برنامه‌نویسی برای سیستم عامل سیمبین

(Symbian OS Workshop)

نویسنده:

Andreas Jakl

مترجم:

موسی مرادی

مقدمه مترجم:

سلام به همه خوانندگان عزیز.

موبایل‌های هوشمند هم پدیده‌ای هستند که به تازگی به وجود آمده‌اند و مانند کامپیوتر به سرعت رشد می‌کنند و فراگیر می‌شوند. به موازات این وضعیت برنامه‌نویسی برای آنها نیز روز به روز پیشرفت می‌کند.

در کشور ما برنامه‌نویسی برای موبایل هنوز چیز خیلی جدیدی است و شاید تعداد برنامه‌های نوشته شده ایرانی تاکنون (البته تاکنونی که من این مطلب را می‌نویسم یعنی آبان ماه سال ۱۳۸۴) از تعداد انگشتان دست هم تجاوز نکند. کشور ما دارای استعدادهای بالقوه و پتانسیل‌های زیادی است، مخصوصاً جوانان مملکت ما که واقعاً تیزهوش هستند و با وجود این همه خلأ آموزش در کشور، به خوبی پیشرفت می‌کنند. با توجه به اینکه به تازگی بعضی از جوانان ما هم به طرف برنامه‌نویسی برای موبایل می‌روند و نیز وجود خلأ آموزشی بزرگ در کشور، بنده تصمیم گرفتم که سهم کوچکی در پر کردن این خلأ ایفا کنم و به همین دلیل این کتابچه را تهیه کردم.

این کتابچه، ترجمه یک کتاب الکترونیکی انگلیسی با نام Symbian OS Workshop است که بنده از سایت http://www.symbian.com/developr دریافت کرده‌ام. این کتاب در حقیقت یک تیوتوریال (Tutorial) است که شما را با مفاهیم پایه‌ای برنامه‌نویسی برای سیستم عامل سیمبین آشنا می‌کند. امید است بتواند کمک کوچکی به شما بکند.

ضمناً این نکته را نیز باید یادآور شوم که برای استفاده از این کتابچه حداقل باید آشنایی متوسطی با زبان C++ داشته باشید. در غیر این صورت درک مطالب برایتان سخت خواهد بود. البته C++ پایه و اساس تمام زبان‌های برنامه‌نویسی است و من به شما توصیه می‌کنم که حتماً آن را فراگیرید و حداقل با مفاهیم تابع و کلاس و شیء و اشاره‌گر و کلاس حافظه و ... آشنا شوید.

البته دقت داشته باشید که این کتاب هم فقط یک نقطه شروع برای شما است و با مطالعه این کتاب، فقط با مفاهیم پایه آشنا می‌شوید. پس حتماً بعد از اتمام این کتاب از منابع دیگری هم استفاده کنید و قدرت برنامه‌نویسی خودتان را بالا ببرید.

و سخن آخر اینکه بنده مترجم حرفه‌ای نیستم، پس اشکالات ترجمه‌ای و نحوی و ... را به بزرگی خودتان ببخشید و سخت نگیرید! بالاخره هدف من از ترجمه این کتاب فقط انجام کمکی هرچند ناچیز به علاقمندان بود وگرنه بنده که هیچ سود مادی از ترجمه این کتاب به دست نمی‌آورم.

البته در صورت تمایل به خواندن کتاب اصلی به زبان انگلیسی می‌توانید آن را از آدرسی که در بالا ذکر شد، دریافت کنید.

در صورت داشتن هرگونه پیشنهاد، انتقاد و ... ، می‌توانید با ایمیل من تماس بگیرید و یا به وبلاگ مراجعه کنید.

به امید روزی که ایرانیان همچون گذشته، پرچمدار علم و فرهنگ جهان شوند...

با تشکر

موسی مرادی – ۱۸ ساله

شهرستان مرند – استان آذربایجان شرقی

mousamk@gmail.com

http://symbiandeveloper.blogfa.com

بسمه تعالی

مقدمه نویسنده:

در این خودآموز، شما چندتا از مفاهیم اساسی برنامه‌نویسی برای موبایل توسط C++ را یاد خواهید گرفت. این خودآموز فقط برنامه‌نویسی برای سیستم عامل سیمبین سری ۶۰ را آموزش می‌دهد.

در ابتدا شما یاد خواهید گرفت که چگونه یک پروژه برای موبایلتان ایجاد کنید. سپس شما کدهای یک بازی از پیش نوشته شده به نام Mopoid را به پروژه‌تان اضافه خواهید کرد. این کار، یک روش لذتبخش برای یاد گرفتن جنبه‌های مهم برنامه‌نویسی سیمبین است که شامل موارد زیر می‌شود:

- تعریف کردن و استفاده کردن از منوها
- پردازش و نمایش متن
- لود کردن و نمایش دادن تصاویر
- نوشتن اطلاعات به فایل‌ها و خواندن اطلاعات از فایل‌ها.
- استفاده کردن از یک تایمر برای رویدادهای مکرر
- ... و خیلی موضوعات کوچک ولی مهم دیگر.

شما نتایج کارهایتان را همزمان مشاهده خواهید کرد. هر موقع که یکی از مختصه‌های سیستم عامل سیمبین دیده شود (مثلاً کار کردن با حافظه)، بعد از آن یک توضیح کوتاهی خواهیم داد که چرا این گونه شد و این قسمت چطور کار می‌کند.

البته مطمئناً فقط مقدار کمی توضیح داده خواهد شد، چرا که توضیح هر کدام از آنها، خود یک کتاب می‌طلبد، در حالی که این خودآموز، فقط قسمت‌های مهم برنامه‌نویسی با استفاده از C++ به شما می‌دهد، و یک نقطه شروع برای پروژه‌های شخصی خودتان است.

من امیدوارم که شما از استفاده کردن از این خودآموز لذت ببرید و در آخر هم یک بازی خوب بسازید.

فهرست عناوین

گام ۰ : آمادگی

مؤلفه‌های زیر را دانلود کنید:

- **ActivePerl**: برای کامپایل کردن پروژه‌تان ضروری است.

http://www.activestate.com/

- **SDK برای سیستم عامل سیمبین**: این خودآموز از SDK 1.2 سری ۶۰ برای سیستم عامل سیمبین، ویرایش نوکیا، استفاده می‌کند تا بهترین سازگاری را با دستگاه‌های سری ۶۰ قبلی داشته باشد.

http://www.symbian.com/developer/sdks_series60.asp

- **Microsoft Debugging Tools**: برای اشکالزدایی برنامه‌ها در شبیه‌ساز در نرم‌افزار C++ BuilderX ضروری است.

http://www.microsoft.com/whdc/devtools/debugging/

- **Visual C++ Toolkit**: فقط در صورتی لازم است که شما ویژوال استودیو ۶ یا net. نداشته باشید. این نرم‌افزار رایگان است و برای کامپایل کردن پروژه‌ها در شبیه‌ساز به کار می‌رود. به آدرس زیر بروید و در آنجا عبارت Visual C++ Toolkit را برای دانلود کردن جستجو کنید.

http://www.microsoft.com/downloads/

- **Borland C++ BuilderX Mobile Edition (v1.5)**: فرم‌ها را پر کنید و این نرم‌افزار را به طور رایگان دانلود کنید.

http://info.borland.com/survey/cbx15_mobile_edition.html

این نرم‌افزارها را به ترتیبی که در بالا فهرست شده است نصب کنید. توصیه می‌شود که SDK سری ۶۰ را در مسیر پیش‌فرض (C:\Symbian\) نصب کنید.

شروع

ممکن است برنامه شما در شبیه‌ساز هنگ کند، می‌توان کاری کرد که به جای هنگ کردن، یک پیغام اخطار مناسبی نمایش دهد. برای فعال کردن این ویژگی یک فایل خالی با نام ErrRd در پوشه C:\Symbian\6.1\Series60\Epoc32\Wins\c\system\Bootdata ایجاد کنید. به همین راحتی!

هنگامی که برای اولین بار C++ BuilderX را اجرا می‌کنید، برنامه نیاز به رجیستر شدن دارد. مورد Activation file را انتخاب کنید و مسیر فایلی را که بورلند هنگام دانلود نرم‌افزار برای شما فرستاده بود را مشخص کنید.

انتخاب یک IDE (محیط برنامه‌نویسی):

ابزارهای ساخت سیمبین در حقیقت دستوری (Command line) هستند و بدون هیچ گونه محیط برنامه‌نویسی کار می‌کنند. ولی به هر حال اگر شما یک IDE داشته باشید، برنامه‌نویسی خیلی راحت‌تر خواهد شد. انتخاب‌های زیادی وجود دارد که هر کدام مزیت‌ها و نقص‌هایی دارند:

Microsoft Visual C++ 6	سریع، کامپایلر و دیباگر کارآمد، بعضی ابزارها برای سیستم عامل سیمبین قابل دسترسی است.
	قدیمی؛ پشتیبانی حقیقی از سیستم عامل سیمبین ندارد.
Microsoft Visual Studio .net	سریع، مدرن، خوب، گسترده (پرکاربرد)
	به طور مستقیم از SDKهای سیمبین پشتیبانی نمی‌کند؛ برنامه‌نویسی برای سیستم عامل سیمبین توسط آن پیچیده است؛ گران است
Metrowerks CodeWarrior	حداقل از سیستم عامل سیمبین پشتیبانی می‌کند.
	این محیط برای مبتدیان سخت و پیچیده است
Borland C++ BuilderX Mobile Edition	طراحی تصویری منوها و پنجره‌های محاوره را پشتیبانی می‌کند؛ اکنون رایگان است
	یک محیط کند و بدترکیب است؛ پشتیبانی از سیستم عامل سیمبین می‌توانست بیشتر از این باشد.
Eclipse	یک محیط بسیار خوب؛ رایگان
	از دیباگینگ (اشکالزدایی) پشتیبانی نمی‌کند؛ در حال حاضر از سیستم عامل سیمبین پشتیبانی نمی‌کند

همانطور که می‌بینید، هیچ محیط «بهترین»ی برای برنامه‌نویسی سیستم عامل سیمبین با زبان C++ وجود ندارد. برای این خودآموز، Borland C++ BuilderX انتخاب شده است؛ زیرا رایگان است و دارای پیچیدگی‌های زیادی برای مبتدیان نیست. اگرچه این IDE هنوز هم به بهبود یافتن نیاز

دارد.

لطفاً توجه کنید: فهرست بالا براساس عقاید شخصی نویسنده نوشته شده است؛ CodeWarrior می‌توانست یک انتخاب خوب باشد، ولی اکنون ویرایش موبایل آن نرم‌افزار توسط نوکیا خریده شده است و در آینده قویتر خواهد شد.

انتخاب SDK سری ۶۰

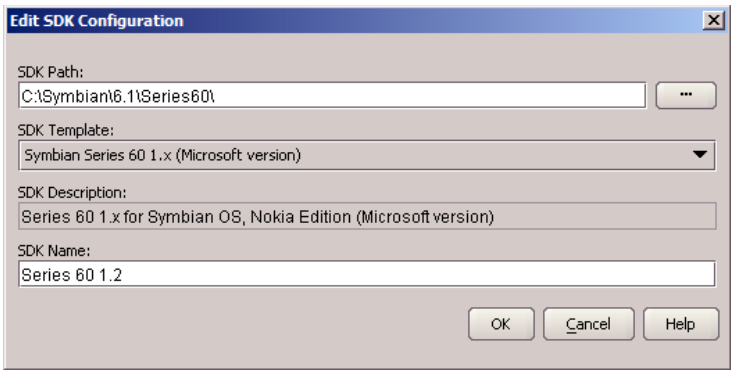
SDK سری ۶۰ برای سیستم عامل سیمبین اکنون در ویرایش‌های بالاتر که گوشی‌های سری ۶۰ جدیدتر را هدف گرفته‌اند نیز قابل دسترسی است. تاکنون تمام دستگاه‌ها، مدل‌های قبلی را پشتیبانی می‌کنند، بنابراین برنامه‌ای که برای سیستم عامل سیمبین 6.x نوشته شده است (توسط SDK v1.2) در سیمبین‌های جدید 7.0 و 8.0 نیز کار می‌کند.

درست است که برنامه‌های کامپایل شده توسط SDK v2.0 (که در حقیقت گوشی Nokia 6600 با سیمبین v7.0 را هدف گرفته است) در مدل‌های قدیمی کار می‌کنند، ولی این مسئله می‌تواند مقداری مشکل‌ساز باشد. اگر سورس‌کد توسط SDKهای جدید کامپایل شود، فایل‌های به حالت باینری به هیچ‌وجه در مدل‌های قدیمی کار نخواهد کرد.

این که از کدام SDK استفاده کنید، به انتخاب خودتان است. اگر شما می‌خواهید که فقط برای گوشی‌های جدید برنامه‌نویسی کنید، و یا اگر می‌خواهید کارهای جدیدی انجام دهید که قبلاً ممکن نبود، از SDK v2.0 (حداقل) استفاده کنید. لود کردن تصاویر و صداها در این ویرایش بهبود یافته است. کار کردن با ارتباطات اینترنتی (HTTP) خیلی راحتتر شده است.

اگر می‌خواهید که یک برنامه تجارتی بنویسید که برای گروه زیادی از مردم قابل استفاده باشد، نباید گوشی‌های سری ۶۰ قدیمی را نادیده بگیرید. شما با انتخاب سری ۶۰ پتانسیل بازار خودتان را محدود می‌کنید، حال اگر برنامه شما فقط گوشی‌های جدید را پشتیبانی کند، شما بازارتان را بسیار محدودتر خواهید کرد و تجارت خوبی نخواهید داشت.

