中，调用了两个名称以 C 结尾的函数。这表示它们将对象放在清除栈中，不再需要该对象时必须清除它们，因此使用了两个 PopAndDestroy( )。

请注意，虽然框架在发生对象遗留时将清除栈中的对象清除，但为确保程序运行时不发生错误，开发人员也必须正确地进行清除。

加载

保存文件只是问题的一半。幸运的是，加载与保存类似。首先，像前面那样创建完整的文件名；然后打开存储区，但这次旨在能够读取它。

```
CFileStore* store = CDirectFileStore::OpenLC(CEikonEnv::Static( )-
>FsSession( ), fullName, EFileRead);
```

像下面这样打开根流：

```
RStoreReadStream stream;
stream.OpenLC(*store, store->Root( ));
```

然后就可以读取数据：

```
TInt versionNumber = stream.ReadInt32L( );
if (versionNumber ! = MOPOID_FILE_VERSION_NUMBER)
User::Leave(KErrNotFound);
iSettings.iHighScore = stream.ReadInt32L( );
```

可以改进错误版本号的处理方式，但对于游戏的第一个版本来说，这足够了。这里在没有找到文件时结束，这意味着无法在文件中找到正确的设置信息。调用该函数（在游戏引擎的 ConstructL( )中）时将捕获加载函数的异常，并忽略 not found 错误（这种错误在文件没有创建时也将发生）。如果发生找不到文件的错误，游戏将最高分设置为默认值 0。

加载所有数据后，不要忘记清除栈！

```
CleanupStack::PopAndDestroy(2);
```

在卸载应用程序时清除

如果用户决定删除游戏，必须将所有文件从手机中删除（这是应用程序获得 Symbian 签名的要求之一）。卸载由程序管理器完成，且在卸载时不会调用应用程序。因此，必须在指定应用程序该如何安装的地方指定需要删除哪些文件。

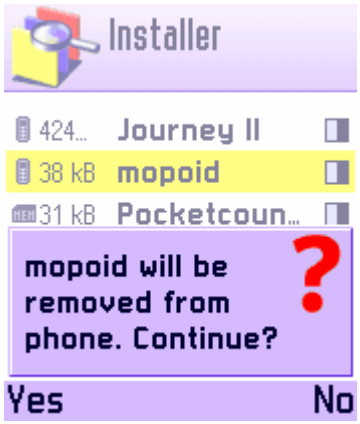


图 28 卸载应用程序时必须删除新的数据文件

不幸的是，在手机安装文件的定义方面，C++BuilderX 的功能有限，且使得这项任务比没有使用 C++BuilderX 时更复杂。

首先，按前面介绍的那样使用蓝色齿轮切换到针对目标设备进行编译（ARMI / UREL）。右击项目窗格中的 Package File 并选择 Export PKG File。这将在目录./dev/Mopoid/group/中创建一个非常小的 Mopoid.pkg 文件。

