J2ME & Gaming

Version 0.5.6

By Jason Lam

http://www.jasonlam604.com/

Who	Date	Comment	Version
Jason Lam	Sept 1, 2003	Initial Creation	0.5.0
Jason Lam	Sept 18, 2003	All Chapters merged into on document	0.5.1
Jason Lam	October 04, 2003	• Added a small intro the JEMBlazer • Added Pause, Remove and Save section under Chapter X – Improving Usability, Chapter as yet to be properly numbered	0.5.2
Jason Lam	October 13, 2003	Completed Chapter 12	0.5.3
Jason Lam	November 29, 2003	Completed Chapter 18	0.5.4
Jason Lam	May 8, 2004	Completed Chapter 17	0.5.5
Jason Lam	June 30, 2004	Completed Chapter 13, and started chapter 14 and continued some of chapter 19. Please note some of the NPC movement code is credited to Jonathan Knudsen. New Artwork credited to Leeman Cheng and Carlo Casimiro	0.5.6

Who is this Book For?

This book is about programming with J2ME on wireless devices with focus on developing games. It is assumed you have some knowledge and programming experience with J2ME and J2SE. The book does not go into detail on topics like how to make high level GUI menu but does demonstrate what a game menu might look like. Nor will it explain in detail how to use the Record Management System (RMS), but will go over topics that use RMS such as high score and game settings. As well a knowledge and experience with threading will be an asset before proceeding with game development. However, the book will go over in detail the new game classes that are now include in the MIDP 2.0.

The book also serves as quick reference for Java programmers who are interested in game mobile game development. The intent of this book is to provide good introduction for experience game developers who developed games in other languages and platforms and now are interested in using J2ME to develop games.

Chapter 1 – Overview

Goal of this Book

Welcome to the world of mobile gaming. The first section (Chapter 1 to Chapter 5) will briefly introduce to you:

- What the mobile gaming industry is about
- What tools are available for you to get started in developing your own mobile game
- Constraints you will face in developing mobile games
- Game Design and Development Process considerations

Afterwards the remaining chapters will walk you through a basic game tutorial, albeit the game itself isn't ground breaking, it does serve the purpose of introducing what makes up a game and more specifically a mobile game using J2ME. As well aside from the game itself it goes through several other important areas of mobile development such as the interface design of a game menu.

Mobile Gaming

Time is Now

Mobile Gaming! It is just that, gaming on the go! You can now interactively play all your favorites from Pac-man, Bust-A-Move to more involved interactive role playing games anywhere you go. As technology improves the better the games become. However, the mobile industry both in gaming and applications is still in its infancy. Developers, manufacturers and carriers are all still looking for that one killer game or application that will revolutionize the mobile industry and ultimately drive millions of dollars in revenue. Yes, graphics are getting better and game play is getting better but there is yet to be a game that will capture gamers by the thousands if not millions. This is a good thing for us developers; it leaves us with plenty of opportunity to cash in before the wave passes us by.

Different from Traditional Gaming

There is a great opportunity for individuals like you because it is relatively a low cost low production task. It is like the old days of game development where one could hunker down in the basement and actually produce a complete game. You can compare the mobile stage that it is currently in, at the time of this writing as the Nintendo age, it has just recently started graduating from the Atari age. About year ago or more most games were re-makes of all the classic games and slowly the graphics have improved along with the gameplay, current games more resemble games you would find on NES or Super NES.

Mobile being in the early stages things such as graphics, artificial intelligence and multiplayer it requires a lot less resources. It doesn't take a team of 60 graphic artists, 20 developers and advance photo imaging equipment to produce one game over a span of a year or so. Game development for mobile devices can easily be developed by a small team of 1 to 5 people on a modest budget of anywhere from few thousand dollars to hundred thousand unlike PC gaming world where budgets usually starts off in the millions. Mobile devices are already enabled with the latest gaming fad the ability to communicate on a network, which naturally suggest the development of multiplayer games. As well phone manufactures, carriers or anyone else do not dictate what games you can make, this is unlike the PC and console gaming world where games are made based on several factors and decisions usually not made by the development team rather by other parties with revenue interest such as the marketing team. Most mobile gaming available tools and standards are openly available through the original creators, manufacturers and carriers. Once again unlike the traditional gaming world finding out information on how to program for XBox, PlayStation or any other console is much more difficult let alone programming for them.

A quick review of why mobile gaming is different from traditional gaming:

- Low Budget
- Small Development Team
- Technology is within reason and is manageable, no need for special equipment such photo imaging equipment
- Based on open standards
- Mobile phones by default are ready for multiplayer gaming

Development Tools

Some available platforms for mobile game development are:

- Java 2 Micro Edition ™ by Sun MicroSystems – http://wireless.java.sun.com
- MoPhun – http://www.mophun.com
- Brew – http://www.brew.com

This not to be confused with mobile operating systems such as PalmOS and SymbianOS, these operating systems themselves support environments like J2ME.

This book will focus on J2ME (Java 2 Micro Edition). With that being said there are several software development kits you can use. To help ensure your game will run on different manufacturers and different models of mobile devices you will mostly like be using more then one SDK. The main J2ME SDK is available from Sun Microsystems at http://wireless.java.sun.com. Other development kits are available at:

Who	Website
Nokia	http://forum.nokia.com
SonyEricsson	http://www.sonyericsson.com
Sony Clie	http://www.cliedeveloper.com/top.html
Motorola	http://www.motocoder.com

Nextel	http://developer.nextel.com
RIM	http://www.blackberry.net/developers/na/index.shtml
Siemens	http://www.siemens-mobile.com
SprintPCS	http://developer.sprintpcs.com/adp/index.do
Palm	http://www.palmsource.com/developers/
Symbian	http://www.symbian.com
BellMobility	http://www.developer.bellmobility.ca/

As well there are few IDEs available for J2ME development. Some of these IDEs provide full integration options with other development kits. For example Sun One Studio Mobile Edition can easily be integrated with SprintPC J2ME SDK, SDK from Sun, SDK from Nokia and SDK from Siemens.

Who	WebSite
Sun One Mobile Edition	http://wwws.sun.com/software/sundev/jde/
JBuilder Mobile Edition	http://www.borland.com
Code Warrior Wireless Studio	http://www.metrowerks.com/MW/Develop/Wireless/Wireless_Studio/default.htm
WhiteBoard SDK	http://www.zucotto.com/home/index.html
IBM Studio Device Developer formly know as IBMVisualAge MicroEdition	http://www-3.ibm.com/software/wireless/wsdd/

What Else is Out There?

Aside from the regular tools and IDEs provided by various software organizations, manufacturers and carriers there are other tools/software packages that may be of interest to you.

3D

Though presently most games are presented with only 2D graphics due the constraints of mobile devices there are higher end devices that are capable of 3D. As well as the technology improves 3D will be common as it is now on both gaming consoles and PC desktop.

There is JSR in progress that supports 3D graphics for J2ME. It is quite possible with the next release after MIDP 2.0 3D JSR 184 *Mobile 3D Graphics API for J2ME™* to find out more visit http://jcp.org/en/jsr/detail?id=184 :

As well the widely used OpenGL now as mobile version named OpenES with focus on providing 3D graphics for mobile handsets, for more information visit http://www.khronos.org/embeddedapi/index.html. The company FatHammer, http://www.fathammer.com, has integrated the OpenGL ES-accelerated OMAP platform into their X-Forge 3D Game Engine SDK.

There is also the MoPhun 3D engine available at http://www.mophun.com.

As you can see 3D is may not be quite here yet for mobile devices but it is sure on its way.

Beyond Stand-Alone Game

Aside from developing stand-alone games there are various other technologies that will enable you to develop multiplayer games.