گام ۱ : تعریف SDK سیستم عامل سیمبین

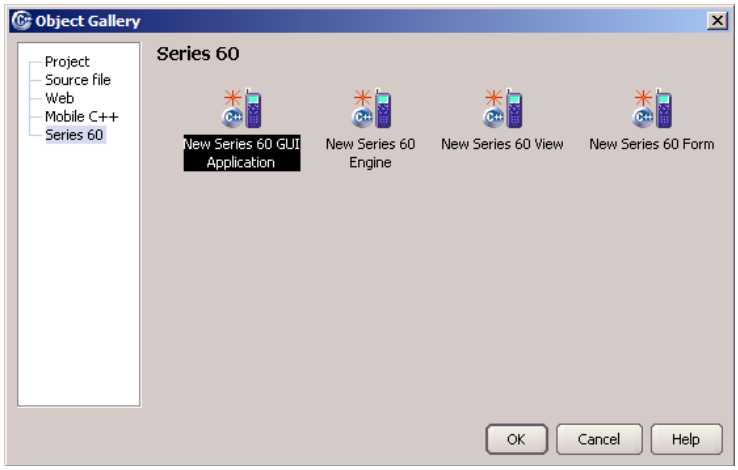


تصویر۱: تعریف SDK سیستم عامل سیمبین در Borland C++ BuilderX

الان موقع آن است که کارمان را شروع کنیم. C++ BuilderX را اجرا کنید. به مسیر Tools → Symbian SDK configuration بروید. یک پیکربندی SDK (Configuration) جدید توسط الگوی SDKی که نصب کرده‌اید، ایجاد کنید. در این مورد: Symbian Series 60 1.x (Microsoft Version). برای SDK v1.2 از ویرایش Borland SDK استفاده نکنید، وگرنه برنامه کامپایل نخواهد شد. مسیر را مانند مسیر نشان داده شده در عکس تنظیم کنید و یک نام مناسب مانند Series 60 1.2 برگزینید.

گام ۲ : ایجاد یک پروژه جدید

یک پروژه جدید از مسیر File → New… → Series 60 → Series 60 GUI Application ایجاد کنید.



تصویر۲: ایجاد یک پروژه Symbian C++ جدید

نام پروژه را Mopoid قرار دهید. (از نام دیگری برای این پروژه استفاده نکنید، زیرا در غیر این صورت، بعضی از فایل‌های از قبل نوشته شده که بعداً ضمیمه خواهیم کرد، کار نخواهند کرد) آن را در مسیر C:\Symbian\dev قرار دهید و به C++ BuilderX اجازه دهید یک زیرپوشه برای این پروژه بسازد؛ که در این صورت پروژه‌های شما جدا از هم نگهداری خواهند شد.

به مرحله بعدی بروید. همان نام را برای پروژه وارد کنید (Mopoid). به علت اینکه ما می‌خواهیم یک بازی بنویسیم، مورد Full Screen را به عنوان view type انتخاب کنید.

UID3 که به طور پیش‌فرض توسط C++ BuilderX پیشنهاد می‌شود، نباید استفاده شود. UID را به چیزی بین 0x01000000 و 0x0FFFFFFF (IDهای رزرو شده برای تست کردن پروژه‌ها) تغییر دهید. توضیح: هر برنامه نصب شده در یک گوشی، باید یک UID منحصر به فرد

داشته باشد. بنابراین اگر می‌خواهید که بازی‌تان را برای عموم منتشر کنید، یک ایمیل به آدرس uid@symbiandevnet.com بفرستید که باید حاوی نام شما و تعداد UIDهای مورد نیازتان باشد. (برای شروع، ۵ تا UID کافی است.) آنها UIDها را در اولین زمان ممکن به شما خواهند فرستاد.

تصویر۳: تعریف کردن ویژگی‌های پروژه (مرحله اول)

(توضیح مترجم: این روش قبلاً استفاده می‌شد. اکنون برای انجام این کار باید در سایت http://www.symbiansigned.com ثبت نام کنید و بعد از Log in شدن، در قسمت چپ، به request UIDs بروید و تعداد UIDهای مورد نیاز را بنویسید. به شما یک بازه داده می‌شود که تمام اعداد داخل بازه، مخصوص شما هستند. ضمناً در این سایت می‌توانید اطلاعات مختلفی را در مورد UID و انواع آن کسب کنید.)

تصویر۴: تعریف کردن ویژگی‌های پروژه (مرحله دوم)

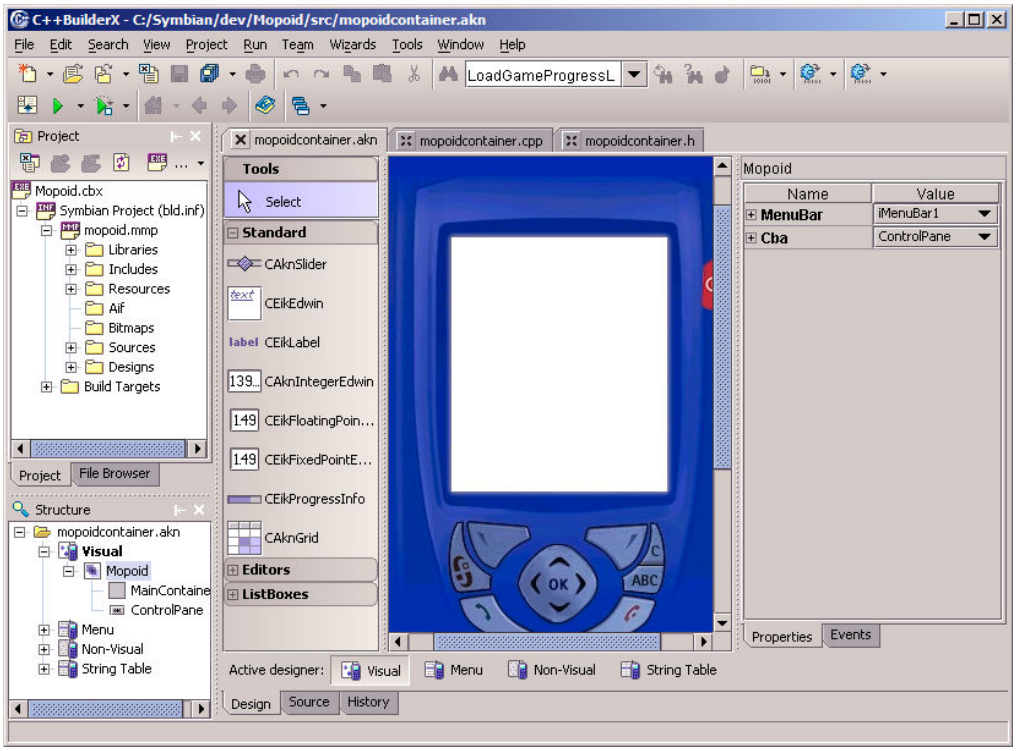
گام ۳ : تست کردن پروژه

محیط کار شما الان باید شبیه تصویر ۵ باشد. اکنون وقت آن است که پروژه را امتحان کنید تا ببینید که آیا همه تنظیمات درست وارد شده‌اند یا نه. دکمه Run Project را فشار دهید. (و یا کلید F9)

اگر همه چیز به درستی کار کند، شبیه‌ساز بعد از مدت کوتاهی نشان داده خواهد شد. اگر مایلید سرعت مقداری بیشتر شود، جستجوگر ویروس

مقیم حافظه را غیرفعال کنید. برنامه شخصی شما در پایین Menu قرار داده خواهد شد، پس برای انتخاب کردن آن، توسط کلیدهای جابجایی به پایین حرکت کنید.

اگر نمی‌خواهید که همیشه هنگام تست کردن، به پایین Menu بروید، آن را انتخاب کنید، کلید سمت چپ (Options) را بزنید، Move را



تصویر۵:محیط کار پیش‌فرض Borland C++ BuilderX شبیه این تصویر است

انتخاب کنید و آن را به بالای Menu انتقال دهید. اگر شما شبیه‌ساز را ببندید و دوباره باز کنید، باهم در موقعیت تنظیم شده باقی خواهد ماند.

هنگامی که شما برنامه Mopoid را اجرا می‌کنید، شما فقط یک صفحه سفید خواهید دید. به این علت که ما انتخاب کردیم که یک برنامه تمام صفحه ایجاد کنیم، که هنوز هیچ وضعیت مرئی در نوار منو ندارد. هنگامی که کلید چپ شبیه‌ساز را می‌زنید، یک منوی پیش‌فرض نشان داده خواهد شد. در مراحل بعدی این منو را طبق نیاز خودمان تنظیم خواهیم کرد.

کشف و رفع اشکال:

اگر Visual Studio .net یا Microsoft Toolkit در رایانه شما نصب شده است و C++BuilderX از آن کامپایلرها برای SDK v1.2 سری ۶۰ استفاده می‌کند، ممکن است error زیر را دریافت کنید.

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

برای حل این مشکل، فایل C:\Symbian\6.1\Shared\EPOC32\Tools\cl_win.pm را در یک ویرایشگر متن (مثلاً Notepad) باز کنید و به دنبال خطی بگردید که حاوی /W4 است و سپس آن را به CLFLAGS = /nologo /Zp4 /W4 /QIfist تغییر دهید. (شما پارامتر /QIfist را به گزینه‌ها افزودید.)

اگر C++ BuilderX، هنگام ساخت errorهایی شبیه زیر ایجاد کند:

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

یک راه حل ممکن این است که دو مسیر زیر را در ابتدای مسیر متغیر محیط قرار دهید.

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

اگر این روش کمکی نکرد، مطمئن شوید که Perl نصب شده است و SDK سیستم عامل سیمبین را نصب تعمیری (repair install) کنید.

مطمئن شوید که SDK بعد از Perl بر روی سیستم نصب شده است.

حتماً بعد از پایان تست کردن پروژه‌تان پنجره شبیه‌ساز را ببندید! اگر شما بخواهید پروژه را کامپایل کنید در حالی که شبیه‌ساز هنوز در پس‌زمینه در حال اجرا است، errorهای مختلفی دریافت خواهید کرد.

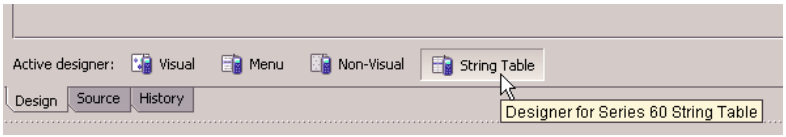
گام ۴ : تعریف کردن رشته‌ها (string)

به طور پیش‌فرض فایل MopoidContainer.akn باید فعال باشد. اگر نیست آن را از طریق پنجره پروژه در سمت چپ اجرا کنید.

Mopoid.cbx → Symbian project (bld.inf) → Mopoid.mmp → Designs → MopoidContainer.akn (از طریق دابل کلیک).

برای طراحی منوهای خودمان، اول باید متنی که می‌خواهیم استفاده کنیم را تعریف کنیم. در سیستم عامل سیمبین، متن‌ها معمولاً در فایل‌های منبع (resource) تعریف می‌شوند. این روش جمع و جور کردن برنامه‌ها را آسانتر می‌کند، به علت اینکه برنامه‌های موبایل جهانی هستند، شاید بخواهید که بازی شما در بیش از یک زبان قابل دسترسی باشد. اگر از فایل‌های منبع استفاده نکنید، در میان انبوه متون چندزبانه و گوناگون سردرگم خواهید شد.

به طراح String Table بروید:



تصویر۶: رفتن به طراح جدول متن (String Table)

در این بازی ما می‌خواهیم یک منوی ساده داشته باشیم که به ما اجازه دهد که یک بازی جدید را شروع کنیم، یک پنجره محاوره «درباره برنامه» را نشان دهیم و از بازی خارج شویم. بنابراین چهار رشته زیر را تعریف کنید:

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

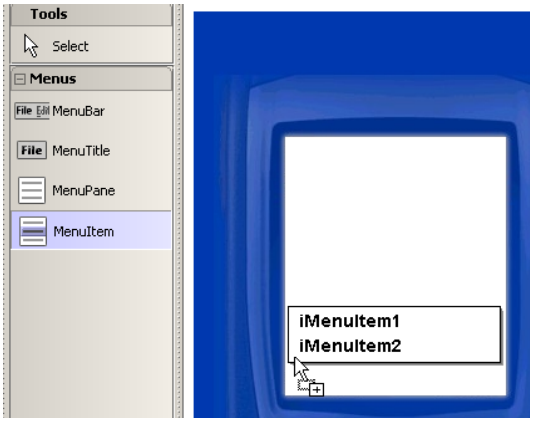
تصویر۷: اضافه کردن رشته‌های جدید به فایل منبع از برنامه

برای پیغام در پنجره محاوره (r_aboutmessage)، شما ابتدا مجبورید که Max Length را به چیزی مثلاً ۶۰ تغییر دهید، بنابراین شما فضای کافی برای پیامتان خواهید داشت. برای ایجاد سطر جدید، می‌توانید از عبارت \n استفاده کنید. همیشه بعد از اتمام نوشتن متن در سلول، Enter را بزنید تا مطمئن شوید که C++BuilderX متن جدید شما را نگهداری می‌کند.

لطفاً توجه کنید: برای این خودآموز، اسامی را دقیقاً همانطور که شرح داده شد، وارد کنید. در یکی از مراحل بعدی، شما چند سورس‌فایل از قبل نوشته شده را به پروژه‌تان اضافه خواهید کرد که تعاریف نام مختلفی خواهند داشت.

گام ۵ : تعریف کردن منوها

اکنون وقت آن است که از رشته‌هایی که تعریف کردیم، در بازی استفاده کنیم. به Menu Designer بروید. خواهید دید که C++BuilderX از قبل دو منو برای شما ایجاد کرده است. ما یک منوی دیگر هم نیاز داریم، پس از پانل سمت چپ ابزار Menu Item را انتخاب کنید و روی منو به منظور اضافه کردن یک آیتم جدید، کلیک کنید.



تصویر۸: اضافه کردن یک منوی جدید توسط ابزار طراحی

حال منوهای جدید را طوری تنظیم خواهیم کرد که از متنی که ما تعریف کردیم، استفاده کنند.

روی iMenuItem1 کلیک کنید تا انتخاب شود. در پانل سمت راست، نام آن را به iMenuItemNewGame تغییر دهید. برای متنش، فقط عبارت r_startgame را به عنوان ID انتخاب کنید. Command را با مقدار پیش‌فرض (۱۰۰۱) رها کنید.

