Creating Games using J2ME

With the advent of Java 2 Micro Edition (J2ME) on mobile phones, it is now possible to create mobile games that aren't all that crappy!

Now, don't get me wrong. Some creative folks have devised truly interesting games using the Wireless Application Protocol (WAP). But that's sort of like building a skyscraper out of Popsicle sticks-in the end the simple text-pushing protocol just won't cut muster if you want responsive, graphically rich games. Gladiator, one of the most popular WAP games, is really nothing more than a rock-paper-scissors clone!
For several years now, phone manufacturers have had native hardware capable of supporting decent games. Nokia's *Snake* game is probably the most notable example. But mobile phone developer's kits are often proprietary. And downloading, installing, sharing, and porting these games is an all-but impossible task.

J2ME Overview

The Java 2 Micro Edition (J2ME) is a true-to-its-word subset of the standard Java libraries. J2ME consists of:

- The Kilobyte Virtual Machine (KVM). This resides on the handheld device and actually runs the bytecode you write.
- Configurations. These are the core classes that must be implemented for a particular type of device. For example, the Connected, Limited Device Configuration (CLDC) is meant for any device that has limited memory, screen size, and input abilities, but that is connected to a network. This includes most mobile phones. Companies like Sun, 3Com, Bull, Ericsson, Matsushita, Mitsubishi, Motorola, Nokia, NTT DoCoMo, and Siemens helped hammer out this standard.
- Profiles. These are Java libraries for specific brands of devices. All the user interface, programming and event models, networking, and error handling is implemented in one of these. For example, the Mobile Information Device Profile (MIDP) was developed using Sun's Community Process by companies such as AOL, Palm, Nokia, Sony, NEC, Motorola, Alcatel, Ericcson, NTT, DoCoMo, Psion, and Siemens. There are different profiles for different device families, with one currently being spec'ed out for the Palm.

While various J2ME profiles can run on anything from advanced set-top boxes to tiny chips on smart credit cards, for the sake of this article we will focus MIDlets-applets in the CLDC configuration written to the MIDP profile. Note that you can run MIDlets on your Palm as well. Check out this page, which has full instructions on converting MIDlet packages and deploying them on the Palm.

Games on Phones?

Mobile phones are a bold new platform for gaming. Today, there are more than 600 million mobile-phone users worldwide. Mobile phone users generally tend to be affluent, educated, and they often have lots of time on their hands. In the United States, for example, people spend 50-percent more time commuting than any other country (Yankee Group). This is the perfect time to pull out a mobile phone and play some quick games.

In the near future, we will likely see micro devices become even smaller and

specialized. Phones the size of earplugs, voice-activated assistants on wristwatches, and smart chips on credit cards are all becoming a reality.

Unsurprisingly, games are keeping up and even helping to lead this paradigm. While it may seem silly to try to achieve a rich, meaningful immersion on a tiny 100x100 pixel screen, there's one thing mobile phone games give you that even the best consoles can't provide: They're always with you, and can be played anywhere you go. This not only means that games can now be more convenient, but wholly new types of games can be designed that take advantage of new lifestyles.

The Joy of Java

Most every major mobile phone and handheld device manufacturer immediately realized the potential of J2ME: If Java were to be placed on the gadget, then hundreds of thousands of developers would immediately be able to create applications and add value. Furthermore, because it's Java, a program written for one device would be able to run on another device with little or no modifications. That certainly makes more sense that trying to force developers to learn a native language and API in order to create programs for your phone.
And so, most every major mobile phone manufacturer joined with Sun to create something called the CLDC: The Connected, Limited Device Configuration, along with the MIDP: The Mobile Information Device Profile. Mobile phone manufacturers have definitely embraced Java in a way that not even PC manufacturers and browser makers have. Java is clearly the future platform of choice for mobile devices, and an ideal platform for mobile games.

The Bad News
Programming games for handheld devices is a total pain. The screen is miniscule—160 by 160 on the Palm, and much less on mobile phones. The memory is limited—the Palm only gives you 2 to 8 Meg, depending on the model. You have little dynamic memory to play around with—most mobile phones only give you 32K, the Palm has a 96K heap limit. There's limited network connectivity, with a teeny, tiny 9.6 kilobit per second bandwidth. And the processor itself is often hundreds of times slower than your average desktop.