By default with J2ME you are able to communicate over HTTP/HTTPS. Another option is to communicate using the Jini the surrogate J2ME architecture. As well if the carriers support socket communication you are able to implement socket networking between games. However, it takes a lot more then just the communication protocol, there needs to be a gaming server that can handle and provide various features, some of these features could be but not inclusive to the following:

- Game Lobby
- Game Rooms
- Track User interaction
- Authentication
- Chat Rooms
- Instant Messaging
- Monitoring and statistics Tools

The Game Room itself acts some like a mediator/traffic cop that relays information back and forth to the clients and to the respective game in play.

It is definitely a lot of work to implement this. However, there are other companies that save you the trouble with their already pre-made game servers:

- DemiVision –http://www.demivision.com (recently acquired by JamDat.com)
- Mforma™ – http://www.mforma.com
- Xadra – http://www.xadra.com
- Butterfly.net – http://www.butterfly.net
- TerraPlay – http://www.terraplya.com

Aside from producing games for wide area networks you can produce games for personal area networks, technology better known as Bluetooth. The idea is the same has a game over HTTP but within a confined local area. Some great links to Bluetooth and J2ME are:

- Zucotto Wireless – http://www.zucotto.com
- RococoSoft – http://www.rococosoft.com

Two other technologies that can bring mobile gaming a new twist is Peer to Peer famously advertised by the success of Napster. For more information on Peer to Peer using Java visit http://www.jxta.org and http://jxme.jxta.org/. Another interesting P2P technology developed by Apple is Rendezvous that enables automatic broadcasting and discovering services between clients/servers.

Not Just Mobile Phones

Aside from cellular phones, mobile games can be developed for various other mobile handsets such PDAs, SymbianOS and Microsoft's SmartPhone. Beware though some of these may or may not have JVM installed to run the J2ME application or game. In other words you may have to include in your deployment the JVM.

You will find as well other niche devices such as the JEMBlazer. What is the JEMBlazer? Here is a direct quote off the website

> *"Tired of playing games on tiny cell phone screens, with poor controls, and crawling system performance? The aJile Systems JEMBlazer JFlame cartridge will turn your Nintendo® Game Boy Advance or Game Boy Advance SP system into a jammin' Java interactive gaming machine capable of playing those free Java games popping up all over the Internet. Based on Sun Microsystem's J2ME Mobile Information Device Profile (MIDP), the JEMBlazer accelerated Java platform will run MIDP 1.0/2.0 games and other multimedia MIDlets. The JEMBlazer cartridge also capable of delivering game sound, so you can enjoy the ultimate MIDP gaming experience."*

For more information on the JEMBlazer visit their site at http://www.jemblazer.com.

Figure 1 - Screen Shot JEMBlazer

What Can I Make?

Start Simple

Okay now that you know what tools and the types of technology that support mobile game development and before we walk through the game example in this book you may want to start playing with the idea of what kind of game you want to make. If don't have any previous game experience you should stick with something simple. Start with a simple standalone game that doesn't do much visually nor interactively and yet still gives you a good understanding how a game works and the other required information needed to make a complete game. Avoid trying to make a killer game on your first try, you'll end up facing many hurdles and probably get discouraged. It is better to finish a simple game and feel sense of accomplishment and then tackle a harder game.

A natural progression is to start with something simple like a turn based low graphic game where both graphics and threading is easier to handle. A candidate for this would be TicTacToe. This will give you simply workings of how games work. As well it will introduce simple artificial intelligence. Then maybe your next choice is to make a Tetris clone. This is great game to learn basic game interaction, user controls/input, graphic manipulation, and the use of tiles in games. There after you may want to make a basic shoot em up game simpler to Space Invaders and eventually make a side scroller game simpler to Mario Brothers. After you've made this for basic games you should be able to combine the skills learned from all four to make more complicated games. Of course, better graphics and artificial intelligence is required to make the game more addictive and playable.

Game Categories / Genre

What kind of games is there? Well generally if you look closely at any game they all fall under a certain kind of genre and really there really hasn't been game that as been new! Really! Some games may seem really cool elaborate but if you break them down they have just re-used one or more ideas from past games. If you look at games like DukeNukem, Doom, Quake, Freelancer, Counter Strike, Return Castle Wolfestein, they are really just the same ole games that were in 2D but now are in 3D.

Games can be basically broken down into the following categories:

Category	Description
Arcade/Action	Fast-paced, Rich graphics, highly interactive. For example, Quake and Freelancer
Puzzle	Games that require logic like Tetris
Card	Games such as Poker, BlackJack
Strategy	Requires a lot the thinking and tactical moves and possible micro management, ie: Command and Conquer, Warcraft
RPG	Games that require you to play a role and usually involves character building over time, ie: Dungeons and Dragons
Sport	Games that resembles sports

The above are more of the common categories of course there are others like trivia, simulation, chance games like Magic 8 ball … etc.

Copying Game Ideas

Most likely if you copy a game for fun or for learning you won't get into trouble. But if you copy a commercial game and plan to market it yourself then you may find yourself in trouble. However, there is thin line between cloning games and copying games. Take for example, you've probably seen dozens of games similar to Tetris and are commercially sold that is because you will notice they don't use the word Tetris which is copyrighted by the creator of Tetris. The thin line is the concept or game idea, its really difficult for one to patent a concept or idea. But if you do plan on making a game for commercial reasons it is probably best that you try and come up with an original idea and possible integrated with ideas borrowed from other successful games. Most game publishers won't publish your game if its something this been already done. This is even more so in the mobile industry where games have a very short self-life of about 3 to 6 months. Even if you just want to release the game for fun most people are looking for a new game to play not another clone of Tetris.

Competition

In any industry you can learn from others or evaluate what hasn't been done yet, so it is best to get to know the competition.

A couple of great places to keep in the know are:

- Midlet Review – http://www.midlet-review.com
- Midlet.org – http://www.midlet.org
- Midlet Centrel – http://www.midletcentral.com
- Micro Java – http://www.microjava.com
- AllNetDevices – http://www.allnetdevices.com
- WirelessDevNet –http://www.wirelessedevnet.com
- Infosync – http://www.infosync.no
- WGR (Wireless Gaming Review) - http://www.wirelessgamingreview.com

Here is list of some the competition you are up against, but on the other hand these can be potential aggregators or partners. The next section will go into more detail in bringing your game to market.

AbstractWorlds
AirG
AnfyMobile
Astraware
Atatio
BlueSphereGames
CheekyGroup
Cocoasoft
Codetoys
CoffeebreakMedia
Comzexx
CovertOperations
DigitalBridges
DistinctiveDevelopment
Dreamquest
DSEffects
Eemo
Elkware
FutureDesignGroup
Gameloft
Getsnax
HandyGames
Hudson
Ifone
In-Fusio
Iomo
Jsmart
Jamdat
livingmobile.net
Macrospace
Mforma
MicroJocs
Mobilegames
MobileScope
Mobizexx
MoBro
Morpheme
Mr.Goodliving
Notthefly
Paletsoft
Perimind
Picofun
segawireless
SoftexIndia
Sorrent
Sumea
Toomtam
THQWireless
WES

Note this list is not complete

Bring to Market

Now that you have the tools, game idea(s) and with the chapters to come the knowledge how to build a game the question is how do make money with my game.

There are several options available to you.

Direct Sale

You can simply put up your own website and use an easy to use online merchant such as PayPal. What is the catch? First of all, if user downloads the game to PC, how does he/she get the game onto his/her mobile handset? The question is then does he/she know how to upload games to his/her mobile handset? Does he/she have the data cable to upload the game? So to maximize successful sales you will definitely need to provide OTA when selling your games, meaning you need a provision infrastructure, which may include your own billing system rather then using something like PayPal. Remember when using something like PayPal there will be manual intervention on your part to verify the orders and to setup a temporary area for the customer to download the game.