لطفاً توجه کنید: زدن Enter پس از تغییر یک مقدار ضروری است. اگر شما فقط توسط ماوس یا کلیدهای مکان‌نما به جای دیگری بروید، ممکن است ورودی شما ذخیره نشود.

iMenuItemNewGame

Name	Value
− **Text**	Start new game
Text	Start new game
ID	r_startgame
+ Flags	False,False
Command	1001
Cascade	(null)

تصویر۹: تنظیم کردن ویژگی‌های یک منو

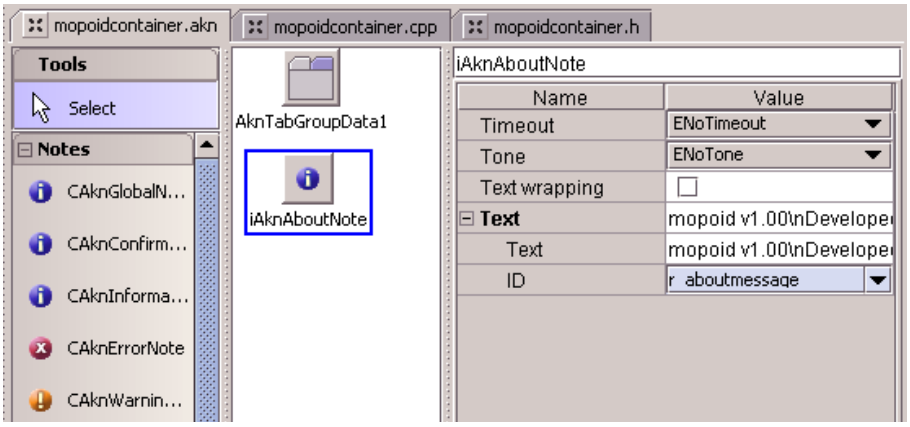
سپس نام iMenuItem2 را به iMenuItemAbout تغییر دهید. کلمه r_about را به عنوان Text-ID انتخاب کنید. مقدار ۱۰۰۲ برای Command مناسب است. همچنین فلگ (flag) دوم یعنی EeikMenuItemSeparatorAfter را نیز علامت بزنید. این فلگ، یک خط کوچک در زیر منو ایجاد خواهد کرد که به صورت تصویری دستور سوم (Exit) را از بقیه جدا خواهد کرد.

آخرین مورد منویی است که برای خروج از برنامه به کار می‌رود. نام آن را از iMenuItem3 به iMenuItemExit تغییر دهید و از عبارت r_exit به عنوان Text-ID استفاده کنید. اکنون Command ID پیش‌فرض را نگه نخواهیم داشت، از EAknCmdExit به عنوان Command استفاده کنید. این دستور همچنین هنگامی که سیستم عامل بخواهد برنامه را ببندد به برنامه ارسال می‌شود، برای مثال هنگامی که گوشی حافظه باقیمانده کافی نداشته باشد. بنابراین ضروری است که هر برنامه‌ای همیشه به این رویداد پاسخ دهد و برنامه را ببندد.

حالا اگر بازی را تست کنید، خواهید دید که منوی Exit به درستی کار می‌کند! هنگامی که در آینده بازی را تست می‌کنید، همیشه توسط Exit از برنامه خارج شوید و فقط شبیه‌ساز را نبندید. به این علت که محیط شبیه‌ساز همیشه به طور اتوماتیک حافظه را بررسی می‌کند و در صورت وجود رخنه به شما اطلاع می‌دهد. اگر شما فقط شبیه‌ساز را ببندید، هنگام باز کردن مدت زیادی طول خواهد کشید که باعث می‌شود پیدا کردن علت خیلی سخت‌تر شود.

گام ۶ : نمایش دادن پنجره محاوره About

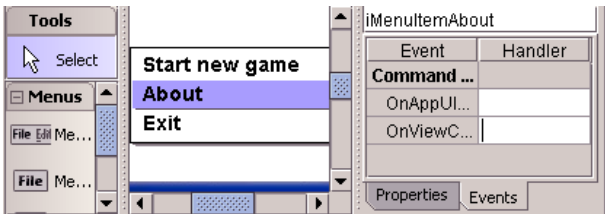
تاکنون ما یک برنامه کوچک با منوی اختصاصی‌اش بدون نوشتن هیچ کدی ایجاد کرده‌ایم. برای نشان دادن پنجره محاوره About، اولین کد را خواهیم نوشت. به نمای طراحی Non-visual بروید.



تصویر۱۰: اضافه کردن یک پنجره About به برنامه

از پانل ابزار سمت چپ CAknInformationNote را انتخاب کنید. این مورد، یک پنجره ساده کوچک است که متن شما و یک آیکن از پیش تعریف شده را دربرمی‌گیرد. سپس، در یک جایی از منطقه سفید وسطی کلیک کنید تا Note ایجاد شود. در پنجره ویژگی‌ها (properties) در سمت راست، نام آن را به iAknAboutNote تغییر دهید و r_aboutmessage را به عنوان Text-ID انتخاب کنید.

اکنون ما پنجره About را تعریف کردیم، برای نمایش دادن آن، ما فقط باید آن را به منوی about متصل (connect) کنیم. به Menu Designer بروید و منوی About را انتخاب کنید.



تصویر۱۱: صدا کردن (اجرای) پنجره About از طریق انتخاب منو

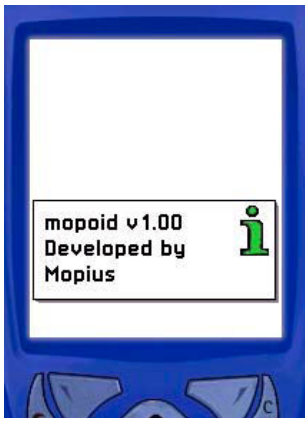
در بخش Properties در سمت راست، به قسمت Events بروید. در جعبه متن خالی که پس از عبارت OnViewCommand قرار دارد، دابل‌کلیک کنید. C++BuilderX به طور اتوماتیک یک تابع (function) برای شما ایجاد می‌کند که هنگامی که کاربر منوی About را انتخاب کرد، صدا زده می‌شود. ولی قبل از نوشتن هرگونه کد، به MopoidContainer.akn بروید و نام تابعی را که IDE جدیداً ایجاد کرده است را به OniMenuItemAboutViewCommandL (با افزودن یک L به انتهای آن) تغییر دهید. Enter را بزنید و سپس C++BuilderX شما را

دوباره به تابع می‌برد؛ اکنون تابع، نام تصحیح شده را دارد.

```
159
160 TInt CMopoidContainer::ExecuteiAknAboutNoteL()
161 {
162   /* 22.01.05 19:33 */
163   TBuf < 256 > TBuf_2561;
164
165   iCoeEnv->ReadResource( TBuf_2561, 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_2561 );
171 }
172
173 void CMopoidContainer::OniMenuItemAboutViewCommandL( TInt aCommand )
174 {
175   ExecuteiAknAboutNoteL();
176 }
177
```

تصویر۱۲: کد مورد نیاز برای نمایش دادن پنجره About

اکنون، کد موجود در خط ۱۷۵ در تصویر ۱۲ را در تابع جدید در فایل MopoidContainer.cpp بنویسید. این خط، اجرای تابعی است که C++BuilderX قبلاً برای ما ایجاد کرده است.



تصویر۱۳: پنجره About در شبیه‌ساز سری ۷۰ (SDK v1.2)

اگر برنامه را تست کنید، پنجره About به درستی کار خواهد کرد. ولی بیایید بررسی کنیم که چرا L را به انتهای نام تابع اضافه کردیم. توجه داشته باشید که تابع ExecuteiAknAboutNoteL() خودش یک L در انتهای نامش دارد. چرا اینگونه است؟ به این علت که حافظه در این تابع تخصیص یافته است. (برای ایجاد و نمایش پنجره‌های محاوره) و به همین خاطر این پروسه امکان دارد به طور کامل انجام نشود. در حال حاضر، گوشی‌های موبایل منابع خیلی محدودی دارند. بنابراین خیلی مهم است که همیشه احتیاط کنید که اگر سیستم عامل نتواند منابع مورد نیاز برنامه شما را تدارک ببیند، چه اتفاقی می‌افتد.

این رویدادها، همانند errorهای دیگر (مثلاً file not found) توسط سیستمی کنترل می‌شوند که شبیه به catch جاوا و C++ جدید است. یک error منتظر می‌ماند تا زمانی که کسی آن را مدیریت کند. افزودن L به انتهای تابع، یک قرارداد عمومی است که از طریق آن به برنامه‌نویس می‌گوییم که اگر در سیستم حافظه کافی وجود نداشته باشد، اجرای این تابع ضروری نیست.

تابعی که پیام ما را نمایش می‌دهد، در صورت عدم وجود حافظه کافی اجرا نمی‌شود. طبیعتاً شما نمی‌توانید این مشکل را رفع کنید و باید به سیستم اجازه دهید که error مناسب را نشان دهد. اگر خودتان تمام errorها را مدیریت کنید، برنامه شما خیلی بزرگ خواهد شد.

اگر یک error وجود داشته باشد، رویداد به طور اتوماتیک به تابع بعدی خواهد رفت. به این معنی که تابع OniMenuItemAboutViewCommandL() نیز می‌تواند به طور کامل انجام نشود. بنابراین، ما L را به انتهای نام آن اضافه کردیم.

تا حالا C++BuilderX یک تابع با نام DispatchViewCommandEvents() نوشته است که به دنبال آن توضیح «این روتین توسط C++BuilderX ایجاد شده است, آن را تغییر ندهید» است, آمده است.

توجه داشته باشید که حرف L در انتهای آن نیامده است. اگر ما خود آن را اضافه کنیم, C++BuilderX این تابع را نخواهد شناخت. اگر به فایل MopoidView.cpp بروید تابع HandleCommandL() را خواهید دید. این تابع, تابعی است که هرگاه کاربر منوی About را انتخاب کرد, توسط سیستم عامل صدا زده می‌شود.

بنابراین، به خاطر انعطاف‌پذیری‌های کم IDE، ما از قبل مقدار کمی کد داریم که با استانداردهای کدنویسی سیستم عامل سیمبین همخوانی ندارند. در این مورد، آنها را نادیده می‌گیریم، همانطور که L برای سیستم نیست و فقط یک چیز کمکی برای ما برنامه‌نویس‌ها است. ولی یک بهانه خوب برای شرح دادن نکات اساسی سیستم و نشان دادن محدودیت‌های کدهای ایجاد شده توسط C++BuilderX به شما است.

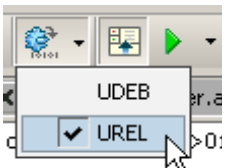
اگر چیزی کار نکند...

... و شما مایلید که علتش را بدانید، فایل Mopoid.step7.zip یک پروژه کاری از آنچه که تاکنون انجام داده‌ایم است. می‌توانید به آن مراجعه کنید. از حالا به بعد، برای هر مرحله، یک پروژه آماده در فایل zip شده همراه این کتاب، موجود است.

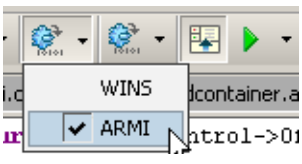
گام ۷ : فرستادن برنامه به گوشی موبایل

طبیعتاً، شبیه‌ساز به خوبی کار می‌کند. ولی بهتر است از مواقعی که فایل در شبیه‌ساز ویندوز درست کار می‌کند ولی در گوشی کار نمی‌کند، آگاهی یابید. بهترین مثال ممکن، متغیرهای استاتیک هستند، که در شبیه‌ساز به خوبی کار می‌کنند، ولی در گوشی موبایل کار نمی‌کنند. به علت اینکه اشکالزدایی مستقیم در موبایل سخت است و بعضی مواقع کار نمی‌کند، و سیستم فقط پیام system error می‌دهد، یافتن علت کار نکردن برنامه در موبایل سخت است، ولی اگر به طور منظم برنامه را در موبایل امتحان کنید، پیدا کردن اشکالات راحت‌تر خواهد شد.

برای اینکه به IDE بگویید که می‌خواهید برای دستگاه موبایل برنامه بسازید، باید پلت‌فرم مقصد برای برنامه کامپایل را به ARMI تغییر دهید. (پردازنده گوشی سری ۶۰ شما، یک پردازنده ARM است) و نوع آن را به UREL (ویرایش انتشاری) تغییر دهید. برای ساخت پروژه‌تان، به مسیر Project → Make project 'Mopoid.cbx' بروید. (همچنین می‌توانید Ctrl + F9 را نیز بزنید.)



تصویر۱۴: کامپایل کردن برای انتشار به جای کامپایل کردن برای اشکالزدایی



تصویر۱۵: تغییر هدف ساخت برنامه

بعد از اینکه این پروسه تمام شد، یک فایل با نام Mopoid_ARMI_UREL.sis در پوشه C:\Symbian\dev\Mopoid\group مستقر خواهد شد. این فایل را به گوشی‌تان انتقال دهید و آن را نصب کنید.

شما می‌توانید انتقال را با PC Suite و یا بلوتوث از طریق راست کلیک کردن روی فایل و انتخاب Send to… → Bluetooth device و یا کابل اینفرارد انجام دهید. برای جزئیات بیشتر در مورد نحوه نصب کردن، به راهنمایی‌های گوشی خودتان مراجعه کنید.

اگر همه چیز به درستی کار کند، فراموش نکنید که برای ادامه برنامه‌نویسی، تنظیمات ساخت را به WINS / UDEB برگردانید!

گام ۸ : اضافه کردن موتور بازی به پروژه

طراحی یک بازی کامل، مدت زمان طولانی نیاز دارد، و در این خودآموز، ما می‌خواهیم که پایه‌های اساسی برنامه‌نویسی برای سیستم عامل سیمبین را مرور کنیم نه صرف کردن زمان طولانی برای ایجاد روتین‌های بازی. به همین خاطر ما چند فایل از قبل نوشته شده را به پروژه وارد می‌کنیم و چند خصیصه جالب و مهم را اضافه می‌کنیم.

لطفاً توجه کنید: امکان این که یک بازی را با ساختاری شی‌گرا بسازیم، وجود دارد؛ ولی فایل‌های تدارک دیده شده، به نوعی نوشته شده‌اند که درک موضوعات مربوط به سیستم عامل سیمبین را راحت‌تر کنند، و برای یک برنامه شی‌گرای کامل تهیه نشده‌اند.

تمام مراحل زیر، هنگامی که شما پروژه شخصی خودتان را می‌نویسید، مهم خواهد بود. شما همچنین باید حتماً فایل‌های گرافیکی و صوتی و سورس‌فایل‌های آماده و بعضی چیزهای دیگر را اضافه کنید.