Programming in a language like Java only adds a layer onto this hard-to-digest cake. Sun's Kilobyte Virtual Machine (KVM) gets its name because it takes up a few kilobytes of space. But even those few bytes are wasteful. The KVM also usurps a fair bit of memory. A native application is the size and power it is, asking for nothing more. In addition, Java classes have the added overhead of a slow startup. While startup is going on, the KVM is going through the class, allocating heap space and verifying the bytecode.

Additionally, a Java programmer is restricted to the capabilities of the virtual machine. Because Java must work on a wide variety of devices, it is written for the lowest common denominator.

For example, here are some big stumbling blocks:

- No transparent images. Without transparency, overlapping sprites look really prickly. Any image that may overlap another image, or a background element will need to be rectangular in order to look good!
- You cannot grab, copy, or edit the pixels of RGB images on the fly. This means that ultra-cool graphic effects like fading in, explosions, and dynamic shadows are impossible.

- There is no fill-polygon or fill-triangle method, which makes rendering 3D images quite difficult.
- You cannot copy raw pixel data to the screen (blit). This makes is pretty much unfeasible to do any texture-mapping or particle effects, etc.
- There is no audio at all. I'll say that again: There is no freakin' audio at all!
- There is no floating-point math. This makes some 3D and physics pretty difficult.
- There is no native support or Java Native Interface (JNI). That means you can't dial the phone, work with any of the ringtones, work with any native user interface widgets, etc.

Additionally, the Anfyteam in Italy has put together a detailed list of gaming and graphics-related stuff that MIDP is missing.

Due to all the restrictions on a small device, there is no way of fitting in a just-in-time compiler (JIT). This means code will run quite slowly. In fact, while playing around with some casual tests, a Java 2 Micro Edition (J2ME) application runs about three to eight times slower than a native Palm application written in a compiled language such as C.

If you want to get some idea of the graphics speed of J2ME, the Anfyteam has created a great benchmark called Amark.

So is J2ME like war—good for absolutely nothing?

The Good News
First, some very good news: Java-enabled mobile phones embed the KVM right on the chipset. That means there are no memory hits, quick startup times, and quicker processing times.
Another interesting development is that several brands of Java phones offer neat extension APIs that let you access special, native features. Motorola and Nokia, for example, have announced game APIs that allow for audio, animations, transparency, and better graphics.
Additionally, it's important to remember that Java is perhaps the easiest modern language to develop in. With garbage collection of old objects and lack of memory allocation and pointers, it's a great way to create quick prototypes and develop them out into full apps. The object-oriented nature of Java makes it easy to maintain, making sweeping code changes with a little modification to a superclass.

Writing a native handheld application requires countless hours of debugging, emulating, testing, deploying, redebugging, and packaging. Write to a wrong memory location, and you can easily fry the machine. Waste a little memory on a desktop or server application, and nobody will bat an eye. But when dealing with a tiny space like the Palm, every bad byte hurts. Memory leaks in Java are possible, but they are much easier to avoid.

Perhaps the most compelling argument for using Java is that - you guessed it - you can write once and run every-damn-where. Theoretically, an application written for a Motorola cell-phone can also run on Nokia, an Ericsson, a Siemens, or a Palm. The same code could even be compiled and run as an applet in a Web browser, as an application on a million-dollar server machine, in your car's dashboard, or—eventually—in a Java-powered neural link to your own brain!