Another way of having direct sales but not worrying about setting up your own server is going with online store such as Handago, http://www.handago.com/. The catch here is then Handago receives a percentage of the games sold. But still advertisement and direct relationship with cellular customers is somewhat limited. The user needs to explicitly visit Handago to purchase games.

Aggregator

Another option is to approach an aggregator. Aggregators are like middlemen between you the developer and the carrier who delivers your product to their cellular subscribers. This is good for small independent developers who do not want to hassle with owning the company and going through all the business red tape. It is a fairly simple process when working with an aggregator, basically all you have to do is sign a NDA, sign a contract and deliver a production ready game. The aggregator takes care of both marketing and billing. Of course, each aggregator will have a different set of terms and profit sharing program; you will need to do some investigation to see which works best for you. Things to look out for are of course the revenue model, SLA (Service Level Agreement) meaning the type of support you will provide, and if the deal is non-exclusive meaning can you have other aggregators sell our product and/or you allowed to work with other aggregators.

Game Publisher

Game publisher or established game organization can help you publish your game. You will have to directly contact these game companies and see if they have a program in place and if so find out if the terms are something you can work with.

Carriers

Direct relationship with carriers is another option you can seek. But most likely you won't get too far simply for the fact most carriers will only deal with large well established gaming companies, usually companies that have already built a solid reputation in the gaming industry with non-mobile products and now are adding to their profile mobile games. As well carriers like to deal with aggregators who not only take care a lot things you rather not deal with they take care of a lot tasks carriers rather not deal with as well.

Manufacturer

Some mobile handset manufacturers like Nokia include in there developer programs bring to market partnership opportunities. You will have to visit each manufacturer you are interested in working with to obtain more detail.

Wireless Talent Agency

There is also the wireless talent agencies, these organization are similar to the agencies found in the movie business they focus on selling you and your skills and portfolio.

Summary

There is a lot more involved in mobile gaming development if you wish to bring your game to market. At the same time there is an abundance of tools and resources to help you get started. Of course you can always code for fun! Well actually if you aren't have fun then in the first place you might want to think about getting into something else. Game coding for fun or profit it is one of the more challenging areas in computer programming because of the change in technology and fierce competition.

Here is a small list of various organizations you may want to start with in bringing your game to market. This is a very short list of available sources to help you market your game:

Company Name	Type of Company	URL
AirG	Game Company	http://www.airg.com
AnfyMobile	Game Company	http://www.anfymobile.com
Nokia	Manufacturer	http://forum.nokia.com
WirelessDeveloper	Agency	http://www.wirelessdeveloper.com
TiraWireless	Aggregator	http://developer.tirawireless.com
Nextel	Carrier	http://developer.nextel
T-Mobile	Carrier	http://www.tmobile.com
CellMania	Aggregator	http://www.cellmania.com
Cingular	Carrier	http://alliance.cingularinteractive.com
Telus	Carrier	http://www.telus.net
BellMobility	Carrier	http://www.developer.bellmobility.ca
4thPass	Aggregator	http://www.4thpass.com

Chapter 2 – Mobile Game Constraints

The biggest differences and hurdles when developing J2ME mobile games you will notice quite quickly are the things you took for granted are no longer conveniently there. Such features such as memory, screen size, even color. All of these are usually not a factor for console and PC development.

These constraints are more evident in a mobile game then in a mobile application because of the high interaction between user inputs, graphics, animation, sound and/or vibration. In addition, you do not only have to take into consideration different manufacturers but as well different mobile handset models for the same manufacturer. Phone models can differ vastly from model to model in memory, color, screen size and user interface.

Memory

Types of Memory

In general working memory otherwise known as heap memory is the area of memory where the game stores information during execution of the game and is released when the game is terminated. You will have to refer to the manufacturers manual for the exact specifications. This is important to you because if the game is bigger then the allocated working memory on a device then the game simply won't run.

Another memory you need to concern yourself with is the storage memory otherwise known as RMS the Record Management System. You need to be aware of the total allowable storage that is available for the particular handsets you wish to deploy to and possibly build an alternative logic into the game for cases when memory does run out. You might ask the question what do I need RMS for? You may want to store local top scores, user preferences and an option for the user to save his/her last game so he/she can continue where he/she left off.

Fragmentation

Similar to your desktop the hard-drive can get fragmented over time, this can occur as well for memory on mobile handsets. Take for example if a large object is needed to be written to memory and is unable to find enough consecutive room for it store itself, it will store itself in pieces across memory. This will not only possibly cause increased memory access time; but also, when the large object is cleared it leaves holes in the memory which then increases the chance of other newly created objects to be spread across memory for the current game session..

Display Size and Color

Aside from memory another factor you must take into consideration is the size of the screen for each mobile handset. Take for example a Sony Ericsson P800 when folded out has pixel display of 208x320 and a Nokia 3650 has a display of 176x208. You may consider releasing specific version that includes the appropriate image sizes. The Nokia may have sprites the size of 16 x 16 pixels and the P800 may have sprites the size of 32 x 32. If you are asking what are sprites, this will be explain later on, for now just think of them as graphic images.

Figure 2 - Illustrate Difference in Screen Size Matters

Though more and more mobile handsets are being released with color screens you should take in consideration the millions of existing phones already sold on the market that only have black and white displays.

Phone Interfaces

In the last couple of years manufacturers are have moved away from the more traditional phone interface / form factor to a more radical layout of buttons and controls. This is due to the fact manufacturers are in search of the best user-friendly interface that appeals to the general public. Though some manufacturers are realizing there isn't really a so called best interface it really depends on what type of user you are and what the mobile handset will primarily be used for other then for calling, such as listening to music, playing games, or taking down business notes and contacts. Of course, the newly design interfaces add a marketing edge and appeal to the general public as well. What does this mean to you as a developer? It is one more thing you need to take in consideration, for example you set the numeric button '5' as the fire button this may or may not be easily accessible across all mobile handsets. Below is an illustration of the different types of form factors available on existing mobile handsets on the market today.

Figure 3- Illustrate Difference in Interface Matters

Missing Classes

Because J2ME, Java Micro Edition, is a subset of the Java Standard Edition it doesn't contain all the packages and classes. In other words you will either have to do without them or implement them yourself. For example, there is no floating number support, in other words you are unable to do decimal calculations or use things such as sin, cos and tan. But these are usually essential when developing games. However, you can overcome the decimal problem by using fixed-point math. This is achieved by mapping decimal numbers to whole numbers. For example, if you wanted precision to 3 decimal places, 1.000, you need to represent the number as 1000. To convert the number back and forth you will need to divide or multiple by the appropriate multiples of 10. For angles you can directly hardcode the fixed point mapped value for the major angles. The major angles would probably be 0, 30, 60,90,120,150,180,210,240,270,300,330,360 given the fact mobile devices have small screen display any more accurate would be pointless. Chapter 5 will go into detail on how to handle math constraints.

There are several other classes that are missing; you probably won't notice them until you need them. Like the float class there maybe already pre-made classes/packages by others who have bumped into the same problem you are facing, so do search on the web before you decide to code it from scratch. An available floating point library can be found at the http://henson.newmail.ru/j2me/Float.htm.

What is Performance to You and the User

When measuring performance there are few things you need to straighten out first. What does performance really refer to? In general, performance means response time and/or throughput. But what does response time really mean? Does it mean the time it takes to carry a particular function or does mean the time the user gets back a response? What is the difference? Take for example, while the game is loading up the user sees a status bar that indicates the load process but the actual loading of the game is not complete. Throughput usually refers to the amount of work done in a given amount of time. Another important performance issue especially when making network games is latency, usually referring to the minimum time to care out any tasks. So before you even begin measuring performance you need to clarify what is does performance mean to you and how to measure performance.