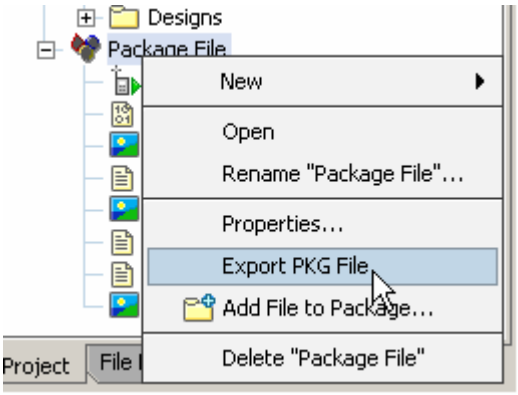


图 29 导出包文件

现在，使用文本编辑器打开这个文件，其内容如下：

```
&EN
#{"Mopoid"},(0x01000001),0,1,0
(0x101F6F88), 0, 0, 0, {"Series60ProductID"}
"C:\Symbian\6.1\Series60\epoc32\release\ARMI\UREL\Mopoid.app"-
"! :\system\apps\Mopoid\Mopoid.app"
"C:\Symbian\6.1\Series60\epoc32\release\ARMI\UREL\Mopoid.r01"-
"! :\system\apps\Mopoid\Mopoid.r01"
"C:\Symbian\dev\Mopoid\data\ball.png"-"! :\system\apps\Mopoid\ba
ll.png"
"C:\Symbian\dev\Mopoid\data\bounce.wav"-"! :\system\apps\Mopoid\
bounce.wav"
"C:\Symbian\dev\Mopoid\data\bricks.png"-"! :\system\apps\Mopoid\
bricks.png"
```

```
"C:\Symbian\dev\dev\Mopoid\data\hit.wav"-"! :\system\apps\Mopoid
\hit.wav"
"C:\Symbian\dev\Mopoid\data\levels.dat"-"! :\system\apps\Mopoid\
levels.dat"
"C:\Symbian\dev\Mopoid\data\panel.png"-"! :\system\apps\Mopoid\p
anel.png"
```

它定义的安装文件只包含一种语言——英语。第二行包含在安装时将显示的应用程序名称、创建项目时设置的 UID 以及版本号。

第三行定义应用程序可在所有 Series60 手机中安装，包括老式 Nokia 7650（它使用 Series 60 SDK v0.9）。如果开发的应用程序不向后兼容，必须使用不同的 Series 60 Product ID。

接下来，这个包文件指定将应用程序和资源文件（包含字符串与菜单定义）复制到设备的文件夹\system\apps\Mopoid\中。开头的！表示用户可以选择将应用程序安装到哪个盘中。Series 60 手机内部的可写内存为 C 盘，而可选的 MultiMediaCard 的盘符为 E。

好了，现在应用程序运行时将创建一个新文件。这个文件不需要安装，但卸载游戏时必须删除。这是如何实现的呢？下面的代码行完成这项工作：

```
""-"! :\system\apps\Mopoid\settings.dat",FN
```

FN 表示 FileNull（空文件），SDK 帮助解释了这行代码的含义：

（空文件指）尚不存在的文件，因此不包含在 sis 文件中。空文件由运行的程序创建，在程序卸载时将被删除。给源文件指定的名称不重要，但应为空（即“”）。注意，这样的文件在程序升级时不会删除。这确保诸如.ini 等存储应用程序设置的文件在升级时不会丢失。

通过命令行进行编译

不幸的是，C++BuilderX 并不关心它导出的包文件。

因此从现在开始，必须通过命令行创建安装文件(.sis)。首先，确保 IDE 的目标编译平台仍设置为 ARMI / UREL (蓝色齿轮)。使用 Project -> Make Project ‘Mopoid.cbx’ 编译项目。

现在，打开一个命令行窗口（选择“开始”->“运行”，然后输入 cmd），并切换到文件夹 C:\Symbian\dev\Mopoid\group。为提高这个过程的速度，可安装 Microsoft 的 Command Window Here PowerToy。输入下面的命令：

```
makesis Mopoid.pkg
```

该命令使用包文件中的信息创建.sis 文件，在同一目录中创建 Mopoid.sis 文件。在手机上试一试这个文件！可以编写一个包含上述命令的小型批处理文件，这样不必每次针对某种设备进行编译时都输入这些命令。

完成后，不要忘了切换回 Windows 仿真器的调试编译设置（WINS / UDEB）！

第 14 步：设置应用程序图标

有些信息（如应用程序图标和标题）存储在一个.AIF 文件（应用程序信息文件）中。这一步将为该项目创建这样的文件。

选择菜单 File->New。选择类别 Mobile C++，选择 New Symbian AIF Wizard 并单击 OK。接受第 1 步的默认值；第 2 步较为有趣，在这里需要添加将显示在手机菜单中的图标。