Creating a Midlet

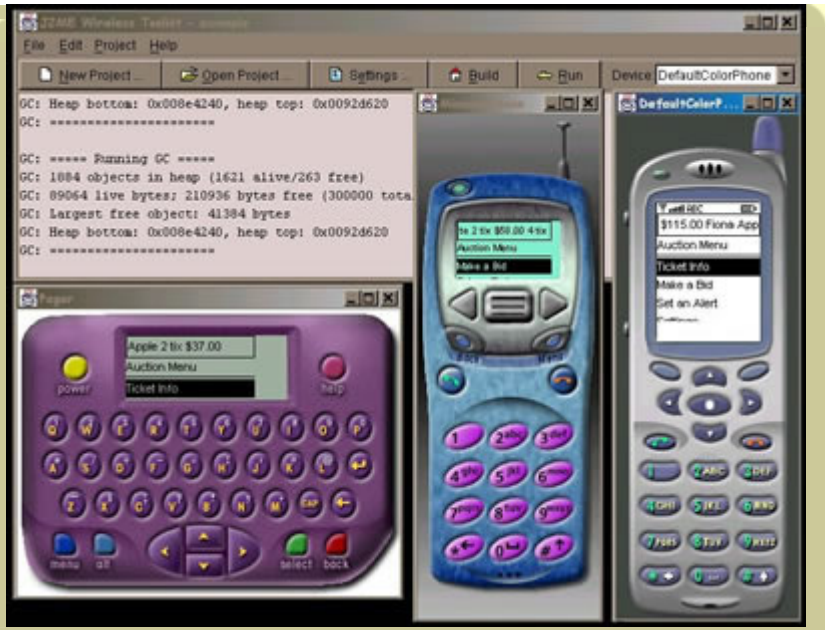
If you have any experience creating Java applications or applets, then programming in J2ME won't seem like such a stretch. The steps are basically the same:

1. Write your program
2. Compile it
3. Preverify it
4. Package It
5. Test It
6. Debug It
7. Release it!

The only thing that should set off your mental alarms is step number three—preverification. This may sound weird and complicated, but is actually quite easy. The purpose of preverification, theoretically, is to go through your bytecode and set hints up so that the actual verification of bytecode on the Palm will happen much more quickly, saving you valuable startup time.

There are many development environments that do all the compiling, preverification, and packaging for you. Metrowerk's Code Warrior has a J2ME plug-in, and Borland's JBuilder has a Handheld Express add-on. In addition, Nokia, Siemens, RIM, Zucotto, and Motorola all offer special SDKs and IDEs for J2ME development. See the resources list at the end of this article for more info.

Sun's own J2ME Wireless Toolkit is free, complete, really easy, and available for Windows, Linux, and Solaris. It also comes with a bunch of source code, including sample games such as *Snake*, *Sokoban*, a Tile Sliding game, *Pong*, and *Star Cruiser*.



Sun's own J2ME Wireless Toolkit is free, complete, really easy, and available for Windows, Linux, and Solaris.

We'll be using the Wireless Toolkit throughout this article.

Note that in order to run the Wireless Toolkit, you'll need Java itself (JDK1.3 or better), which has all the engines and libraries necessary to compile code. If you

don't already have the JDK, you can grab it here. Be sure to install it per directions, with all the proper settings for classpaths and paths.

Write Your Program
Just like an applet is based on the java.applet.Applet calss, class, a MIDLet always extends javax.microedition.midlet.

A MIDlet handles basic event handling and can be thought of as the one and only "window" that your application runs within.

Enough talk. Let's look at some code. Let's pass up the wonderful cliché of a nice Hello World program and create a full single-player *Tic-Tac-Toe* game instead.

Writing the Midlet
The heart of every MIDlet is the `startApp()` method. It kicks the MIDlet into action and should set up all the components or other objects you are interested in. Every MIDlet must also contain a `pauseApp()` and `destroyApp()` methods, which are called if and when a user chooses to either pause or quit your program.

Graphically speaking, the actual screen on your mobile phone or other devices is a Display object. Every MIDlet has one and only one Display object, which you can access using `getDisplay(this)`.

The Display
A Display can consist of either a Screen object or a Canvas. Every Screen has an optional title (usually at the top) and Ticker (usually running along the bottom). Specific types of Screens include:

- List: A row of elements that a user may choose from, perfect for a menu.
- Alert: A "dialog box" that appears and shows some text and/or an Image to the user and then, after a set amount of time, disappears. This is perfect for important messages.
- Form: A screen for user input which may contain all or some of the following MIDP objects: ImageItem, StringItem, TextField, DateField, Gauge, or ChoiceGroup. A ChoiceGroup is equivalent to radio buttons or check boxes.
- TextBox: A box that allows the user to type in or edit some text.
  Using the ease and convenience of Screens, creating quick MIDlets is a snap.

Getting Graphical

But let's get real, here. Most game developers want to customize components, capture user input, and create more detailed graphics. So pretty much every game will need something called a Canvas.