As for the user's perception of performance it is largely based on perceived performance or responsive performance. Meaning what the user sees or feedback he/she receives from the game. You should never allow the game to pause without any feedback, this will likely occur when initially loading the game resources or when network communication is involved. During these time-consuming tasks the user should be given an indicator informing the user the game is busy loading or some sort of feedback so the user is not left wondering if there is problem with the game.

Ways to Improve Performance

Code First Improve Performance Later

First of all, when designing and developing your game the rule of thumb is to code the game without any real concern about performance. You should focus more on making the code clean, working and understandable. Then after do a performance evaluation and fix what needs fixing. Otherwise you might end up either wasting your time or even worse hinder your program by implementing unnecessary performance tactics. As well by profiling your code you can find areas where execution is occurring the most. There isn't really any point in trying to improve performance where the code only executes 10 percent of the time. This is usually referred to as the 80/20 rule or 90/10 rule, simply focus on improving on area where execution occurs most of the time. A good approach to performance profiling is to perform bench marking before and after the performance changes where implemented. This will allow you to evaluate if there is really a gain in performance and to determine if of trade offs are worth it. For example, if there isn't much gain in performance then you may want to consider keeping the more abstract and re-useable code.

Less Object-Oriented Code

In general, with object oriented programming languages there tends to be more classes and more overhead with frameworks and design patterns being implemented into the over all design. This is usually a good thing but in the mobile world where memory is limited a more procedure approach in programming is sometimes the preferred method of coding. Though it may feel wrong at first you should avoid using patterns like the model-view-controller that tend cause your program code/classes to blow up exponential. After saying this do not start coding completely procedurally, just keep this in mind and beware you may have to remove some object-oriented approaches when you start finding performance or memory problems.

Less 3rd Party Libraries

With the idea of using fewer classes, you should also avoid using 3rd party APIs to reduce both memory space and memory execution. For example, using own defined interpretation of data when communicating over a network may improve performance. Instead of a more abstract industry defined method of communication such as web services via SOAP protocol. Not only does this add to the over all size of your program but adds processing to parse and build the XML data to be transmitted.

Minimize Communication

When performing network communication you should avoid multiple fetches to the server. For example, unlike in web development large images are sliced into several tiny images and are fetched on individual basis. To avoid increase chances in latency, fetching all resources at once is the preferred method in mobile development.

Combine Graphics

Common resources all games have are graphics; by having the graphics all on one sheet not only can possibly reduce latency in network games but also can reduce the size of the game. Remember each image contains in the header pre-defined information, if you have 10 images then this information is there 10 times, likewise if you have all the images combined one sheet then the header information only exists once. Of course you will need a method to extract the individual images, this will be explained later.

When using images in your game ensure that have been optimized and compressed, note not all art/graphic tools optimize images by default this may have to be explicitly done.

Garbage Collector

Though we as Java developers cannot explicitly make the JVM cleanup unused objects we can help it along by assigning null to objects we do not wish use anymore, we can use object pools to reuse objects instead creating new ones. As well to kick start the garbage collector we can invoke it by calling *System.gc();* or *Runtime.getRuntime().gc();* Now after mentioning this there are many people who believe these approaches work and many who do not believe that these work. Well without getting bogged down in another never-ending argument do the memory profiling for yourself and determine if it works for you. Beware it may act differently on an actually mobile handset.

Shorten Names and Obfuscate

Aside from reducing the amount classes in your program it is a good idea to reduce variable name lengths, class name lengths and method name lengths. If possible only use the default package; do not create unnecessary package structures.

Of course, you do not want to make your code unreadable by replacing all your variables, methods and classes with single letter names. But you can get help by putting your code through an obfuscator. An obfuscator reduces the code by using various techniques one of which is to replace long names with short names. This makes your code more efficient in size; as well gives another positive side effect, if the code were to be reversed engineered it is now a lot harder to read. There are various obfuscators available:

- ProGuard - http://proguard.sourceforge.net/
- SourceGuard – http://www.javalobby.com
- RetroGuard – http://www.retrologic.com
- CodeShield - http://www.codingart.com/codeshield.html
- yGurad - http://www.yworks.de/en/products_yguard_about.htm
- IBM JAX – http://www.alphaworks.ibm.com/tech/JAX
- JShrink – http://www.e-t.com/jshrink.html

Coding Tips

The following tips are only suggestions and may or may not give gains in performance, it is to your own judgment and discretion to use them or not.

1. Use StringBuffer instead of String because of the fact the String object can not be changed. Any modification to a String variable is actually a new object creation.
2. Accessing class variables directly is faster then using setter and getter methods
3. Using local variables are more efficient then instance/class variables
4. Using variables are more efficient then arrays.
5. Avoid synchronization in loops because there is an extra over head to lock and unlock each time the loop occurs
6. Counting down in loops is faster then counting up
7. Use compound operators like x += 1 instead o f x = x + 1 because fewer byte codes is generated
8. Remove constant calculations in loops
9. Reuse objects
10. Assign null to unused objects, especially unused threads
11. Try to use already built in methods, for example if you want to copy data from one array to another use *System.arraycopy* more then likely this method will be more efficient then the one you created yourself

Good Coding Practice

Though this doesn't directly affect performance, in fact this tip is more of an over all tip for any kind of development. You should have in place some sort of source control such as CVS and a structured build process. More on how to approach development with good design and process will be further explained in Chapter 3. You may want consider using tools such as Ant to build the code. The more control you have on your entire development process the more able you are to trace and pinpoint problems as they occur.

Appendix A demonstrates the use of J2ME and Ant; includes the use of an obfuscator, ProGuard.

Summary

Mobile development especially with developing games is a very challenging due to the fact of several limitations that mobile handsets naturally inherit. However, once you have mastered the techniques in improving performance you will be able to work wonders on mobile handsets.

Chapter 3 – Before Code

Overview

Regardless if you are coding for fun or for profit, if in fact you are making a game for profit it is definitely important that you understand and apply game design and methodology process in game development. Yes, many regard this as a waste of time or even a roadblock to creativity. Game development can be very challenging but aside from the heavy emphasis on creativity, graphics and animation there really isn't much difference when compared to other industries in the software development world. This is even more so now when the gaming industry is now moving towards multiplayer and MMOG (massively multiplayer online gaming) gaming. This still remains true with mobile games; though most mobile games are currently stand-alone there is a huge push towards making network games.

What this really means is gaming is moving towards the service industry. What do we mean by service? Well the service is the game itself, it is seamless available online 24 hours and 7 days a week especially if the there is subscription paid service in place. In any online game authentication, privacy and security are of the most important factors if not the most import for any system that is publicly accessible.

Let us compare an online game to a large retail chain with real time systems in place to support multiple systems including CRM, Supply Chain management and online ordering.

Area	Online Game	Retail Chain
Good Game Play	Yes	n/a
Good AI	Yes – Need intelligent enemies, need intelligent friendly bots	Yes – project future sales, specials, track user buying habits, provide customer specific specials…etc
Intuitive Interface	Yes – easy to use and easy to adapt to	Yes – easy to use and easy to adapt to
Persistence Data / Database	Yes – to hold game info, saved games, game attributes… etc	Yes – to hold all items, specials, allowances, customer info… etc Most likely will require more storage space then games, data warehouse.
Real-Time	Yes – Reduce latency, increase throughput, increase response time	Yes - Reduce latency, increase throughput, increase response time
Monitoring	Yes – Monitor performance, potential problems ie people trying to cheat the system	Yes – monitor all flow of production, problems that arise
Authentication, Security, Privacy	Yes – encrypt private data especially if subscription model is in place and credit card transaction are done	Yes – encrypt private for things such private user information (address, phone#, credit card…etc)
Good Graphics	Yes – may be more so in gaming with 3D rendering	Yes – especially for consumer applications like an online store. As part of successful marketing campaign presentation of product is key to a successful sale

As you can see there really is not difference, granted some areas on one system may require more emphasis such as game like Doom 3 will definitely require a lot more graphic capabilities. On the other hand the up time for a system such a monitoring tool for nuclear plant is a lot more critical then multiplayer gaming hosting service.