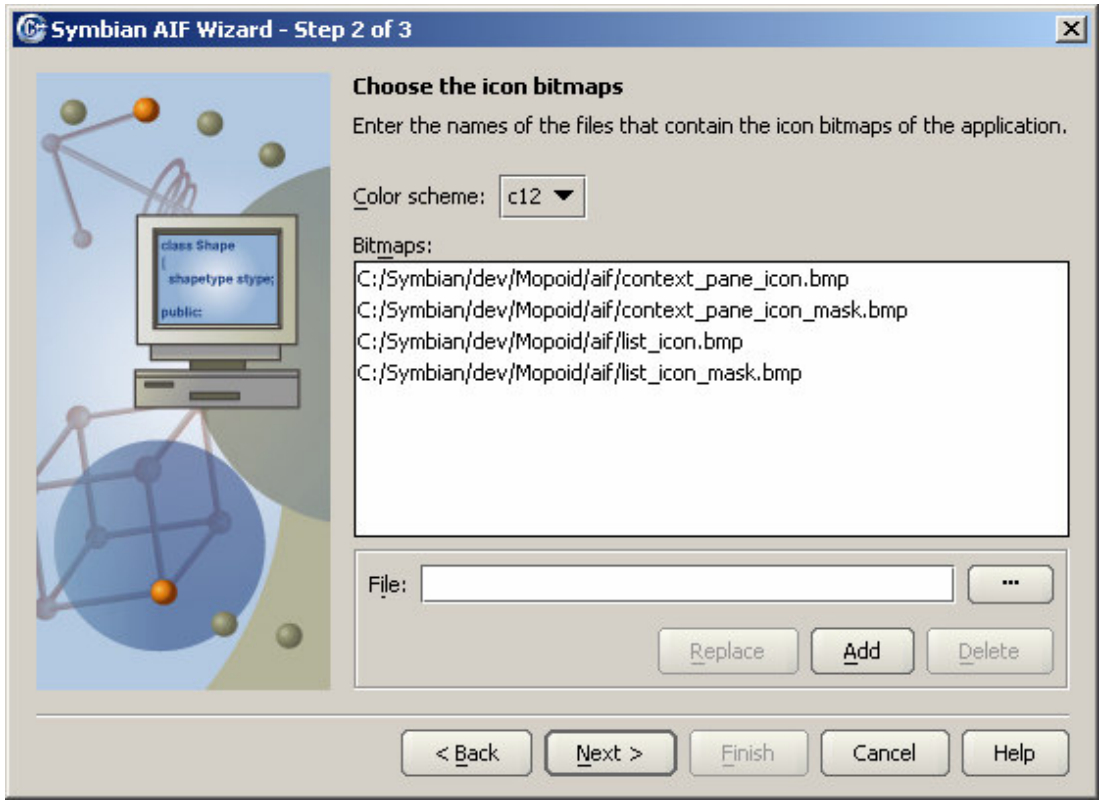


图 30 在游戏中添加图标文件

按如上图所示的顺序添加项目的 aif 目录中的位图——总是先添加图标的位图文件，然后添加其遮罩。选择文件后不要忘记单击 Add 按钮。

添加这四个文件后，进入第 3 步，在这里添加游戏标题。由于 Mopoid 只支持英语，因此输入标题 Mopoid 并保留语言设置 ELangEnglish 不变，然后单击 Add。

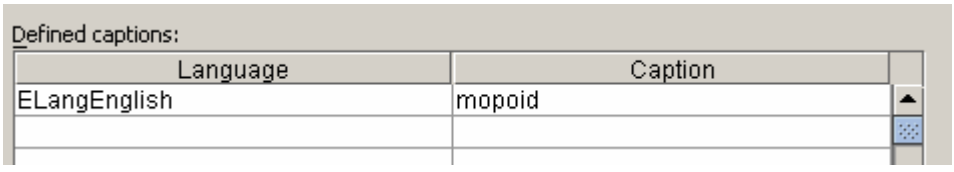


图 31 定义了英语标题

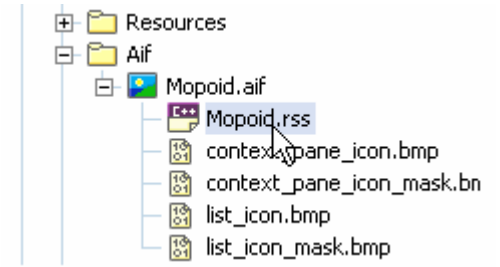


图 32 打开源文件以修复 C++BuilderX 的错误

单击 Finish 后，一切应正常完成。然而，在最新的 C++BuilderX 版本中，存在一个奇怪的错误：如果此时编译游戏，将看不到任何图标。为解决这个问题，双击项目窗格中的 Mopoid.rss（它在 Aif 文件夹中）将其打开。