آنزیپ کردن (Unzip) فایل

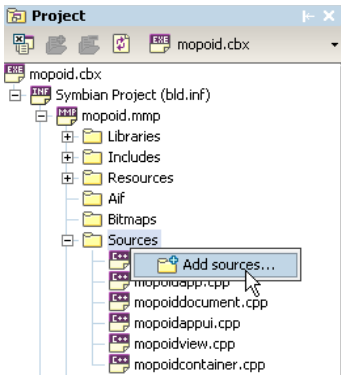
فایل Mopoid.Step8.Update.zip را به پوشه C:\Symbian\dev\ اکسترکت (استخراج) کنید. اطمینان حاصل کنید که شما ساختار پوشه‌ای آرشیو را حفظ کرده‌اید! تمام فایل‌ها را بازنویسی کنید. اگر هیچ اخطاری برای بازنویسی داده نشد، شما آن را به پوشه صحیح اکسترکت نکرده‌اید.

شرح داده‌ها و اطلاعات بازی

آرشیو، حاوی یک پوشه جدید با نام Data است که حاوی فایل‌های گرافیکی و صوتی برای بازی ما است. هنگامی که شما برنامه را در شبیه‌ساز اجرا می‌کنید، تمام فایل‌های موجود در پوشه خودش را جستجو می‌کند. فایل \data\dobitmaps.bat را اجرا کنید تا به طور اتوماتیک پوشه نظیر را ساخته و همه فایل‌های اطلاعاتی را در آنجا کپی کند. اگر SDK سری ۶۰ شما در مسیر پیش‌فرض (C:\Symbian) نصب نشده است، و یا اگر شما از یک SDK غیر از ویرایش 1.2 استفاده می‌کنید، شما در ابتدا مجبورید که اسامی پوشه‌ها را در فایل batch تصحیح کنید. (برای تغییر دادن آنها، روی فایل راست‌کلیک کنید و Edit را انتخاب کنید و بعد از درست کردن آدرس‌ها، تغییرات را ذخیره کنید.)

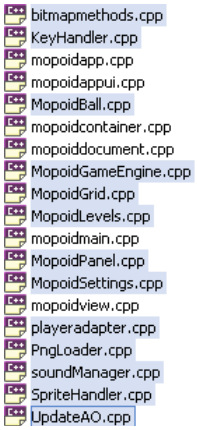
اضافه کردن سورس‌فایل‌ها به پروژه

در مرحله بعد، شما باید به IDE بگویید که می‌خواهید چند سورس‌فایل جدید را به پروژه اضافه کنید. در پنجره project روی آیتم Sources راست‌کلیک کنید و ...Add sources را برگزینید.



تصویر۱۷: اضافه کردن سورس‌فایل‌های دیگر

تمام فایل‌های پوشه /src/ را که هنوز جزئی از برنامه شما نیستند، انتخاب کرده و اضافه کنید. (مطابق تصویر ۱۷)



تصویر۱۷:فایل‌هایی که باید اضافه شوند.

اضافه کردن کتابخانه‌های خارجی به فایل تعریف پروژه

بازی نهایی شما، کارهای زیادی انجام خواهد داد. (مانند لود کردن و اجرای صداها، نمایش گرافیک‌های PNG و دسترسی به فایل‌ها) این توابع به چند کتابخانه نیاز دارند که توسط SDK تدارک دیده شده‌اند. تنها چیزی که شما باید انجام دهید، این است که آنها را به پروژه اضافه کنید. روی Mopoid.mmp در پنجره تعریف پروژه‌تان دابل‌کلیک کنید. این فایل، فایل تعریف پروژه سیستم عامل سیمبین است که حاوی ارجاع‌هایی به سورس‌فایل‌ها و کتابخانه‌های مورد نیاز پروژه شما (و چند چیز دیگر) است. به انتهای فایل بروید و خطوط زیر را که تعریف تمامی کتابخانه‌های دیگر است، اضافه کنید.

```
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
```

تعریف متن جدید

یک بازی واقعی، مقدار زیادی متن نیاز دارد. رشته‌های زیر را در جدول رشته‌ها (string table) همانطور که در گام ۴ توضیح داده شد، تعریف کنید. در هر مورد بعد از علامت : یک فاصله نیز بزنید، به خاطر این که یک عدد به متن‌ها اضافه خواهد شد.

نام	متن
r_score	Score:
r_level	Level:
r_pause	Game Paused
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

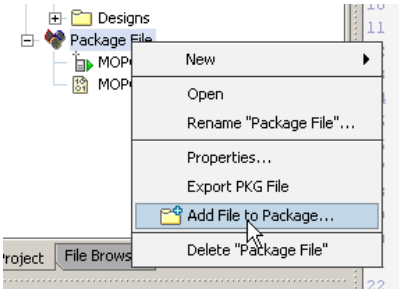
بازی جدید

دستور شروع بازی جدید باید به تابع شروع بازی در موتور بازی، متصل شود. یک تابع رویدادی جدید برای منوی Start New Game ایجاد کنید. (شبیه همانچه که برای پنجره About انجام دادید.) ولی اکنون نام تابع را تغییر ندهید که حاوی پسوند L شود (زیرا تابع شروع بازی، حتماً باید اجرا شود.) متن زیر را به تابع جدید OniMenuItemNewGameViewCommand اضافه کنید.

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

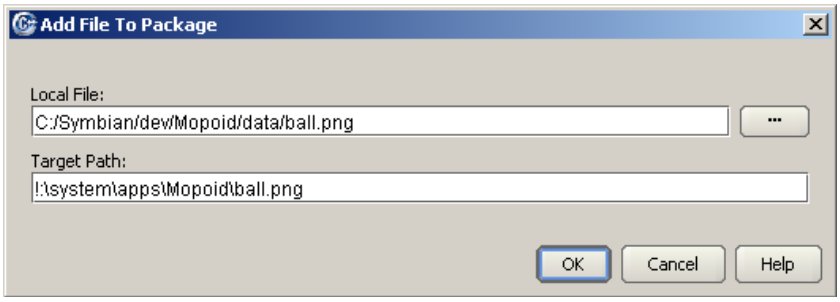
اضافه کردن فایل‌های صوتی و تصویری

چند فایل png. و wav. به صورت آماده وجود دارند. شما باید آنها را به پروژه اضافه کنید، که در آن صورت آنها به موبایل کپی خواهند شد. برای انجام این کار، تنظیمات ساخت را با استفاده از چرخ‌دنده‌ها به ARMI UREL تغییر دهید. الان شما باید یک ورودی با نام Package File در پانل project ببینید. هنگام اضافه کردن فایل‌ها، مواظب باشید، زیرا C++BuilderX یک باگ (اشکال) کوچک دارد.



تصویر ۱۸: اضافه کردن فایل‌های خارجی به پکیج

روی Package File راست‌کلیک کنید و Add File to Package… را برگزینید. یک پنجره محاوره باز خواهد شد. اولین فایل پوشه /data/ از پروژه Mopoid را انتخاب کنید. (فایل ball.png) هنوز روی OK کلیک نکنید زیرا در غیر این صورت مجبور خواهید شد که فایل را از پکیج پاک کنید و دوباره شروع کنید.



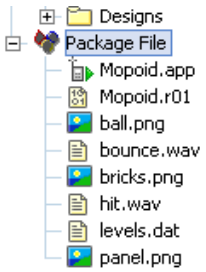
تصویر۱۹: نام فایل را به Target Path اضافه کنید!

نام فایل (\ball.png) را به Target Path در دومین جعبه متن، اضافه کنید. هنگامی که انجام دادید، روی OK کلیک کنید.

Target Path نام و مکان فایل را در دستگاه مشخص می‌کند. اگر نام فایل تعیین نشود، گوشی قادر به کپی کردن فایل نخواهد بود! علامت ! در ابتدای آدرس، به این معنی است که کاربر می‌تواند انتخاب کند که برنامه را به کجا نصب کند؛ بعداً در این مورد بیشتر توضیح داده خواهد شد.

باگ موجود در C++BuilderX 1.5 این است که: بعد از اینکه یک فایل را اضافه کردید، شما هرگز نخواهید توانست Target Path را تغییر دهید. IDE به شما اجازه تغییرش را می‌دهد، ولی هنگامی که روی OK کلیک می‌کنید، تغییرات شما را بدون اطلاع دادن به شما، نادیده می‌گیرد. بنابراین، فراموش نکنید که نام فایل را بعد از اضافه کردن فایل، اضافه کنید!

این کار را برای تمام فایل‌های .wav و .png و فایل levels.dat انجام دهید. هنگامی که تمام کردید، پانل project شما شبیه تصویر ۲۰ خواهد بود:



تصویر۲۰: اکنون تمام فایل‌ها بخشی از پروژه هستند.

تست کردن

به حالت WINS / UDEB بروید. هنگامی که بازی‌تان را اجرا می‌کنید، باید هنوز هم کار کند. اگرچه صفحه فقط سیاه است و فقط یک متن در آن دیده می‌شود. در چند مرحله بعدی، ما تمام قسمت‌هایی را که نیستند، اضافه خواهیم کرد.

کشف و رفع اشکال

این بخش، نسبتاً ریسک‌دار است. اضافه کردن سورس‌فایل‌های آماده به پروژه، فقط در صورتی بدون اشکال خواهد بود که شما دقیقاً مانند مطالب توضیح داده شده در مراحل قبلی انجام دهید. اگر شما هنوز هم مشکلی دارید و پروژه کار نمی‌کند، از پروژه موجود در Mopoid.Step8.zip استفاده

کنید.

گام ۹ : لود کردن تصاویر

C++ سیستم عامل سیمبین دارای پشتیبانی داخلی از فایل‌های mbm. است. این فایل‌ها، مجموعه‌ای از فایل‌های تصویری فشرده شده RLE هستند. مزیت‌های آنها عبارتند از: لود کردن آنها سریع و کار کردن با آنها راحت‌تر است، ولی آنها فضای زیادی از دستگاه موبایل را اشغال می‌کنند. بازی که از تصاویر زیادی به این سیستم استفاده می‌کند، بسیار بزرگ خواهد شد. بنابراین، بهتر است که از فایل‌های png. یا jpg. استفاده کنیم. متأسفانه روتین‌ها و متغیرهای از پیش تعریف شده‌ای برای لود کردن این گونه تصاویر وجود ندارد و انجام دادن دستی آنها نیز نسبتی پیچیده است. همچنین باز کردن آنها از حالت فشرده شده، مقداری زمان می‌برد، بنابراین آنها به طور غیرهمزمان کار خواهند کرد. در سیستم عامل سیمبین ویرایش 7.0، یک روش انتخابی دیگر نیز برای لود کردن تصاویر ممکن است، ولی اگر می‌خواهید بازی شما توسط گروه زیادی از مردم استفاده شود، مجبوری که از توابع قدیمی استفاده کنید. برنامه‌نویسی و شرح کامل پروسه لود کردن یک تصویر، خودش می‌تواند به اندازه یک خودآموز باشد، بنابراین یک کلاس کامل از قبل تهیه شده است (PngLoader.cpp). و یک فایل دیگر هم توسط نوکیا تهیه شده است (bitmapmethods.cpp) که در بعضی از وظایف عمومی کمک می‌کند. بعضی توابع دیگر نیز به آن اضافه شده است که بسیار پرکاربرد هستند. شما می‌توانید به راحتی آن دو فایل را به پروژه‌های شخصی خودتان اضافه کنید!

در پروژه Mopoid، یک کلاس، مسئول لود کردن و ذخیره کردن تمام تصاویر است. اگر یک تصویر در جای دیگری مورد نیاز باشد، می‌تواند آن تصویر را با فرستادن ID تصویر به تابع کلاس CSpriteHandler دریافت کند.

کلاس‌های C و T

به نظر شما معنی C در اول نام کلاس چیست؟ به طور مختصر، به این معنی است که این کلاس، همیشه به حالت گروه ایجاد می‌شود و اکثر مواقع آبجکت‌های شخصی خودش را دارد. هنگامی که مخرب (desstructor) کلاس C ما صدا زده شد، آن آبجکت‌ها (اشیا) نابود خواهند شد. دومین نوع مهم دیگر، کلاس‌های T هستند. این کلاس‌ها در حقیقت شبیه به یک نوع داده (data type) ساده هستند، یک کلاس از این نوع، نمی‌تواند مخربی داشته باشد. برای مثال بنیادی‌ترین نوع داده‌های سیمبین، دارای یک پیشوند T هستند. (TInt, TReal, …)

ConstructL

فایل SpriteHandler.cpp را به منظور دیدن کدهای کلاس مدیر تصویر (bitmap manager class) باز کنید. توجه خواهید داشت که سازنده (Constructor) خالی است، و شما کدتان را در تابعی به نام ConstructL() می‌نویسید، که توسط NewL() صدا زده می‌شود.

علت این کار، در وضعیت حافظه‌های موبایل‌ها است. اگر شما یک برنامه برای PC بنویسید، (در بیشتر موارد) نگران حافظه آزاد نیستید. در اینجا، وضعیت به گونه دیگری است. ممکن است که حافظه موبایل تماماً اشغال شود و ایجاد یک شیء از یک کلاس، انجام نشود.

به هر حال، هنگامی که شما حافظه را در سازنده اختصاص می‌دهید، و این پروسه به طور کامل انجام نمی‌شود، حافظه اختصاص داده شده بی‌صاحب باقی می‌ماند و این بد است. بنابراین کد اجرا شده در یک سازنده C++ هرگز نباید ناتمام رها شود!

راه حل ساده است (یک سازنده دو مرحله‌ای)؛ فقط تمام کدهای اختصاص دادن حافظه را به متد ConstructL() انتقال دهید. حالا شما

می‌توانید آبجکت یا شی را به طور عادی ایجاد کنید، و سپس در خط بعدی، این تابع را صدا بزنید. این کار به شما فرصت کنترل کردن عملیات رها شده را به طور صحیح می‌دهد.