But the real point here is gaming is not different then any other software development and that it should be approached with good design and a development process.

Though the above comparison is based on network games good design and process still applies to stand-alone games. Design and process definitely contributes to the success of delivering a good quality game within budget and on time.

Game Design

Game design is one of the key elements in any successful game. Gaming today is lot more then the Wow factor! This might have been true in the early days when one or two coders would bunker down in the basement or attic and hack out some killer game. This isn't so true of today; games are a multi-million dollar business. Yes, it was mentioned earlier mobile currently is not to that scale but still doesn't mean good game design is not important. The costs of the mobile game development itself maybe lower then compared to the cost of developing a game on for the PC or gaming console but the other associated costs will remain the same. Costs such as the amount resources and marketing strategies put into advertisement and costs for infrastructure especially if the game is multiplayer. So really with all that investment the game should be developed right from the start with the goal of being successful, meaning a well designed game.

Aside from the business side of things it is as simple as this if the game sucks no one will play it and worse your reputation as an independent developer could be tarnished. We will go over some the elements in good game design; however, this is just really a brief overview of what makes a good game. There are entire books written on game design alone, if you lack the experience in game design I highly recommend in investing some money and time into one these books, it truly is worth it.

Re Use and Component Development

In a software development you should give some thought on reusing any components you have developed such artificial intelligence algorithm, 3D math engine, or even as small as customize sprite class.

Aside from reusing your own code in several different projects look at purchasing or using open source software. For example, why develop you own customized database when there are dozens of databases on the market that have proven track record. Another consideration is what happens if the old team members with the knowledge of the proprietary database system leave for whatever reason, who is going take over? The task for someone to familiar themselves with the database let alone become the expert will be huge loss in time and resources. This wouldn't be the case if you used already made database such as mySQL or Oracle there are thousands of experts out there and companies that provide support for such well-known databases.

Demographic and Target Audience

When making a game you must understand the audience. Take for example; a soccer game in North America isn't as of big a hit when compared to a hockey game, or baseball game. But in Europe soccer is one of the most popular sports and sells dramatically. Another example would be adult and gambling games are more acceptable in parts of Europe and most of Asia, if not all. Unlike in North America where games that contain adult content and/or gambling is generally frowned upon by the general public. Culture and tradition are other major factors to consider demographically. It may not be wise to produce a game where one can kill cows in a society where cows are worshiped.

Even within the same country you need consider age, background, and type of person you are trying to target. A golf game will mostly likely only capture the interest of golfers and fast action brainless total carnage killing game will appeal more to young male teenagers and young male adults. You may consider targeting very specific audiences like the female teenager market, tweens or the children's market.

Game players themselves can be broken up into 3 basic categories:

- **Hard Core Game Player** – these are people who spend a great deal of time and money on gaming. They will even sacrifice other commits for pleasure of gaming. As well a hard core gamer will try to keep up with all the latest and greatest games and accessories for games with no real regard of how much it costs.

- **Moderate / Casual Game Player** - Moderate game players are somewhat like hardcore game players in that they will spend both a great deal of time and money on there gaming habits. However, the key difference is that a casual player will not sacrifice as much such as spending time with the family. A casual player will be conscious about where he/she spends his/her money on.

- **General Mass Game Player** – They type of people are typical people who play very simplistic well know classic games like TicTacToe, Monopoly and Blackjack.

Referring to the above categories and with the current state of mobile technology it is probably best to target the *general* and *moderate* game players.

Whatever the case may be research is needed to determine what type of audience and demographic you want to target the game towards will play a factor in the over game design.

Type of Game

In chapter one we mentioned Game Categories, each of these categories can break down into 4 simple types of games:

- Stand-Alone without persistence
- Stand-Alone with persistence
- Multiplayer without persistence
- Multiplayer with persistence

Stand-alone is self-explanatory; an example of a stand alone game without persistence does not allow you to continue your game from the where left off the last time and a stand-alone with persistence does. Persistence can be easily implemented in numerous ways such as a flat file, XML file or use of a database. A multiplayer is similar except multiplayer with persistence usually referred to MMOG (multiplayer massively online gaming), which has they unique realism of not only can you continue where you left off but the game itself and everyone else in it continues on 24 hour seven days week basis. This alone is what makes MMOG a very difficult game to make along with the fact that MMOG is much more then a game but rather a game that serves a community of real life people who all need to be entertained.

Long Term Plans

Though most games are self-contained you still may want to consider if you going to release any significant updates or patches, such as additional maps, characters or levels. This of course goes back to the design of the game components allowable for easy maintenance and upgrades.

Technology

This is one those areas that can stir up a heated conversation, many will argue technology is just a tool that allows you to realize your design. But on the other hand technology itself may have constraints that will and will not allow you to do certain things. This is especially more so when dealing with the constraints of a mobile handset. The advice here is to keep an open mind and try avoid having technology dictate the design of them game and yet when designing the game keep in mind the limitations of the technology you are using.

Game Play

Good game play basically is the idea of keeping the player interested in playing the game over and over again. The game must challenge the user is some shape or form whether it be skill, memory, thinking, problem solving, obtain high score or even pure seek and destroy; there has to be an achievable goal for the user. Albeit that goal may be nearly impossible but not so difficult that it is impossible otherwise you've taken the user to the other extreme and the user gives up. Two other possible contributing factors to a good game play is the character development, the player can improve his/her characteristics over time. The other is game balance; this sort of refers back to obtaining the goal. In

game of Pan-Man the balance is between the ghosts ability to catch and not being able to catch the user.

Real-Life

Real life added in games adds to the over all effectiveness of game play. If the player finds the game too unreasonable from reality he/she will become discourage from playing the game. Likewise too much realism in a game can make the game too difficult making it once again discouraging to play.

Be Creative

Though we stress the importance of design and use of good development processes you should still be creative as well. The trick is here is to find the balance between the two worlds of creativity and structure. Though not an easy task, try to create games that have not been done before, unless of course you business goal is target a certain demographic like general players who only have interests in Connect Four or Tetris. But even so it is quite possible to come up with new innovative puzzle game that simply just puts in a different twist on old game.. An original game will contribute to the success of your game.

Borrow

Creativity can be hard! Actually damn hard at times! So it does not hurt to borrow ideas from other games or even from other resources. It can be anything you see in normal daily livelihood from what you see in the movies, to what you read or even what see as you driving down the road. Inspiration can come from anywhere the hard part is recognizing it when does happen.

Design Patterns

Though use of design patterns are very common when developing with an object-oriented language you should consider the extra memory that will taken up when using design patterns. However, this doesn't mean design patterns are not good, as mentioned before you should go ahead and design the game as best you can then after it is coded re-factor/optimize the appropriate areas.

Quick Summary

Game design is a vast area and is acquired through both in theory and real-life game development experience. It is highly advisable for you to do addition research when treading into areas that are more complex and maybe beyond your current gaming design capabilities such as designing a MMOG. However, hopefully you have gotten the basics and when you do create, and design your first couple of games you will have these things in mind to help make your game a success.