找到包括 num_icons 的代码行，将其值 4 改为：

```
num_icons = 2;
```

这里只添加了两个不同大小的图标及其遮罩。C++BuilderX 应该知道这些需求，但它显然不知道。

第 15 步：在后台处理

Symbian OS 是一个多任务操作系统。应用程序运行时，很多其他事件可能导致应用程序在后台运行或在应用程序上显示警告，如有来电或短信、低电量警告、用户按红色按钮导致应用程序进入后台等。

图 33 进入后台时游戏应自动停止

即使对 Mopiud 这样的小型游戏来说，处理这种事件也很重要。用户无法玩时，游戏必须暂停。另外，在游戏暂停是应停止屏幕刷新，以尽量减少系统资源的占用。还有，还应保存游戏进度。

在应用程序进入后台时，Symbian OS 将向它发送一个事件，这导致活动视图的函数 HandleForegroundEventL( )被调用。C++BuilderX 在创建代码时没有实现这种功能，下面来实现它。

打开 MopoidView.cpp 并切换到其头文件。注意到 CMopoidView 类是从 CAknView 派生而来的，后者定义了 HandleForegroundEventL( )。要实现该函数，将下列定义添加到公有函数定义的后面：

```
void HandleForegroundEventL(TBool aForeground);
```

参数 aForeground 指出应用程序进入后台还是再次获得关注。切换回 MopoidView.cpp 并将下列函数添加到文件末尾：

```
// Handle any change of focus
void CMopoidView::HandleForegroundEventL(TBool aForeground)
{
 if (aForeground) // gained focus
 {
// Don't resume the game - wait for user to resume it
iContainer->iGameEngine->iHaveFocus = ETrue;
```

```
 }
else // lost focus
{
   // Pause game
if (iContainer)
{
iContainer->iGameEngine->PauseGame( );
iContainer->iGameEngine->iHaveFocus = EFalse;
}
}
  // call base class event handler
CAknView::HandleForegroundEventL(aForeground);
}
```

从中可知，根据应用程序得到还是失去了关注，相应地修改了游戏引擎的一个状态变量。注意，最好不要在应用程序获得关注后立刻恢复游戏，而应让用户准备好后再启动游戏。

这里的实现重写了父类的实现，但在函数末尾还需要调用父类的实现。

第 16 步：周期性事件

游戏本身的更新是通过一个活动对象（Active Object）完成的，与其对应的类在文件 UpdateAO.cpp 中。这意味着游戏的帧频并非固定的，游戏计算两帧之间的时间，以确定球与平板自前一帧以来移动了多长距离。

这意味着在内部计算所有的值时使用的都是浮点数。不幸的是，手机的处理器不支持浮点数，因此必须通过软件来模拟，所以这种计算比整数计算慢得多。

对于很多游戏来说，使用固定帧频（定时器定期回调）足够了。这样，移动距离将为固定整数，不需要进行更精确的计算。

让背景光打开

Mopoid 仍使用固定定时器（实际上这也是 Symbian OS 提供的一个活动对象）。它每隔 5 秒钟将玩家的分数降低，使游戏更具挑战性。另一个问题是，如果一段时间没有按下任何键，手机将关闭背景光。如果此时用户还在玩游戏，只是等待球再次回来，这将很糟糕。因此，需要每隔几秒钟重置手机的用户不活动定时器，这是通过下面的调用实现的，应将其加入到游戏引擎类的定时器回调函数（DoCallBack( )）中：

```
User::ResetInactivityTime( );
```

使用定时器

现在要做的是启动定时器，这样该函数才会被调用。稍微向上滚动，找到游戏引擎类的 StartTimerL( )函数。定时器对象被定义为游戏引擎类的一个私有成员变量：

```
CPeriodic* iPeriodicTimer;
```