به ویژه اگر شما نیاز دارید که بیش از یک شیء از کلاستان ایجاد کنید، باید آن کدها (اختصاص دادن و صدا زدن ConstructL()) را به تعداد متعدد بنویسید. برای جلوگیری از این کار، سازنده دوبخشی معمولاً توسط یک NewL() استاتیک یا تابع NewL() مدیریت می‌شود. به عنوان یک قاعده کلی، هنگامی که شما می‌خواهید یک آبجکت را به یک متغیر نسبت دهید، NewL() باید اجرا شود. NewL() آبجکت را در پشته cleanup می‌گذارد و برای متغیرهای اتوماتیک مفید است. در مورد این موضوع بعداً بیشتر بحث خواهیم کرد. حالا، بیایید به کدهای مربوط به لود کردن تصاویر نگاهی بیندازیم.

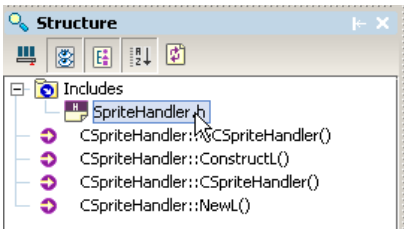
نحوه لود کردن تصاویر

ابتدا باید نام فایل تصویری را که می‌خواهیم لود کنیم، تعریف کنیم. این کار به صورت زیر انجام می‌شود:

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

این خط یک آبجکت با نام KBmpPanel می‌سازد که رشته "panel.png" را به طور مستقیم و به صورت باینری در برنامه ذخیره می‌کند. رشته‌ها در C++ سیستم عامل سیمبین متفاوت با C++ استاندارد کنترل و مدیریت می‌شوند. باز هم علت این کار، محدودیت حافظه موبایل است. در C++ سیستم عامل سیمبین، رشته‌ها، **واصف** (descriptor) نامیده می‌شوند. عادت کردن به آنها واقعاً سخت است. ولی از طرف دیگر، کمتر از رشته‌های C++ استاندارد حافظه اشغال می‌کنند.

هنگامی که ما یک تصویر را لود کردیم، نیاز داریم که آن را در جایی ذخیره کنیم. برای دیدن آن، شما باید یک نگاه به فایل سرآیند کلاس sprite handler بیندازید. به دلایلی نامعلوم، C++BuilderX آن فایل‌ها را در ساختار پروژه نشان نمی‌دهد. به جای آن، شما باید سورس‌فایل را باز کنید، بخش ضمیمه‌های آن را در پانل Structure در سمت چپ باز کنید، و از آنجا به فایل ضمیمه بروید:



تصویر۲۱: چگونه به فایل‌های ضمیمه شده برویم.

در اعلان کلاس (class declaration)، شما اعلان‌های زیر را که مربوط به یک متغیر خصوصی (private) است، خواهید دید.

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

چرا متغیر به جای m_sprites یا sprites، iSprites نامیده شده است؟ این مورد نیز یک رهنمون دیگر کدنویسی C++ سیستم عامل سیمبین است. تمام متغیرهای عضو، باید دارای یک پسوند i باشند. به این ترتیب، تمام متغیرهای پارامتری نیز باید یک a در ابتدایشان داشته باشند.

به فایل Spritehandler.cpp برگردید. بیایید آرایه‌مان را با تصاویر پر کنیم. ما این کار را با صدا زدن یک متد استاتیک از کلاس CPngLoader انجام می‌دهیم.

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

توجه داشته باشید که ما به کلاس می‌گوییم که تصویر را به عنوان EColor4K لود کند. به این معنی که تصویر، ۴۰۹۶ رنگ است که این مقدار

برابر عمق رنگ دستگاه‌های سری ۶۰ قبلی مانند Nokia N-Gage و Nokia 7650 است. اکنون دستگاه‌های جدید، EColor64K (یا بالاتر) را هم پشتیبانی می‌کنند. اگرچه برای گرافیک‌های یک بازی ساده مانند Mopoid، رنگ‌های زیادی نیاز نداریم.

ESpritePanel یک ارائه متنی یک ID است که enumeration نامیده می‌شود و در MopoidSharedData.h تعریف شده است.

تصویر۲۲: ماسک پانل

تصویر۲۳: تصویر پانل

ما می‌خواهیم که پانل ما، گوشه‌های گرد داشته باشد، پس ما یک ماسک نیاز خواهیم داشت. یک روش برای انجام این کار، این است که یک فایل تصویری دومی داشته باشیم که حاوی تصویر ماسک سیاه و سفید است. به هر حال، همانطور که قبلاً گفته شد، لود کردن فایل‌های png. مدتی طول می‌کشد، پس بهتر است که ماسک را خودمان ایجاد کنیم. از کد زیر استفاده کنید:

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

این تابع یک تصویر ماسک بر پایه رنگ پیکسل بالا و سمت چپ تصویر منبع، ایجاد می‌کند. اگر شما نمی‌خواهید که تمام پیکسل‌هایی که رنگ مشخص شده را دارند، شفاف شوند، می‌توانید از دومین و سومین پارامتر جهت مشخص کردن رنگ مورد نظرتان استفاده کنید.

حالا، تصویر توپ را به همان روش به عنوان تصویر پانل، لود کنید. نام فایل، ball.png است، اسامی enumeration برای IDها عبارتند از: ESpriteBall و ESpriteBallM.

آجرها، مقداری متفاوت هستند. آنها شکل مستطیلی دارند و به ماسکی نیاز ندارند. البته، ما رنگ‌های متفاوتی را به آنها نسبت می‌دهیم. به جای لود کردن یک به یک تصاویر، بهتر است که تمام گرافیک‌ها را در یک فایل جمع کنیم و بعد از لود کردن، آنها را جدا کنیم. با این روش، فایل‌های تصویری فضای کمتری اشغال می‌کنند، و اگرچه برنامه حافظه موقت (Temporarily memory) بیشتری نیاز خواهد داشت، ولی سرعتش به مقدار زیادی افزایش خواهد یافت.

تصویر۲۴: تصویر ۳ آجر در یک فایل

تصویر آجر را با کد زیر لود کنید:

```
_LIT(KBmpBrick, "bricks.png");
CPngLoader::LoadAndSplitImageL(KBmpBrick, &iSprites[ESpriteBrickNormal], 3, EColor4K);
```

این کد فایل تصویری را لود می‌کند، آن را به سه تصویر تقسیم می‌کند و آنها را در آرایه iSprites قرار می‌دهد، که با موقعیت ESpriteBrickNormal شروع می‌شود.

حذف تصاویر

هنگامی که شما سعی می‌کنید که برنامه را اجرا کنید، برنامه باید تصاویر را لود کند، که شما آن را نخواهید دید. ما کد مورد نیاز برای نمایش آنها را هنوز ننوشته‌ایم. الان سعی کنید که از برنامه خارج شوید. شما یک اخطار شبیه تصویر ۲۵ خواهید دید.

این error به شما از وجود یک رخنه در حافظه قسمتی از برنامه شما اطلاع می‌دهد. پیدا کردن عامل صدمه زننده، واقعاً سخت است؛ به همین دلیل، همیشه به یاد داشته باشید که به جای فقط بستن پنجره شبیه‌ساز، از خود برنامه خارج شوید.

تصویر۲۵: این پیام به شما درباره یک رخنه حافظه، آگاهی می‌دهد.

در اینجا، ما حافظه اختصاص یافته به فایل‌های گرافیکی را هنگامی که آبجکت CSpriteHandler نابود شد، آزاد نمی‌کردیم. خوشبختانه یک راه حل راحت وجود دارد. کلاس TFixedArray از iSprites این کار را به راحتی برای ما انجام می‌دهد. نوشتن کد زیر در مخرب کلاس، تمام کار را انجام می‌دهد:

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

گام ۱۰ : نمایش تصاویر

تنها لود کردن تصاویر به شما کمک زیادی نمی‌کند. (شما باید آنها را به صفحه نمایش نیز بیاورید.) در برنامه ما، کلاس موتور گرافیکی بازی، مواظب آماده‌سازی بافر پشتی تصویر است. به این معنی که تمام گرافیک‌ها در یک تصویر خارج از اندازه صفحه کشیده می‌شوند. هنگامی که این پروسه تمام شد، تصویر کلی به صفحه نمایش کپی می‌شود. این روش از خاموش و روشن شدن تصویر جلوگیری می‌کند، که در صورت عدم استفاده از روش مذکور، ممکن بود اتفاق بیفتد.

ابتدا به متد ConstructL() از کلاس CGameEngine بروید. کدی که چند محاسبات اندازه‌ای روی فایل‌های گرافیکی که اخیراً لود کردیم، انجام می‌دهد را از حالت کامنت خارج کنید. اگر مایل هستید، می‌توانید نگاهی به کد بیندازید، ولی این کد الان برای ما مهم نیست.

خود قاب (frame) در متد DrawFrame() کشیده می‌شود، که در پایین سورس‌کد در همان کلاس است. توجه داشته باشید که تابع از نوع const تعریف شده است، به این معنی که تغییر اطلاعات عضو کلاس موتور بازی، امکان‌پذیر نیست.

کنترل و مدیریت شفافیت

بیایید نگاهی به خط زیر داشته باشیم:

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

هنگام کشیدن یک تصویر سیستم عامل سیمبین، یک قلم (pen) و یک قلم‌مو (brush) به خاطر زمینه گرافیکی یک تصویر ایجاد می‌کند، که عملیات کشیدن و نقاشی کردن را مدیریت می‌کنند. یک مثال: هنگامی که شما یک مستطیل می‌کشید، خطوط مستطیل توسط تنظیماتی که شما به قلم اعمال کرده‌اید، کشیده می‌شوند، و توسط قلم‌مو، مستطیل رنگ می‌شود.

ما از آن توابع کشیدن استفاده نمی‌کنیم، ولی باید کاملاً مواظب این رفتار باشیم! اگر یک قلم‌موی سخت انتخاب کنیم و بخواهیم یک تصویر دارای قسمت‌های شفاف بکشیم، آن قسمت‌ها با رنگ کنونی قلم‌مو و مشخصات دیگر آن، پر خواهند شد! به همین خاطر است که ما قلم‌مو را قلم‌موی

تهی تنظیم کردیم، که به پس‌زمینه یک تصویر اجازه می‌دهد که در قسمت‌های شفاف آن دیده شود.

کشیدن یک تصویر

ابتدا، توپ را خواهیم کشید. اولین خط از قبل وجود دارد، این خط موقعیت توپ را محاسبه می‌کند و آن را در یک متغیرTPoint قرار می‌دهد. این نوع متغیر به طور اتوماتیک حاوی یک x و یک y مختصاتی است و بسیار هم پرکاربرد است.

این خط را مستقیماً بعد از آن بنویسید:

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

این خط، تابع BitBltMasked() محتوای گرافیک بافر پشتی را صدا می‌زند، که در حقیقت یک تصویر منبع ماسک زده را به تصویر مقصد کپی می‌کند. اولین پارامتر، موقعیت را مشخص می‌کند، دومی تصویر مبدأ است، سومی اندازه تصویری است که ما می‌خواهیم کپی کنیم، چهارمی ماسکی است که می‌خواهیم استفاده کنیم و پنجمی مشخص می‌کند که ماسک معکوس شود یا نه.

برای جستجوی پارامترهای توابع تهیه شده توسط APIهای سیستم عامل سیمبین، می‌توانید نگاهی به help سیمبین داشته باشید. به SDK Help در مسیر Start Menu → Symbian 6.1 SDKs → Series 60 → Documentation بروید و در آنجا جستجو کنید.

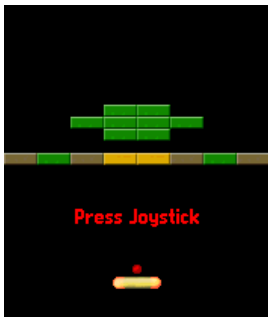
حالا مقداری به پایین بیایید و همان کار را برای پانل انجام دهید. برای موقعیت آن، از مقدار ذخیره شده در iPanel.iPos استفاده کنید. IDهای enumeration برای spriteها عبارتند از ESpritePanel و ESpritePanelM. اندازه، iPanel.iSize است.

همانطور که قبلاً دانستیم، آجرها به شفافیت نیاز ندارند. در ضمن شما باید مواظب باشید که منطقه‌های شفاف بزرگی نکشید، زیرا که سرعت برنامه را کمتر می‌کند.

تابع کشیدن یک تصویر بدون یک ماسک (BitBlt()) به تابع قبلی شبیه است و کاربرد آن مقداری راحت‌تر است. شما فقط دو پارامتر دارید، موقعیت و تصویر منبع. کد زیر را در قسمت علامتگذاری شده در حلقه for که برای کشیدن تمام آجرهای مرئی تکرار می‌شود، قرار دهید.

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

اگر حالا برنامه‌تان را اجرا کنید، نسبتاً بهتر از قبل خواهد بود:



تصویر۲۶:بازی Mopoid ما، اکنون می‌تواند تصاویر را نمایش دهد.

البته هنوز هیچ گونه واکنش و فعالیتی ندارد. برای فعال کردن این بخش، کلیدهای زده شده را کنترل و مدیریت کنید.

گام ۱۱ : مدیریت کلیدها

مدیریت کلیدهای فشرده شده در سیستم عامل سیمبین نسبتاً راحت است. برنامه به طور اتوماتیک، متد OfferKeyEventL() را از کلاس حامل (CMopoidContainer) صدا می‌زند. کدی که توسط C++BuilderX ایجاد شده است، این اجرا را به تابع HandleKeyEvents() ارسال می‌کند. (که در مورد برنامه ما ضروری نخواهد بود)

تابع HandleKeyEvents() دو پارامتر می‌گیرد (const TkeyEvent&aKeyEvent و TeventCode aType). اولی (aKeyEvent) حاوی اطلاعاتی در مورد دکمه زده شده است. دومی (aType) به شما می‌گوید که چه نوع رویدادی اتفاق افتاده است.

در بازی Mopoid، ما می‌خواهیم که تازمانی که کاربر کلید جهت (در این برنامه، کاربر باید کلید joystick را بزند) را نگه داشته است، پانل به طور پیوسته حرکت کند. برای داشتن این رفتار، ما باید یک فلگ را به کار بیندازیم و بعد از رها شدن دکمه، آن را از کار بیندازیم.

یک کلاس مخصوص که از قبل آماده شده است (TkeyHandler)، جهتی که کاربر فشار می‌دهد را، نگهداری می‌کند. این کلاس، مخصوص کلاس موتور بازی ما است که وظیفه‌اش بررسی ارسال رویدادهای حرکتی به پانل است.