A Canvas must define the `paint()` method. The Graphics object allows for simple renderings using familiar Java methods such as `drawArc()`, `drawImage()`, `drawLine()`, `drawRect()`, `drawString()`, `fillRect()`.

Note that images need to be in the PNG format, and color images are supported on phones that have color screens. You could create an image as follows:

```
Image house = Image.createImage("/house.png");
```

You should always create your own Canvas subclass. For example:

```
class SillyCanvas extends Canvas
{
     public void paint(Graphics g)
     {
        g.drawRect(1,1,getWidth()-2,getHeight()-2);
        g.setFont(Font.getFont(Font.FACE_MONOSPACE,
           Font.STYLE_PLAIN,Font.SIZE_SMALL));
        g.setColor(0);
        g.drawString("Hello World", getWidth() / 2, 0,
           Graphics.HCENTER | Graphics.TOP);
        g.drawImage(house, 10,30, g.TOP|g.LEFT);
     }
}
```

This Canvas contains a rectangle bordering the entire screen with the string "Hello World" at the top, horizontal center. A house image is also drawn at the (10,30) coordinate.
You could drop this canvas into the Display by creating it in your MIDlet:

```
private SillyCanvas aCanvas = new SillyCanvas();
```

And then, in the MIDlet's startApp() method or anywhere else that made sense, you could set the current Display to show the special canvas:

```
theDisplay.setCurrent(aCanvas);
```

Note that a mobile phone's Display can only show one Screen or Canvas at a time. A typical application starts with a List as a main menu of choices, branching into other types of Screens depending on the user's selection.

Animating
The typical way of animating with MIDP is to use double buffering:

1. Have your main Canvas class draw a special offscreen image.
2. Create a separate Thread class that contains that offscreen image instance. Draw any animations or other graphics to the offscreen image within the threaded class.
3. Call `repaint()` on the MIDlet's main Canvas. Use the `sleep()` method within an endless loop to achieve slower or faster frame rates.

Double buffering in this way guarantees that your animation will occur at the same rate no matter which phone your game is running on.

Note that many phones automatically double-buffer your graphics for you. You can check whether double buffering is supported within a Canvas using the `isDoubleBuffered()` method.

You may want to check the `isDoubleBuffered()` method and only create an offscreen Image if double buffering is not supported, otherwise you can leave it as null. You can then change the graphics context within your paint method appropriately:

```
if( offscreen != null ){
          g = offscreen.getGraphics();
     }
```

Rolling your own double buffering can make for smoother animations, but copying images can very slow and memory-intensive. Also, some systems repaint the screen slower than the image can be copied-these phones will show graphic flickering.

Be aware that the refresh rate of most mobile phone screens is much slower than you may be used to for PC game programming. Frame rates of 4 fps are common... if you're lucky.

Creating *Tic Tac Toe*

So let's create a quick and dirty *Tic Tac Toe* MIDlet would create a form and add it to the display using Display's `setCurrent()` method.

Our *TicTacToe* game will consist of a 3x3 array of characters. Each square in the grid will be assigned a number, corresponding to the position on the phone's keypad. When you hit a number, you put either an X or an O in the array, depending on whose turn it is. So we won't even use any graphics. We'll simply use the Canvas to draw the array, as a few rows of Strings. The game continues as so until somebody wins. **[Code Listing: *Tic Tac Toe*]**

If all goes well, you'll wind up with a fresh, happy `TicTacToe.class` file. To test the application out, Sun has been kind enough to release a nice emulator. Just hit the Build button in the Wireless Toolkit. This will create the necessary Java classes. Your classes will also be automatically preverified and packaged into JAR and JAD files.

You can now hit the Run button. You can choose which Device you want to emulate by playing with the Device pull-down menu.

***Tic Tac Toe* MIDlet running on the Motorola i85s**

If you happen to actually own an i85s phone, you can plug it into your PC and use the phone's linking software to install the MIDlet directly (the JAR and the JAD file in the \J2mewtk\apps\TicTacToe directory) into the phone's memory. This is similar to the way you might install an application on the Palm Pilot.

In the very near future, Motorola and other mobile phone manufacturers may make it easy for users to download MIDlets right to their phones, on demand.

Networking

Of course the real fun of having an application on a mobile phone is networking. The mobile games that are most likely to be successful will involve lots of multiplayer ability, bringing players together in ways that nobody has ever experienced before.