Development Process

To increase your chances to success when developing games or any software at all a development process needs to be applied. Though process along with design is traditional viewed as a waste of time and once again a roadblock on creativity it will truly help the development process. There is a need for an organized structure process from design, implementation (development time), deployment, post deployment, build process to source control. Some the benefits of process are:

- Disallows scope creep, in the gaming world stops people from adding "neat/cool to have features"
- Handles all High Risk Scenarios up front
- Plans for every stage (design, development, deployment, post production support)
- Source Control Management
- Iterative Development
- Build Management

List above are just a few of the advantages when using and applying a good process to your development. Even if you are single developer developing a single game, it is not only good practice; but also, can potentially save you from problems for the time when you start going beyond your simple development environment. This can easily happen right at the moment you decide to support more then one handset, then three, then four then a dozen different handsets for different carriers and manufactures. Quite possibly after the excitement of successfully deploying your game onto a carrier you start your second one then your third, so on and so forth. It can't be stressed enough to that you should manage and follow a development process to increase success rate and decrease failure.

Methodologies

Now that you understand the importance of process, we will now briefly go overview few well-known process methodologies. Afterwards we will draw the conclusion which process best fits gaming development.

Chaos

Well by name of the methodology, this really isn't a methodology; however, unfortunately many are still using this. Usually this occurs with small startup companies, individual or few individuals who think up a cool idea and decide to start coding right away. The thinking here is lets get something rolled out fast as possible and throw in what ideas and neat to have features as we go along. In the end you may have complete game, but after it hits production and is deployed to thousands or even millions of handsets there are weird bugs that start showing up. Ok, even if the first patch works, it still doesn't save you the revenue lost because extra support and possible refunds that accumulated with the release of the first faulty version.

Chaos is obviously not a choice for game development, and really isn't a choice for any real development. But this shouldn't be mixed up with proof of concept or RAD, in these cases chaos may be acceptable. But the problem with this is in many situations where the proof of concept should have been thrown out and restarted with a proper process it is thrown into production with added features.

Water Fall

One of the older and traditional development processes followed is the Water Fall Model. Where development is done in phases with documents for each phase. Once a phase is approved it is finalized and the next phase begins. Where the next phase is based on the last phase's documents but because the last phase is frozen any changes to the last phase is prohibited, unless it is a major one. That is why the diagram below the arrow going back on phases is represented as a thin line. One other major significance to this process is the majority if not all testing is done at the end of the project completion.

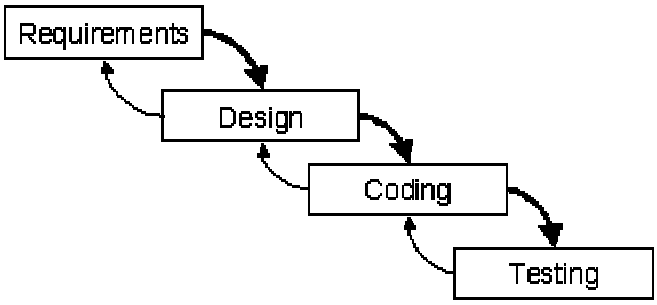


Figure 4 - Diagram Water Fall Process

As you can see this is not feasible when developing games. The rule about freezing at the end of each phase is what stops any changes that needs to be made, thus in game

developers view this is what stops creativity. Testing in the end is definitely bad for gaming where a lot of it is based on creativity and the unknown. Testing should be done through the process, especially on components that have not been done before. For example, a new multiplayer engine using a new protocol needs to be proven up front that it works efficiently and can handle the specified load.

In general, the waterfall process is frowned upon in today's development world. However, it has been said if you have a very well defined set of requirements and specifications for a very well now system and you know what the results should be the water fall process may be the process to follow. An example of system where you would know all the rules and possible outcomes would be an accounting system. However, in the gaming world where there are a lot of change, creativity and development that pushes limits of technology the waterfall model is definitely not an adequate process to use. Because the waterfall model has traditional been around for long time since the 1970s and has failed in game development it has instilled in game developers process and design is waste of time.

Spiral Process

The spiral process is quite similar to the waterfall process except there is more of them, this is a really a very loose comparison you will have to do more research if you are interested in the spiral process. You can think of it has bunch mini waterfall process chained together. However, the improvement the spiral process provides is easier to adapt to changes, user feedback is done through the development cycle and testing is done throughout. The spiral characteristics itself tends to lead to poor project management, hard to track and hard to determine the project end date. So once again this doesn't suit a gaming development where the project end date and tracking are critical to the project.

Rational Unified Process (RUP)

RUP is document centric process; these documents are referred to as artifacts. The key artifacts are the Use Cases that guides the entire development process along. RUP is strict and ridge process the forces collaboration among all parties of interest in the project. This strong discipline is what keeps the project on track, deals with high risks up front; provides excellent communication between all parties involved. As well recognizes software development is on going changing process and encourages change through an iterative process.

The main disadvantage seen by some is the amount of artifacts needed in every phase of the process. Many projects fail not because of RUP itself but members of the team are unable to stick the ridge discipline of documentation. This again is especially so in the gaming world where documentation is seen as *boring thing to do* and blocks creativity. However, RUP may be option in game development when the game project is very large and large amount of artifacts are necessary to support the discipline. A possible example would be a MMOG project that requires good process for requirements/features gathering, good game design, a large team made up of people with different expertise. Expertise such as AI developers, Graphic artists, Network developer specialists, Network

infrastructure specialists, QA team and the list goes on and on. When this being such as complicated project made up with so many members a rigid process such as RUP may be the answer.

More Information about the Rational Unified Proess can be found at http://www.rational.com

XP

Extreme programming has become more popular in the last few years it goes against a traditional software development in the sense that is focused on short cycle development rather then driven by documents. Some the features of XP are:

- Small teams
- Developer develop in pair (pair-programming)
- Test before code
- Several Iterations
- Highly interactive with the end user

XP is exactly what the name says and extreme approach to getting the requirements right as soon as possible. This is done by having 2 coders working together side by side at the same workstation coding both test cases and the project itself based on excellent feedback provided by the user of the system.

XP is potential candidate for game development if and only if the project and project team fit the requirements exactly for the XP development. Otherwise XP can easily fall back into a waterfall like approach. For example, XP will only work for small teams say about the size of 2 to 12 members. As well for pair programming to work both developers much work well with each other and are at the same level of expertise.

More information about Extreme Programming can be found at http://www.extremeprogramming.org/

Agile

Agile is are softer versions of XP, some these are:

- Scrum
- SDM
- Lead Development

Process that falls under the Agile concept are the characteristics of measuring progress through a workable software, customer needs are first priority, small team sizes, continuous testing, documents are done throughout and delivered throughout.

More information about Agile development can be found at http://www.agilealliance.org

Summary

We've briefly reviewed some the processes that are out there in development world. The question really is are any these good in the gaming environment? Yes, XP and Agile are excellent candidates when you have small-centralized teams that allows for focused iterative development. RUP is an excellent candidate when you have large teams made up of different members with different expertise who may be spread out demographically.

However, if carefully done you can make hybrid process is RUP as the main process but certain areas to with the RUP process use characteristics from XP.

What about an individual developer, following process with large amount of documentation or unable to satisfy the requirements to make proper work (unable to do pair programming when there is only one of you) can become discouraging resulting in falling back to a chaos like development. As individual you should at minimum have complete game design document, build process and source control. Though many will argue this isn't enough but given the fact that there is only one person this is reasonable. As well you should incorporate an iterative process with unit testing throughout the each iterative process.

As reminder all the above-mentioned process where only brief introductions. There are vast amounts of information of each of those processes. The main goal of this section was to introduce you to these processes and how they can possible fit together in a gaming environment. It is ultimately your decision or the people in charge to decide what process best fits the development environment you are in.