وظیفه‌ای که ما باید الان انجام دهیم، تا اندازه‌ای ساده است؛ بنابراین ما فقط کد رویداد نگه داشتن کلید را می‌نویسیم. سورس‌کدهای از قبل نوشته شده، حاوی بقیه کدها هستند. در فایل MopoidContainer.cpp به دنبال قسمتی از دستور if بگردید که رویدادهای نگه داشتن کلید (EeventKeyDown) را مدیریت می‌کند. در داخل آن، شما یک عبارت switch خواهید دید که کلیدهای منحصر به فرد را مدیریت می‌کند که توسط اسکن‌کدهایشان از بقیه جدا شده‌اند.

در اینجا دو مورد دیگر را نیز کنترل خواهیم کرد: EstdKeyLeftArrow که هنگامی که کاربر joystick را به چپ حرکت داد، صدا زده می‌شود و EstdKeyRightArrow برای جهت دیگر. برای ارسال رویداد به قسمت مدیریت کلیدها که قبلاً ذکر کردیم، شما باید مورد زیر را صدا بزنید:

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

مطمئناً این کد برای جهت راست، متفاوت خواهد بود. اگر شما به تابع HandleKeyEvents() نگاهی بیندازید، متوجه خواهید شد که مقدار برگشتی آن از نوع بولین (منطقی یا بولی) است. این مقدار به سیستم می‌گوید که آیا فشار دکمه، مدیریت شد یا نه. اگر نه، به برنامه بعدی که ممکن است در پس‌زمینه اجرا شود، ارسال می‌شود. (سیستم عامل سیمبین یک سیستم عامل چند وظیفه‌ای یا multitasking است!)

به همین خاطر ما به سیستم می‌گوییم که این زده شدن دکمه را مدیریت می‌کنیم و لازم نیست به برنامه دیگری ارسال شود. بنابراین فلگی که در ابتدای تابع تعریف شده است را به صورت زیر بنویسید:

```
handled = ETrue;
```

توجه داشته باشید که C++ سیستم عامل سیمبین، تعاریف شخصی خودش را از مقادیر بولی دارد. این نوع داده، Tbool است و می‌تواند حاوی مقادیر ETrue و EFalse باشد.

فراموش نکنید که یک عبارت break بعد از هر case که در عبارت switch نوشتید، بنویسید.

بعد از اینکه کد را نوشتید، آن را در شبیه‌ساز تست کنید. شما باید قادر به انتقال پانل باشید، توپ حالا می‌تواند بپرد و شما حالا می‌توانید بازی

کنید! کد این مرحله، همان کد C استاندارد است، اگر شما مشکلی داشتید، یک نگاه به فایل MopoidContainer.cpp در Mopoid.Step11.zip بیندازید.

گام ۱۲ : نمایش متن

بازی هنوز فاقد یکی از قسمت‌های مهم است و هیچگونه اطلاعاتی به کاربر در مورد امتیازها، تعداد جان یا مراحل نمی‌دهد. اینها را اکنون اضافه خواهیم کرد. به متد ()DarwFrame از کلاس CGameEngine بروید.

تنظیم فونت

خوشبختانه توابعی از قبل آماده وجود دارند که متون را مدیریت می‌کنند و قادرند آن را ترازبندی کنند. ولی اول، ما باید به زمینه گرافیکی تصویر بگوییم که از چه فونت و چه رنگی می‌خواهیم استفاده کنیم. کدهای زیر را بعد از کامنت Hud // بنویسید.

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

بیایید آن را جزء به جزء تحلیل کنیم. با خط اول آشنا هستیم و می‌دانیم که نوع قلم را تعریف می‌کند. در بخش‌های قبلی، شرح دادیم که برای چارچوب کلی جعبه‌ها استفاده می‌شود. همچنین برای زمینه کلی فونت نیز به کار می‌رود. در برنامه ما، نوع قلم را به solid تغییر می‌دهیم، بنابراین ما متنی که می‌کشیم را به طور واقعی می‌بینیم.

خط بعدی، رنگ را تعریف می‌کند. ما یکی از مقادیر پیش‌فرض را استفاده می‌کنیم که رنگ قرمز است. اگر مایلید که رنگ RGB را خودتان تعریف کنید، از دستور زیر استفاده کنید:

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

خط بعدی، یکی از فونت‌های استاندارد سیستم را برمی‌گزیند. سیستم عامل سیمبین مکانیسم‌های فونت زیادی را تدارک دیده است و همچنین امکان استفاده از فونت‌های شخصی خودتان را نیز می‌دهد. برای یک بازی Mopoid ساده، ما ساده‌ترین روش را انتخاب می‌کنیم و از AnnotationFont استفاده می‌کنیم. این فونت، یک فونت بولد (ضخیم) با سایز متوسط است.

چهارمین خط، یک واصف ایجاد می‌کند که معادل رشته در زبان‌های دیگر است. بیشترین اندازه آن ۳۰ کاراکتر است. TBuf کلاسی است که از کلاسTDes مشتق شده است و دارای توابع مختلفی برای دستکاری کردن داده‌های رشته‌ای است. ما از آنها برای قرار دادن محتوای متون استفاده می‌کنیم. توجه داشته باشید که TBuf در پشته ایجاد می‌شود که در موبایل‌ها، خیلی محدود است. بنابراین از آن برای رشته‌های بزرگتر از ۲۵۶ کاراکتر استفاده نکنید. برای موارد دیگر، از واصف‌های توده‌ای (HBufC) باید استفاده کنید.

خواندن متن از فایل‌های منبع (Resource)

در مرحله بعد ما یکی از رشته‌هایی را که در فایل منبع تعریف کردیم (از طریق جدول رشته C++BuilderX)، می‌خوانیم. یک تابع استاتیک از کلاس CEikonEnv این کار را انجام می‌دهد و متن را مستقیماً در داخل واصف می‌نویسد (که قبلاً تعریف کرده‌ایم). منبع رشته، نام RS_R_SCORE را می‌گیرد. البته ما آن را فقط R_SCORE تعریف کرده بودیم، ولی C++BuilderX یک _RS نیز به جلوی نام اضافه می‌کند.

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

شکلبندی و نمایش متن

رشته‌ای که ما از جدول خواندیم، فقط حاوی یک متن استاتیک (ثابت) است: "Score: " ما باید امتیاز کنونی بازیکن را نیز به انتهای آن اضافه کنیم. تابع AppendNum() که توسط واصف تدارک دیده شده است، این کار را انجام می‌دهد.

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

الان متن آماده نشان داده شدن در صفحه نمایش است. ما می‌توانیم موقعیت آن را در یک نقطه خاص به صورت زیر تنظیم کنیم:

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

لطفاً توجه کنید: مختصه ۲۵ برای y، پایین‌ترین قسمت متن را مشخص می‌کند، بنابراین (۵,۲۵) پایین و سمت چپ متن را مشخص می‌کند، نه بالا و سمت چپ که شما شاید انتظارش را داشتید!

ما همچنین می‌خواهیم بیشترین امتیاز را نیز نشان دهیم. کد این قسمت را خودتان بنویسید. متن RS_R_HIGHSCORE را از منبع به داخل همان واصف موقتی بخوانید. این کار، محتوایش را که قبلاً داشت، بازنویسی می‌کند که ما به آنها دیگر نیاز نداشتیم، زیرا قبلاً روی صفحه چاپ شده است. عدد را از متغیر iSettings.iHighScore بخوانید و به واصف الحاق کنید و متن را در موقعیت (۵,۳۸) چاپ کنید (یک خط پایین‌تر از خط قبلی).

ترازبندی (Aligning) متن

الان بازی امتیاز کنونی و بیشترین امتیاز را در سمت چپ صفحه نشان خواهد داد. ما همچنین می‌خواهیم اطلاعاتی در سمت راست صفحه نیز نمایش داده شود.

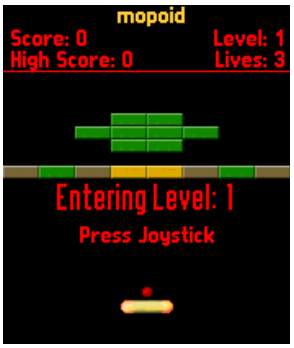
مراحل	**متن:** RS_R_LEVEL
	مقدار: iSettings.iLevel
	موقعیت: y: ۲۵ و x: ۳ واحد فاصله از سمت راست
تعداد جان‌ها	**متن:** RS_R_LIVES
	مقدار: iSettings.iLives
	موقعیت: y: ۳۸ و x: ۳ واحد فاصله از سمت راست

رشته‌ها را همانند قبل، بخوانید و شکلبندی مناسب را همانند قبل، انجام دهید. فقط چاپ متن متفاوت است:

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

TRect(…) یک مستطیل تعریف می‌کند که برای ترازبندی متن به کار می‌رود. در برنامه ما، اندازه مستطیل همان تمام صفحه است. برای بهینه‌سازی برنامه، شما باید TRect صفحه را فقط یک بار در ابتدای تابع تعریف کنید و از این متغیر در تمام توابعی که آن را نیاز دارند، استفاده کنید. SCREEN_WIDTH و SCREEN_HIGHT تعاریفی هستند که توسط ما ایجاد شده‌اند. سری ۶۰ دارای یک رابط کاربر (User Interface) استاندارد است که وضوح صفحه آن در تمام دستگاه‌ها، برابر 178x208 پیکسل است. در آینده این مقدار بیشتر خواهد شد و انعطاف‌پذیری بیشتری نیز خواهد داشت. پارامتر سوم، موقعیت y را مشخص می‌کند، چهارمین پارامتر نوع ترازبندی، معادل x را مشخص می‌کند.

پیام‌های وضعیتی نیز شبیه همین روش کار می‌کنند. برای فعال کردن آنها، کد زیر آن قسمت را از حالت کامنت خارج کنید. اگر نگاه کوتاهی به آن داشته باشید، خواهید دید که از فونت ()TitleFont استفاده می‌کند که از AnnotationFont بزرگتر است. این قسمت، همچنین کد را در وسط چاپ می‌کند. هنگامی که بازی را شروع می‌کنید، باید شبیه تصویر ۲۷ باشد.



تصویر۲۷:بازی Mopoid همراه با متن

شگفت‌آور است! نه؟ برای بهتر کردن بازی، هنوز چند قسمت باقی مانده است که باید اضافه کنیم...

گام ۱۳ : خواندن و نوشتن فایل‌ها

این بخش برای پروژه‌های آینده شما، خیلی مهم خواهد بود. در بازی Mopoid، ما فقط بالاترین امتیاز را ذخیره می‌کنیم نحوه کار خواندن و نوشتن فایل‌ها را شرح دهیم.

آمادگی

MopoidGameEngine.cpp را باز کنید و از قسمت Includes به فایل MopoidGameEngine.h بروید. قبل از اینکه اعلان کلاس شروع شود، خواهید دید که ما قبلاً شماره ورژن بازی و نام فایل را تعریف کرده‌ایم.

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

C++ سیستم عامل سیمبین، چند API برای دسترسی به فایل‌ها تدارک دیده است. ابتدا باید فایل سرآیند مربوطه، ضمیمه شود. به همین منظور، خط زیر را در کنار خط‌های مربوط به ضمیمه کردن فایل‌های دیگر، بنویسید:

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

این کار باعث می‌شود تا ما تمام توابعی که نیاز داریم، ضمیمه شوند. ما در مراحل قبلی، کتابخانه‌های مورد نیاز را به پروژه‌مان اضافه کرده‌ایم.

تنظیم نام فایل

در فایل سرآیند، ما فقط نام فایل اطلاعاتی‌مان را ذخیره کردیم، مسیر (آدرس) فایل باید به طور پویا توسط برنامه اضافه شود. این کار می‌تواند توسط اضافه کردن مسیر برنامه، انجام شود، زیرا ما می‌خواهیم فایل اطلاعاتی‌مان در همان پوشه‌ای که کاربر بازی را نصب کرده است، باشد. به فایل MopoidGameEngine.cpp برگردید و به پایین بروید تا به SaveGameProgressL() دسترسی داشته باشید. این کد ابتدا باید در این تابع اجرا شود:

```
TFileName fullName(KMopoidDataFile);
CompleteWithAppPath(fullName);
#ifdef __WINS__
fullName[0] = 'C';
```

```
#endif
```

همانطور که می‌بینید، ما باید بررسی کنیم و ببینیم که آیا بازی در شبیه‌ساز ویندوز اجرا می‌شود یا نه. اگر در شبیه‌ساز اجرا شود، محیط به ما جواب می‌دهد که برنامه در درایو Z نصب شده است، که همان درایو ROM است (جایی که سیستم خودش در تلفن نصب شده است). در ظاهر، برنامه به این درایو (مجازی) کامپایل شده است. این درایو قابل نوشتن نیست، بنابراین ما باید درایو را به درایو C تغییر دهیم. در موبایل، شما آدرس واقعی جایی که برنامه نصب شده است را خواهید گرفت.

باز کردن فایل

راه‌های مختلفی برای نوشتن داخل فایل‌ها وجود دارد. در اینجا، ما از انعطاف‌پذیرترین راه استفاده می‌کنیم. که برپایه جریان‌هایی (stream) ایجاد شده است که در یک ذخیره‌گاه (مثلاً یک فایل) قرار دارند. این روش همچنین امکان مرتب‌سازی مستقیم آبجکت‌ها را در یک فایل می‌دهد و در صورتی که بخواهید وضعیت یک بازی خیلی پیچیده را ذخیره کنید، خیلی کمک خواهد کرد.

ما از یک فایل ذخیره‌گاه مستقیم استفاده می‌کنیم که امکان تغییر اطلاعات در آن نیست. این موضوع مشکلی محسوب نمی‌شود، زیرا ما هر دفعه کل فایل را بازنویسی خواهیم کرد. کد زیر ذخیره‌گاه فایل را باز می‌کند:

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

توجه داشته باشید که ما از فایل سرور (file server) محیط CeikonEnv استفاده می‌کنیم. فایل سرور سیستمی است که نوشتن و خواندن فایل‌ها را کنترل می‌کند. برای کل سیستم فقط یک فایل سرور وجود دارد و شما نمی‌توانید برای خودتان یک فایل سرور شخصی ایجاد کنید. (فقط امکان اتصال به آن وجود دارد.) به علت اینکه این روش پردردسر است، ما از فایل‌سروری که سیستم برای خواندن فایل منبع ما استفاده می‌کند، به طور مکرر استفاده خواهیم کرد.