因此，在上述函数中，必须创建定时器对象并启动它：

```
iPeriodicTimer = CPeriodic::NewL(CActive::EPriorityLow);

iPeriodicTimer->Start(KTickInterval, KTickInterval,
TCallBack(TimerCallBack, this));
```

注意，这里使用了 Symbian OS 提供的 CPeriodic 对象的静态函数 NewL( )。这是因为 iPeriodicTimer 是一个成员变量，因此该对象没有放到清理栈中。如果发出严重错误，该对象将被游戏引擎的析构函数删除。

接下来的一行代码启动定时器，并要求它每 5 秒钟调用回调函数一次（KTickInterval 设置为 5000000）。最后的参数指定定时器每隔 5 秒种调用的回调函数。该函数名为 TimerCallBack( )，也是游戏引擎类（通过 this 引用）的一部分。这个函数已由示例代码实现了。

停止定时器

游戏暂停或结束时，必须停止（并删除）定时器。这很简单，只需将下列源代码加入到 StopTimer( )函数中。

```
if (iPeriodicTimer)
{
 iPeriodicTimer->Cancel( );
 delete iPeriodicTimer;
 iPeriodicTimer = NULL;
}
```

调用定时器的 Cancel 函数来停止它，然后删除定时器对象并将其设置为 NULL，这样所有人都知道该定时器已不复存在。如果以后需要它，再次创建它即可。

第 17 步：最后的说明

游戏完成了！如果读者愿意，可查看本教程没有介绍的其他源代码。例如，声音播放例程非常有趣。该游戏还使用了一个外部关卡文件，可使用文件编辑器对其进行，从而快速创建自己的关卡。游戏读取并解析该文件。

作者简介

本教程由 Andreas Jakl 撰写。他是 Mopius 的创始人，Mopius 是一家生产创新性手机产品的公司，旨在向用户提供了前所未有的体验。

游戏 The Journey 首批基于位置的手机冒险游戏之一。它以开源方式发布，已被下载 10000 多次。其下一代 The Journey II 拓展了它的概念，创建了一个庞大而迷人的虚拟世界供玩家探索。这两个游戏获得了很多奖项，其中包括 Most Innovative Game（Mobile Fun Awards）和 Austrian State Price 的 jury 奖。该游戏还曾在电视上放映过。

联系方式

若要获得更多信息或与作者联系：

Mopius
Andreas Jakl
Widerinstr. 22
3100 St. Pölten
Austria / Europe
电子邮件：mailto:contact@mopius.com
网站：http://www.mopius.com/
电话：+43 (0)676 / 722 82 78

版权声明

第 18 步：练习

如果读者要自己完成一些任务，下面是一些如何改进该游戏的建议：

其他按键处理方式

有些 Series 60 手机的游戏杆很不完美，用户可能更喜欢使用数字键来控制面板。可添加另一种按键处理方式，让用户能够使用数字键 4（左箭头）与 6（右箭）控制面板。可以像处理游戏杆移动那样使用相同的 swith case 语句来处理这些按键。

提示：使用诸如 case '4'等代码来处理数字键。

定义更多关卡

这个游戏示例只有 5 个关卡——对试验而言这足够了，但可以定义更多关卡。看看 levels.dat 文件是如何定义关卡的、确定文件中的数字定义了哪些类型的砖块以及一个关卡定义包含多少行。然后添加新关卡，并设置游戏使其知道新添加的关卡。修改文件 level.dat 后，别忘了执行 dobitmaps.bat 将 level.dat 文件复制到正确的目录，让仿真器能够找到它！

保存游戏进度

让用户能够继续玩游戏很重要，尤其当游戏很长时。因此，（至少）应让用户能够保存当前关卡以及用户进入该关卡时的分数。应将分数存储在一个新成员变量中。

要保存这些信息，需要在数据文件中添加一个标志，它记录游戏是否处于活动状态、用户进入当前关卡时的分数以及当前关卡。

如果用户离开游戏时游戏还处于活动状态，则将该标志设置为 true，并保存游戏数据。下次用户启动游戏时，应恢复这些值。