Lastly do not forget your main objective to make a game. There isn't any point in using and putting large amounts of effort into design and process if you fail to produce a successful quality game on time and within budget.

Build Process

A contributing factor to development process is to have in place a good build process. A build process is nothing really special it is simply a set of guidelines and rules that dictate how the code will be built and a formalization of where things should be. For example, as simple as organizing a nice directory structure will help you keep your code and resource in line. Where to separate your resource files in one area, source in another, jad & manifest in other, and final build in other directory all under the same project. Each project would have a similar break down. This would be your development layout by default this is some what already setup for you when using the Wireless Tool Kit from Sun Microsystems. To help you automate all builds or to invoke automatic unit testing with tools such as JUNIT (http://www.junit.org) it recommended you use build tool like ANT (http://ant.apache.org) which does exactly that. Remember you may or may not be the only one who will be working with the same game. By formulizing a build process it keeps everyone in sync with how things are to be done.

Source Control

Good management of your source code can be vital to the success of your project. Source control is essential anytime there is more then one developer working on one project. This ensures all code is checked in and checked out without loss or over written code. Source control gives you further management with respect to different stages of testing from development, test, quality acceptance and ultimately production.

Even as single individual developer source control allows you to manage several projects at once or even the different versions of the same project deployed on different carriers or different mobile handsets.

By keeping everything in organized manner it will definitely reduce/prevent source code confusion and loss of productivity and possibly loss of code.

Some source control programs out there are CVS, Visual Source Safe, BitKeeper and eChangeMan.

Chapter 4 – MIDP 2 Game Classes

The popularity of J2ME and game development has sprouted several carrier and manufacturer specific custom classes that support game development. Of course, the main problem with this is portability, for example using Siemens Sprite class would make it difficult for you to port your game to a Nokia handset, simply for the fact you now have to re-implement the sprite class.

But now with the release of MIDP 2.0 some of those problems in game portability is taken away with the introduction of five new classes:

- GameCanvas
- Sprite
- Layer
- LayerManager
- TiledLayer

Though you probably won't use the Layer class directly, it is actually an abstract class used by Sprite, LayerManager and TiledLayer. These classes can be found in the following package java.microedition.lcdui.game.

Not only is portability less of problem but also with these new game classes your code may potentially become a lot smaller now that you do not have to implement custom classes such as Sprite. These classes are now apart of the underlying Java environment on the mobile handset.

GameCanvas

The GameCanvas brings to the table 2 very important features that solve two problems that previously existed when using MIDP 1.0. First feature is now a single buffer for each GameCanvas that is created. This is important because it not only minimizes the heap usage your game functionality; but also, the game can now be control in one single loop. Let us take a look at the old way of looping through a game using MIDP 1.0.

```
public void theGameCanvas extends Canvas implements Runnable {
  public void run() {
    while (true) {
          repaint();  // update game display
      }
  }

  public void paint(Graphics g) {
    // painting // redraw occurs here
  }

  protected void keyPressed(int keyCode) {
    // obtain user inputs
  }
}
```

As you can see there are basically three different areas of functionality, painting of the screen, run() area and key input; all of which run on different threads. Because of the different threads the final display to the user may sometimes seem jerky especially for arcade/action type games that require a lot o graphics and interactive ness. As well it is somewhat awkward to keep track of the three different functionalities in three different places.

Now with MIDP 2 and the implementation of GameCanvas, everything is a lot more cleaner, easier to use and more efficient. Aside from running in single thread the GameCanvas no longer waits for a keyPressed event instead it used the a technique called polling, meaning you can determine which keys where pressed at any point time by using the getKeysState() method provide by the GameCanvas. With the use of the buffer in GameCanvas the gaming technique called double buffering graphics is done automatically. All you have to do is call the flushGraphics() method to output the graphics to the display. Double Buffering is a technique that is used to avoid flickering on the display by simply drawing a temporary image off set from the actually display and when completed the image is then shown in visible display area.

Basic GameCanvas Example

The following is a simple example of GameCanvas being used, in this example the string character ‘x’ moves according to the users input

GameCanvas:

```
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;

public class ExampleGameCanvas extends GameCanvas implements Runnable {
  private boolean isPlay;   // Game Loop runs when isPlay is true
  private long delay;       // To give thread consistency
  private int currentX, currentY;  // To hold current position of the 'X'
  private int width;        // To hold screen width
  private int height;       // To hold screen height

  // Constructor and initialization
  public ExampleGameCanvas() {
    super(true);
    width = getWidth();
    height = getHeight();
    currentX = width / 2;
    currentY = height / 2;
    delay = 20;
  }

  // Automatically start thread for game loop
  public void start() {
    isPlay = true;
    Thread t = new Thread(this);
    t.start();
  }

  public void stop() { isPlay = false; }

  // Main Game Loop
  public void run() {
    Graphics g = getGraphics();
    while (isPlay == true) {

      input();
      drawScreen(g);
      try { Thread.sleep(delay); }
```

```
      catch (InterruptedException ie) {}
    }
  }

  // Method to Handle User Inputs
  private void input() {
    int keyStates = getKeyStates();

    // Left
    if ((keyStates & LEFT_PRESSED) != 0)
      currentX = Math.max(0, currentX - 1);

    // Right
    if ((keyStates & RIGHT_PRESSED) !=0 )
      if ( currentX + 5 < width)
        currentX = Math.min(width, currentX + 1);

    // Up
    if ((keyStates & UP_PRESSED) != 0)
      currentY = Math.max(0, currentY - 1);

    // Down
    if ((keyStates & DOWN_PRESSED) !=0)
      if ( currentY + 10 < height)
        currentY = Math.min(height, currentY + 1);
  }

  // Method to Display Graphics
  private void drawScreen(Graphics g) {
    g.setColor(0xffffff);
    g.fillRect(0, 0, getWidth(), getHeight());
    g.setColor(0x0000ff);
    g.drawString("X",currentX,currentY,Graphics.TOP|Graphics.LEFT);
    flushGraphics();
  }
}
```

Main Midlet:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class ExampleGameCanvasMidlet extends MIDlet {
  private Display display;

  public void startApp() {
    display = Display.getDisplay(this);
    ExampleGameCanvas gameCanvas = new ExampleGameCanvas();
    gameCanvas.start();
    display.setCurrent(gameCanvas);
  }