The MIDP specification makes it extremely easy to pass data back and forth. Many phones support Datagrams, though they are not guaranteed to work on every device. The most common and easy method of MIDP network communication, of

course, is using HTTP.
Every Motorola i85s phone, for example, has its own static IP address. Having one phone communicate with another is only a matter of simple peer-to-peer networking.

In addition, the phone can connect to any outside server machine. This server can be used as a gateway for pretty much any type of game traffic or other network communication imaginable. For instance, a gateway can be set up to access an entire database of sports scores, and then stream only the latest requested scores to a MIDlet.

To connect to a server, simply have the MIDlet use the Connector class:

```
Datagram dgram = dc.newDatagram(message, msglength,
     "datagram://www.myserver.com:9000");
dc.send(dgram);
dc.close();
```

The remote "server" could, of course, be another device. Just create an endless loop that listens to a port and waits for some data:

```
DatagramConnection dc = ( DatagramConnection)Connector.open
     ("datagram://:"+receiveport);
while (true)
{
     dgram = dc.newDatagram(dc.getMaximumLength());
     dc.receive(dgram);
     reply = new String(dgram.getData(), 0,dgram.getLength());
}
```

Since the Datagram protocol is standard, one could write a dastardly-simple server component in Java Standard Edition (or any other language), running on any PC. For instance:

```
DatagramSocket receiveSocket = null;
DatagramPacket receivePacket = new DatagramPacket(bytesReceived,
bytesReceived.length);
receiveSocket.receive(receivePacket);
```

A simple game can be written with relative ease. For instance, `NumberPick.java` is a MIDlet that allows a user to pick a number between 0 and 9. It then sends this number to a server (at a fictitious machine with the host name of myserver.com) Note that for the sake of testing, it's a good idea to set the server to 'localhost' - that way you can run the server and the client from the comfort of your desktop machine. **[Code Listing: *Number Picker*]**

Running on the myserver.com machine is NumberServer.java, which simply sits there and waits for a message on port 9000. If the number is 4, then the server sends back a simple one-byte "Y" message. Otherwise it sends back an "N". **[Code Listing: *Number Server*]**

Maybe not as compelling a game as *Unreal* or *Quake III*, but it's a start.

Final Tips

As you go forth into the world coding games more complicated and sophisticated than *TicTacToe* and *NumberPick*, be sure to remember that every byte counts! It's

exceptionally easy to run out of memory on these phones.

Try to avoid Hashtables and Vectors, and recycle any objects you no longer need. For example, instead of creating two buttons on two separate screens, try to merely change the label on an existing button.

Also, avoid using costly operations like string concatenations. Use a StringBuffer instead. As for interface design, remember your audience and the limitation of the device. Use few, large, simple components that require as few keypad presses as possible.

While your application is running, you can sniff out the memory using: `Runtime.getRuntime().freeMemory()` and `Runtime.getRuntime().totalMemory()`

Remember to strategically garbage collect whenever resources fall too low using: `Runtime.getRuntime().gc()`

Since the MIDlet classes only contain a basic set of graphic user interface controls, you might want to opt for a better library.

A site called Trantor in Germany offers a package known as kAWT. This is a lightweight version of Java's AWT specially tailored for J2ME. There are versions for the Palm, as well as for MIDP. It allows your MIDlets to use standard Java widgets such as Panels and Containers and makes the MIDlet code truly upwardly compatible with applets. The only caveat is that kAWT will suck away an additional 27K or so of memory.

Finally, I highly recommend you use a code packer or obfuscator to compress your bytecode as much as possible. A good obfuscator such as IBM's jax can make your final application as much as 30 percent smaller!

Resources
More info about J2ME:
java.sun.com/j2me/

More info about the MIDP specifically:
java.sun.com/products/midp/

For some great tutorials and sample programs, check out:
webdev.apl.jhu.edu/~rbe/kvm/

The Java Mobile site has lots of articles, tips, and sampleapplications:
www.javamobile.org/

The Micro Java Network has all sorts of great articles and forums. Here are the games:
www.microjava.com/downloads/games

Visit Bill Day's J2ME Archive with tons of sample applications, links to IDEs and SDKs, and anything else J2ME related. There is also lots of source code:
www.billday.com/j2me/