ایجاد یک جریان

یک ذخیره‌گاه، می‌تواند حاوی جریان‌های (stream) مختلفی باشد؛ یکی از آنها باید جریان ریشه (root stream) باشد. برای ذخیره بالاترین امتیاز، فقط یک جریان نیاز است. جریان‌های چندگانه می‌توانند برای ذخیره اطلاعات بازیکنان مختلف استفاده شوند.

جریان ایجاد می‌شود و ID خودش را برمی‌گرداند، که بعداً وقتی که می‌خواهیم این جریان را به عنوان جریان ریشه ذخیره‌گاه تنظیم کنیم، از این ID استفاده خواهیم کرد.

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

نوشتن اطلاعات

بالاخره ما به نقطه‌ای رسیدیم که اطلاعات به فایل نوشته می‌شوند. ما دو مقدار صحیح به جریان خواهیم نوشت. در سیستم عامل سیمبین، یک TInt از نوع یک متغیر ۳۲ بیتی تعریف شده است، بنابراین ما از تابع WriteInt32L() که عضو جریان است، برای نوشتن اطلاعات استفاده خواهیم کرد.

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

نوشتن ورژن فایل به جریان اجباری نیست. اگرچه می‌تواند بسیار پرکاربرد و مفید باشد؛ مثلاً فرض کنیم خریدار شما بازی Mopoid شما را روی موبایلش نصب کرده است. بعد از مدتی شما یک ویرایش جدید منتشر می‌کنید که نام بازیکن را هم ذخیره می‌کند. خریدار شما، آن را دانلود می‌کند و بازی قبلی موبایلش را ارتقا می‌کند. فایل اطلاعاتی قدیمی پاک نخواهد شد!

حالا کاربر بازی ورژن جدید شما را شروع می‌کند، این بازی فایل اطلاعاتی قدیمی را پیدا می‌کند و سعی می‌کند آن را لود کند و انتظار دارد که بیشترین امتیاز و نام کاربر را دریافت کند. ولی، به علت این که این فایل توسط بازی قدیمی ایجاد شده است، حاوی نام بازیکن نیست و این موضوع نیز مشکل‌ساز خواهد شد. به راحتی می‌توانیم با بررسی ورژن فایل، این مشکل را رفع کنیم.

بستن فایل

شما تقریباً کار را تمام کرده‌اید. فقط اطمینان حاصل کنید که اطلاعات به فایل نوشته شده‌اند و سپس فایل را ببندید:

```
// 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
```

پشته Cleanup

در تابعی که اخیراً نوشتیم، از پشته Cleanup سیستم عامل سیمبین استفاده شده است که یکی از مهمترین قسمت‌های برنامه‌نویسی C++ سیستم عامل سیمبین است. فهم کامل آن مشکل است و البته در بعضی از نوشتجات دیگر توضیح داده شده است، بنابراین در اینجا فقط به طور خلاصه توضیح می‌دهیم.

فرض کنیم شما در یک متد، یک آبجکت در یک گروه ایجاد کرده‌اید و متغیر شما به آن اشاره می‌کند. اگر اجرای بعضی از توابع، قبل از تمام شدن متد، ناتمام رها شود، متد شما فوراً متوقف خواهد شد و برنامه، متغیرهای اتوماتیک را پاک خواهد کرد.

به هر حال در این مورد، شما فقط اشاره‌گر به آن آبجکت را دارید که از بین خواهد رفت ولی آبجکت خودش در حافظه بدون ارجاع باقی خواهد ماند و این بد است.

بنابراین، آبجکت‌هایی که توسط متغیرهای اتوماتیک استفاده می‌شوند، باید به پشته Cleanup برده شوند. اگر اجرای تابع رها شود، برنامه به طور اتوماتیک تمام آبجکت‌های موجود در پشته Cleanup را پاک خواهد کرد.

طبیعتاً شما از این به بعد از پشته Cleanup به صورت زیر استفاده خواهید کرد.

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

در مثال فایل، ما دو تابع صدا کردیم که دارای یک C در انتهای نامشان بودند. به این معنی که آنها یک آبجکت در پشته Cleanup دارند که باید هنگامی که دیگر به آنها نیاز نیست، پاک شوند. بنابراین از دو PopAndDestroy() استفاده کردیم. لطفاً توجه داشته باشید که هر وقت که برنامه پشته Cleanup را به خاطر کامل اجرا نشدن تابع، پاک می‌کند، شما باید خودتان توسط اجرای یک برنامه error-free مواظب باشید که عملیات پاک کردن به درستی انجام شود.

لود کردن فایل

ذخیره کردن یک فایل، فقط نصف داستان است. خوشبختانه نحوه لود کردن نیز شبیه همان است. ابتدا نام فایل را همان طور که قبلاً مشاهده کردید، کامل کنید. سپس ذخیره‌گاه باید دوباره (این دفعه به منظور خواندن) باز شود.

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

سپس، باز کردن جریان ریشه ...

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

... می‌توانیم اطلاعات را بخوانیم.

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

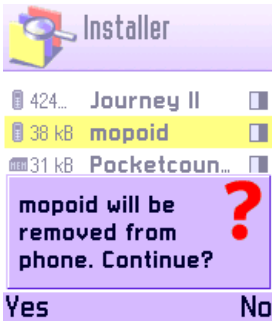
شما می‌توانید مدیریت خطای ورژن را بهبود ببخشید، ولی برای اولین نسخه از بازی، کافیست. در اینجا ما فقط error پیدا نشدن را می‌دهیم به این معنی که قادر به پیدا کردن اطلاعات تنظیمات صحیح در فایل نیستیم. صدا زدن این تابع (در ConstructL() از موتور بازی)، استثنائات تابع لود را پیدا می‌کند و آنها را نادیده می‌گیرد. که حتی در صورتی که تاکنون فایلی ساخته شده نباشد نیز اتفاق خواهد افتاد. در مورد error پیدا نشدن فایل، بازی به سادگی مقدار بالاترین امتیاز را مقدار پیش‌فرض که همان صفر است، در نظر می‌گیرد.

فراموش نکنید که بعد از لود کردن تمام اطلاعاتتان، پشته را پاک کنید!

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

پاک کردن اطلاعات، هنگامی که برنامه شما پاک شد

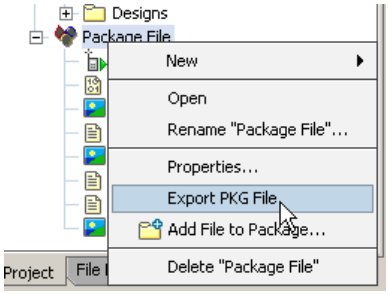
ممکن است کاربر تصمیم بگیرد که بازی را پاک (Uninstall) کند. تمام فایل‌های شما باید از موبایل پاک شوند. (این مورد البته یکی از ضروریاتی است که باید کامل شوند تا برنامه شما نشان "Symbian Signed" داشته باشد.) UnInstall کردن توسط یک مدیر برنامه انجام می‌شود و برنامه شما هنگام UnInstall شدن، اجرا نخواهد شد. بنابراین شما باید بعد از اینکه مشخص کردید برنامه‌تان چطور باید نصب شود، مشخص کنید که چه فایل‌هایی باید پاک شوند.



تصویر۲۸: برنامه Uninstall باید فایل‌های اطلاعاتی را به درستی پاک کند.

متأسفانه C++BuilderX در تعریف کردن فایل‌های قابل نصب، کاملاً محدود است و انجام این کار را پیچیده می‌کند. و شما این کار را خودتان و بدون C++BuilderX انجام خواهید داد. همانطور که مراحل قبلی انجام دهید، به قسمت ساخت برای دستگاه (ARMI / UREL) بروید. روی Package File راست‌کلیک کنید و Export PKG File را برگزینید. این کار یک فایل کوچک با نام Mopoid.pkg در پوشه

/dev/Mopoid/group می‌سازد.



تصویر۲۹: صدور فایل پکیج

حالا این فایل را با یک ویرایشگر متن باز کنید. باید شبیه زیر باشد.

```
&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\ball.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\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\panel.png"
```

این فایل مشخص می‌کند که فایل نصب، فقط حاوی یک زبان (انگلیسی) است. دومین خط حاوی نام برنامه است که در هنگام نصب نمایش داده خواهد شد. UID آن، همان است که هنگام ایجاد پروژه به آن دادید. و نیز در انتهای دومین خط، ورژن برنامه آمده است.

سومین خط، نشان می‌دهد که برنامه شما می‌تواند در تمام دستگاه‌های سری ۶۰ حتی در Nokia 7650 (که از ویرایش SDK 0.9 سری ۶۰ استفاده می‌کند) نصب شود. اگر شما برنامه‌ای می‌نویسید که از دستگاه‌های قدیمی پشتیبانی نمی‌کند، باید از یک Product ID سری ۶۰ متفاوت استفاده کنید.

فایل پکیج مشخص می‌کند که هم برنامه و هم فایل منبع (که حاوی رشته‌ها و تعاریف منوها است)، به دستگاه و داخل پوشه \system\apps\Mopoid کپی خواهند شد. علامت ! در ابتدای آن به این معنی است که کاربر می‌تواند انتخاب کند که در کدام درایو می‌خواهد برنامه را نصب کند. حافظه داخلی و قابل نوشتن دستگاه‌های سری ۶۰، درایو C است و MultiMediaCard نیز دارای حرف E است.

خب، برنامه ما هنگام اجرا شدن، یک فایل جدید ایجاد خواهد کرد. بنابراین این فایل نیاز ندارد که از قبل نصب شود، ولی باید هنگام Uninstall شدن بازی پاک شود. چگونه این کار را انجام می‌دهد؟ خط زیر، کاری که ما می‌خواهیم را انجام می‌دهد:

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

FN به معنی FileNull است. هلپ SDK یک تعریف خوب از معنی این خط دارد.

«فایلی که هنوز وجود ندارد، در فایل sis هم ضمیمه نشده است. این فایل توسط برنامه در حال اجرا ایجاد می‌شود و هنگامی که برنامه پاک شد، حذف می‌شود. نام نسبت داده شده به سورس‌فایل مهم نیست و می‌تواند خالی هم باشد، (به صورت "") توجه داشته باشید که بعضی فایل‌ها هنگام آپگرید کردن برنامه به یک نسخه بالاتر، حذف نمی‌شوند. پس شما خیالتان راحت باشد فایل‌هایی مانند فایل‌های ini. که اطلاعات برنامه را نگهداری می‌کنند، هنگام آپگرید شدن از دست نخواهند رفت.»

ساخت برنامه از طریق خط فرمان DOS

متأسفانه، C++BuilderX مواظب فایل پکیجی که ایجاد کرده است، نیست. بنابراین از حالا باید فایل نصب (.sis) را از طریق خط فرمان نصب کنیم. ابتدا اطمینان حاصل کنید که IDE شما هنوز هم روی کامپایل شدن برای ARMI / UREL تنظیم شده است (چرخ‌دنده‌های آبی). پروژه را از طریق Project → Make Project 'Mopoid.cbx' کامپایل کنید. حالا، پنجره خط فرمان را باز کنید (Start → Run… → cmd) و در پنجره خط فرمان به پوشه C:\Symbian\dev\Mopoid\group بروید. برای انجام سریعتر این کار می‌توانید نرم‌افزار Command Window Here مایکروسافت را نصب کنید. دستور زیر را وارد کنید:

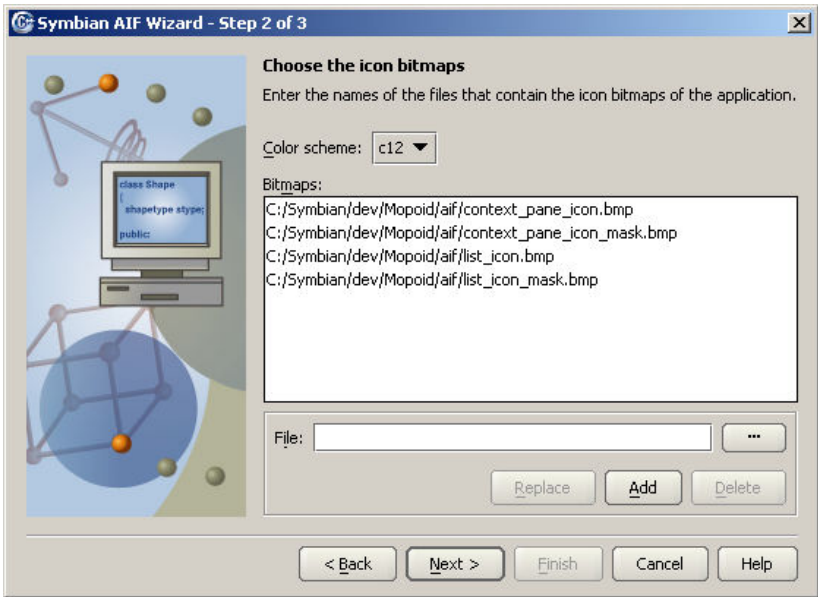
```
makesis Mopoid.pkg
```

اکنون باید فایل Mopoid.sis در همان پوشه موجود باشد. آن را روی موبایلتان امتحان کنید. می‌توانید برای راحتی کار و این که همیشه این دستور را انجام ندهید، یک فایل batch بسازید که حاوی دستور باشد و هر دفعه فقط آن را اجرا کنید.

هنگامی که کارتان تمام شد، فراموش نکنید که IDE را به کامپایل کردن برای شبیه‌ساز ویندوز تنظیم کنید. (WINS / UDEB)

گام ۱۴ : تنظیم آیکن برنامه

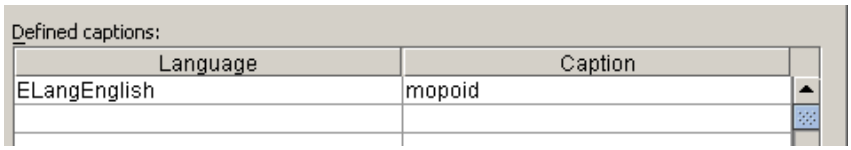
بعضی اطلاعات، مانند آیکن و عنوان برنامه، در یک فایل AIF. (Application Information File: فایل اطلاعات برنامه) ذخیره شده‌اند. در این مرحله، ما این فایل را برای پروژه‌مان ایجاد خواهیم کرد. به File → New… بروید. Mobile C++ را انتخاب کنید و New Symbian AIF Wizard را انتخاب کنید و روی OK کلیک کنید. مقادیر پیش‌فرضی که در مرحله اول نشان داده می‌شوند را قبول کنید. مرحله دوم مقداری جالبتر است، اکنون شما آیکن‌هایی را در موبایلتان که در برنامه Menu ظاهر خواهند شد، اضافه خواهید کرد.



تصویر۳۰: اضافه کردن آیکن‌ها به برنامه

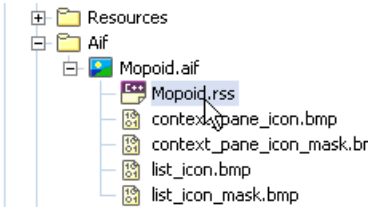
تصاویر موجود در پوشه aif پروژه را به همان ترتیبی که در تصویر ۳۰ مشاهده می‌کنید، اضافه کنید. همیشه فایل مربوط به آیکن اصلی را اول اضافه کنید. فراموش نکنید که بعد از انتخاب کردن یک فایل، دکمه Add را بزنید!

هنگامی که اضافه کردن هر چهار فایل را انجام دادید، به مرحله سوم بروید. اکنون شما عنوان بازی را اضافه خواهید کرد. به علت اینکه Mopoid فقط انگلیسی را پشتیبانی می‌کند، کلمه Mopoid را بنویسید و زبان را بگذارید همان ELangEnglish باقی بماند. روی Add کلیک کنید.



تصویر۳۱: عنوان انگلیسی، اکنون تعریف شده است

بعد از کلیک کردن روی Finish، همه چیز تمام خواهد شد. البته یک باگ عجیب در نسخه کنونی C++BuilderX وجود دارد و آن است که اگر شما برنامه را کامپایل کنید، هیچ آیکنی نخواهید دید. برای درست کردن این اشکال، در پانل project روی Mopoid.rss دابل‌کلیک کنید (در پوشه aif است) تا باز شود.



تصویر۳۲: باز کردن فایل منبع برای درست کردن یک باگ C++BuilderX

خطی که حاوی num_icons است را بیابید و مقدارش را از ۴ به ۲ تغییر دهید:

```
num_icons = 2;
```

ما فقط دو آیکن در ابعاد متفاوت و ماسک‌های مربوطه‌شان را اضافه کردیم. C++BuilderX در حقیقت باید این را بداند، ولی ظاهراً انجام نمی‌دهد.

گام ۱۵ : مدیریت وضعیت بودن برنامه در پس‌زمینه

سیستم عامل سیمبین، یک سیستم عامل چندوظیفه‌ای است. هنگامی که برنامه شما در حال اجرا است، رویدادهای متفاوتی می‌توانند برنامه شما را به پس‌زمینه بفرستند و یا یک پیام روی برنامه نشان دهند. مثال‌ها: زنگ زدن موبایل و یا آمدن یک پیام، اخطار کمبود باتری، زده شدن دکمه قرمز توسط کاربر و غیره.

تصویر۳۳: هنگامی که بازی به پس‌زمینه فرستاده می‌شود، باید متوقف شود.

حتی برای بازی کوچکی مانند Mopoid هم خیلی مهم است که این رویداد را کنترل کند. این بازی باید تا هنگامی که کاربر قادر به بازی کردن نیست، مکث کند (pause). همچنین، باید منابع سیستم را با متوقف کردن آپدیت صفحه متوالی، مقداری آزاد کند. این وضعیت همچنین یک وضعیت

خوب برای ذخیره کردن پیشرفت بازی است.

سیستم عامل سیمبین هنگامی که برنامه‌تان به پس‌زمینه فرستاده می‌شود، یک رویداد به برنامه شما می‌فرستد (تابع HandleForegroundEventL() صدا زده می‌شود.)

فایل MopoidView.cpp را باز کنید و به فایل header آن بروید. توجه داشته باشید که کلاس CMopoidView از CAknView مشتق شده است که تابع HandleForegroundEventL() را تعریف می‌کند. به علت اینکه این تابع را خواهیم نوشت، تعریف زیر را به انتهای تعاریف توابع عمومی (Public) اضافه کنید:

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

پارامتر aForeground مشخص می‌کند که آیا برنامه ما به پس‌زمینه رفته است، و یا اینکه دوباره فوکوس را به دست آورده است. به MopoidView.cpp برگردید و تابع زیر را به انتهای فایل اضافه کنید:

```
// 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);
}
```

همانطور که می‌بینید با توجه به اینکه برنامه فوکوس را دریافت کند، یا از دست دهد، یک متغیر کنترل وضعیت از کلاس موتور بازی، متناسب با وضعیت تغییر خواهد کرد. توجه داشته باشید که بهتر نیست که بعد از دریافت فوکوس، بازی را ادامه (resume) دهید، بلکه بهتر است به کاربر اجازه دهید، هر وقت آماده بود، بازی را شروع کند.

گام ۱۶ : رویدادهای متناوب

آپدیت بازی، خودش با استفاده از یک Avtive Object انجام می‌شود. کلاس نظیر آن در فایل UpdateAO.cpp قرار دارد. به این معنی که بازی نیاز ندارد که یک فرکانس (سرعت انیمیشن یا تعداد تصاویر نمایش داده شده در واحد زمان) ثابتی داشته باشد، بلکه زمان بین فریم‌ها را اندازه می‌گیرد تا بفهمد توپ و پانل از آخرین فریم چقدر حرکت کرده‌اند.

و آن نیز به این معنی است که تمام مقادیر توسط اعداد اعشاری به طور داخلی محاسبه می‌شوند. متأسفانه پردازنده‌های موبایل‌ها، مقادیر اعشاری را پشتیبانی نمی‌کنند، بنابراین تمام کارها در نرم‌افزار شبیه‌سازی می‌شوند که این موضوع، محاسبات را خیلی کندتر از محاسبات اعداد صحیح می‌کند.

برای بسیاری از بازی‌ها، استفاده کردن از یک فرکانس ثابت، کافی خواهد بود که به طور متوالی توسط یک تایمر صدا زده می‌شوند. در این صورت

حرکات می‌توانند اعداد ثابت صحیح باشند، و نیازی به محاسبات دقیق نخواهد بود.

روشن نگه داشتن صفحه

Mopoid هنوز از یک تایمر ثابت استفاده می‌کند (که در حقیقت یک آبجکت فعال است که توسط سیستم عامل سیمبین تدارک دیده شده است). این تایمر به منظور کاهش امتیاز بازیکن هر پنج ثانیه یکبار، به کار می‌رود و در نتیجه، بازی را رقابتی‌تر می‌کند. موضوع دیگر این است که اگر در مدت زمان مشخصی، کلیدی فشرده نشود، سیستم، صفحه را خاموش می‌کند. این موضوع هنگامی که کاربر هنوز در حال بازی است و منتظر این است که توپ دوباره پایین بیاید، بد خواهد بود. بنابراین ما باید تایمر عدم فعالیت گوشی را هر چند ثانیه یکبار، ریست کنیم. این کار توسط اجرای خط زیر که شما باید به تابع ()DoCallBack در کلاس موتور بازی اضافه کنید، انجام می‌شود:

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

استفاده کردن از یک تایمر

تنها چیزی که الان باید انجام دهیم، این است که تایمر را فعال کنیم، بنابراین این تابع، همیشه صدا زده خواهد شد. مقداری به بالا اسکرال کنید (بروید) که در آن صورت تابع ()StartTimerL از کلاس موتور بازی را خواهید یافت. آبجکت تایمر، از قبل به صورت یک متغیر خصوصی در کلاس موتور بازی تعریف شده است:

```
CPeriodic* iPeriodicTimer;
```

بنابراین در تابعی که در بالا ذکر شد، ما باید آبجکت تایمر را ایجاد کنیم و آن را شروع کنیم:

```
iPeriodicTimer = CPeriodic::NewL(CActive::EPriorityLow);
iPeriodicTimer->Start(KTickInterval, KTickInterval, TCallBack(TimerCallBack, this));
```

توجه داشته باشید که ما از تابع ()NewL استاتیک از آبجکت CPeriodic که سیستم عامل سیمبین تدارک دیده است، استفاده می‌کنیم. ما این کار را انجام می‌دهیم زیرا iPeriodicTimer یک متغیر عضو است، بنابراین، آبجکت به پشته Cleanup نخواهد رفت اگر یکی از قسمت‌ها کار نکند، آبجکت، توسط مخرب کلاس موتور بازی حذف خواهد شد.

در خط بعدی ما تایمر را شروع می‌کنیم و به آن می‌گوییم که هر ۵ ثانیه یکبار، دستورات را اجرا کند (KtickInterval به عدد ۵۰۰۰ تنظیم شده است). آخرین پارامتر، تابع اجرایی را مشخص می‌کند که تایمر هر ۵ ثانیه یکبار باید آن را اجرا کند. نامش ()TimerCallBack است و قسمتی از کلاس موتور بازی است (توسط this ارجاع شده است). این تابع، قبلاً توسط کدهای نمونه، تعریف شده است.

متوقف کردن تایمر

هنگامی که بازی مکث کرد و یا پایان یافت، تایمر باید متوقف شود (و حذف شود). این کار کاملاً ساده است. کد زیر را در تابع ()StopTimer بنویسید:

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

برای متوقف کردن آن، تابع کنسل کردن تایمر صدا زده شده است. سپس آبجکت پاک خواهد شد و به NULL تنظیم خواهد شد که در این صورت، هر کسی می‌فهمد که تایمر دیگر وجود ندارد. اگر ما دوباره به آن نیاز پیدا کنیم، فقط کافی است که دوباره آن را ایجاد کنیم.

گام ۱۷ : نکات پایانی

بازی تمام شده است! اگر بخواهید، می‌توانید به قسمت‌های دیگر سورس‌کد که جزو این خودآموز نیستند، نگاهی داشته باشید. برای مثال، روتین‌های اجرای صداها، واقعاً جالب هستند. این بازی همچنین از یک فایل «مرحله» خارجی استفاده می‌کند. شما می‌توانید آن را توسط یک ویرایشگر متن معمولی ویرایش کنید و مراحل شخصی خودتان را به راحتی بسازید. این فایل توسط برنامه خوانده می‌شود و مراحل ساخته می‌شوند.

در مورد نویسنده

این خودآموز توسط Andreas Jakl نوشته شده است. او بانی Mopius است که یک کمپانی است که محصولات مخصوص موبایل جدیدی تولید می‌کند و به کاربران اجازه می‌دهد که چیزهایی را تجربه کنند که تا به حال ندیده‌اند.

بازی The Journey یکی از اولین بازی‌های ماجرایی مخصوص موبایل در جهان است. این بازی به صورت یک بازی open source منتشر شده است و تاکنون بیش از ۱۰۰۰۰ مرتبه دانلود شده است. ویرایش بعدی آن، یعنی The Journey II همان مفهوم قبلی را ادامه می‌دهد و یک دنیای مجازی دلربا و بزرگ ایجاد می‌کند که می‌تواند توسط بازیکن جستجو کشف شود. این دو بازی تاکنون چندین جایزه برده‌اند، مثل Most Innovative Game (Mobile Fun Awards) و همچنین یک جایزه از Austrain State Price. این بازی همچنین در تلویزیون نیز نشان داده شده است.

می‌توانید دموی بازی را از سایت کمپانی دانلود کنید.

(توضیح مترجم: البته من که از این بازی هیچ سر در نیاوردم!)

تماس

برای دریافت اطلاعات بیشتر و یا تماس با نویسنده می‌توانید با ایمیل contact@mopius.com تماس بگیرید و یا به آدرس http://www.mopius.com مراجعه کنید.

گام ۱۸ : تمارین

اگر می‌خواهید تعدادی از وظایف را خودتان انجام دهید و قسمت‌هایی به اضافه کنید، در زیر پیشنهاداتی هم می‌دهیم که چگونه بازی را ارتقا دهید:

مدیریت کلیدهای دیگر

موبایل‌های سری ۶۰ دارای یک جوی‌استیک هستند که برای کنترل اشاره‌گرها به کار می‌رود. بعضی کاربرها ممکن است ترجیح دهند که پانل را از طریق اعداد کنترل کنند. کدهایی را اضافه کنید که اجازه کنترل پانل از طریق دکمه‌های 4 (برای چپ) و 6 (برای راست) را به کاربر بدهد. شما می‌توانید آنها را در همان عبارت switch که برای حرکت‌ها ایجاد کردیم، مدیریت کنید. راهنمایی: کلیدهای اعداد را با چیزی شبیه "case '4':" کنترل کنید.

تعریف مراحل زیاد

بازی نمونه فقط حاوی ۵ مرحله است که برای امتحان کردن کافی هستند ولی می‌توانند بیشتر از این باشند. به فایل levels.dat نگاه کنید تا ببینید که مراحل چگونه تعریف شده‌اند و چه آجرهایی توسط اعداد تعریف می‌شوند و نیز چند خط یک مرحله را می‌سازند. سپس مراحل جدیدی را

اضافه کنید و بازی را طوری تنظیم کنید که افزوده‌های شما را بشناسد. به یاد
را اجرا کنید تا این فایل به پوشه صحیح کپی شود تا شبیه‌ساز بتواند آن را پیدا

ذخیره کردن مقدار پیشرفت بازی

وقتی که بازی بزرگتر شد، مهم است که کاربر بتواند ادامه بازی را بعداً
دست آورده است را ذخیره کند. شما باید امتیاز را در یک متغیر عضو جدید ذخ

برای ذخیره کردن آن، فایل اطلاعاتی را توسط یک فلگ که نشان می‌د
شدن به مرحله داشت و مرحله‌ای که اکنون بازی می‌کند، بسط دهید.

هنگامی که کاربر بازی را ترک می‌کند و بازی هنوز فعال است، به فلگ
شروع می‌شود، آن مقادیر باید برگردانده شوند.