  public Display getDisplay() {
    return display;
  }

  public void pauseApp() {
  }

  public void destroyApp(boolean unconditional) {
    exit();
  }

  public void exit() {
    System.gc();
    destroyApp(false);
    notifyDestroyed();
  }
}
```

Figure 5 - Simple GameCanvas Example Source Code

Figure 6 - Simple GameCanvas Example Emulator Screen Shot

Sprite

What is a Sprite

In any successful game one of the contributing factors are excellent graphics. Most objects in a game are categorized as a special kind of graphic called sprites. A sprite can be anything from a bullet, monster, the main character, enemies, special power items, keys and doors to name a few.

Figure 7 - Example of a Sprite

A lot times sprites are animated graphics. These animated graphics are made up of several instances of the same sprite with each of them slightly different. These sets of sprites are usually referred to as a set of frames. These frames can be render on the screen sequentially or non sequentially. Typically sprites are displayed sequentially for ease of coding.

Figure 8 - A Sprite Set

Figure 6 Illustrates a set of sprites representing the wait frame and 4 basic other frames.

Sprite Constructor

There are 3 constructors that come with the Sprite class

- Sprite (Image image) – Creates single frame sprite, non-animated
- Sprite (Sprite sprite) – Creates a new Sprite from another Sprite
- Sprite (Image image,int frameWidth, int frameHeight) – Creates an animated sprite with 2 more frames, the frameWidth is the width of one sprite and the height is the height of one sprite

Figure 9- Sprites Set with Grid Overlay

Taking another look at Figure 6 we can break down the entire sprite set into the individual frames. In this particular example the total width is 160 pixels, which divided by 5 gives you a frame width of 32 pixels. The height for each frame is 32 pixels. The height and weight do not have to be the same, but the width and height must remain constant for all sprites in the same set. In other words you cannot have frame one with

the width of 32 pixels and the remaining sprites have a width of 16 pixels; all frames must have the same width. In the constructor Sprite (Image, image, int frameWidth, int frameHeight) you will notice you do not have to specify the number of frames, this will be automatically calculated by the sprite class.

Why 8 Bit, 16 Bit, 32 Bit?

You will notice most graphics including sprites will usually fall under the same width and height constraints. This is because the number of pixels used are in correlation to the number colors used. This is referred to as bit depth or color depth. The more bits per pixel the more colors there are. The formula is

$$2^{\#\text{ of bits}} = \text{total colors}$$

Using the formula the bit depths are 8,16,24, and 32 bits translate to

$2^8 = 256$ colors
$2^{16} = 65{,}536$ colors
$2^{24} = 16.7$ million colors
$2^{32} = 16.7$ million colors plus an 8-bit alpha channel, an alpha channel is really a mask, it specifies how a color of one pixel when merged with another pixel should be. Merging occurs when one pixel is on top of another pixel.

However, you can have Sprites bigger then 8 by 8 pixels and only with 256 colors. Ideally this is probably your best option when dealing with graphics for mobile handsets. The more colors there are the more processing is required to render the graphics.

Sprite Collision

Sprite Collision is one of the most essential functions of any interactive action arcade like game. This can be anything from a sprite running into wall, a sprite firing a bullet or two sprites running into each other. Collision detection doesn't necessary means death or game over. It can mean unlimited amount of possibilities such as level ups, power ups, loss of traction in a racing game, opening a door, or indicator allowing the player to climb the ladder.

So how do we detect sprite collision? Naturally we should detect if the pixel of one Sprite overlaps/collides with another pixel of another Sprite.

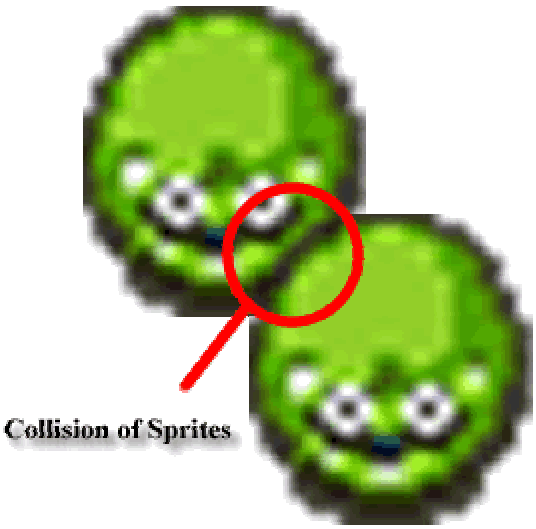


Figure 10- Sprite Collision

Fortunately, the Sprite class comes with a method that does just that.

collidesWith(Image image, int x, int y, boolean pixelLevel)

or

collidesWith(Sprite sprite, boolean pixelLevel)

As you can see you can detect collision with another sprite or an image. The method with image as an input you need to specify the location of the image, this is referring to x and y of the image's top left upper corner. The pixelLevel is a boolean value where *true* indicates pixel level detection and *false* indicates rectangle intersects. You can define the size of the rectangle intersects you may want to consider defining the rectangle intersects to be slightly smaller then the image itself. This then eliminates the odd areas of the

sprites where collision occurs because the invisible rectangles collide but there are no opaque pixels that have collided.

Pixel-Level detection is when the opaque pixel of one sprite overlaps another sprites opaque pixel. A more simple method of collision detection is the collision of rectangle intersects of the two sprites usually the rectangles are the size of the images' bounds.

There is a third method

collidesWidth(TiledLayer tiledLayer, Boolean pixelLevel)

This is similar to the last two methods except the collision is checked against a graphic tile layer. TiledLayers will be explained in more detail in the next section.

Display Sprite

To display or render the sprite simply call the paint method, you will need to pass in the Graphics object as it is the required the parameter. Reminder you may have to use the setVisible(boolean) method first calling the paint method, you will need to pass in the Boolean value true.

Display Sprite Sequence

There are few methods available when dealing with sprite frame sequence.

- getFrameSequenceLength() – returns the number of elements in a frame sequence
- getFrame() – retrieves the current entry index number in the frame sequence, this is not the frame that was currently displayed
- nextFrame() – set frame sequence to the next frame, if the sequence is at the last frame it starts at first frame
- prevFrame() – set frame sequence to the previous frame, if the sequence is at the first frame it sets the frame to the last frame
- setFrame(int sequenceIndex) – to manually set the sequence in the frame sequence
- setFrameSequence(int[] sequence) – to manually preset a predefined frame sequence

See J2ME API for more details.

Sprite Transparency

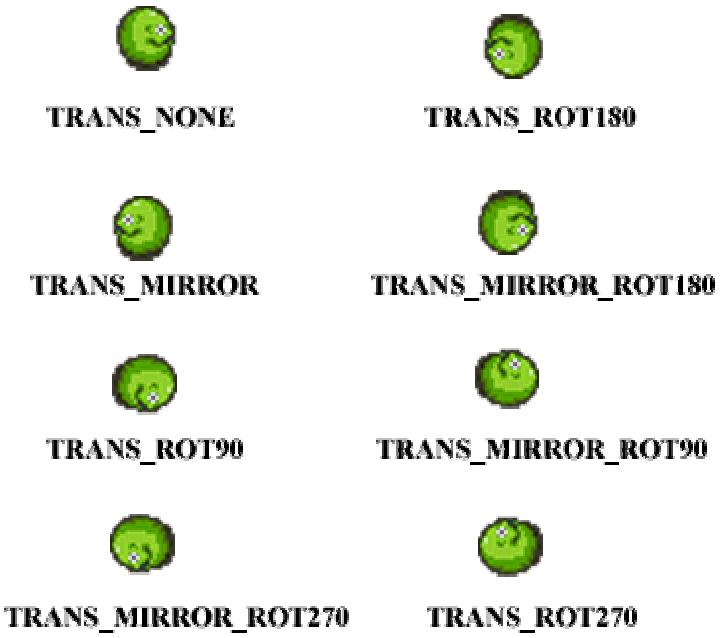
A reminder you need to beware of transparent and non-transparent images depending on the situation. If it is a simple game with little animation such as TicTacToe transparent images may not be important or required. In highly interactive games where there is a lot potential for sprite collision and/or varying background images you may want to consider using transparent images. As well you should check with the mobile handset manufacturer an ensure that the mobile handset you will be deploying to does indeed support transparent images if you do decide to use transparent images.

Figure 11 - Non Transparent Sprite Vs Transparent Sprite

Sprite Transforms

There are methods that manipulate that can manipulate a Sprite in rotations of 90 degrees and mirrored versions of the rotations. The transform is set by invoking the setTransform(transform) method. The parameter is actually an integer value; however, there are valid predefined static values available for your usage:

Static Fields	Integer Value
TRANS_NONE	0
TRANS_MIRROR_ROT180	1
TRANS_MIRROR	2
TRANS_ROT180	3
TRANS_MIRROR_ROT270	4
TRANS_ROT90	5
TRANS_ROT270	6
TRANS_MIRROR_ROT90	7



When transforms are applied the sprite is automatically repositioned so that it remains in the center. Therefore the reference point for the sprite does not change, but the values returned from getX() and getY() will change according to the current position of the upper-left corner of the sprite.

Sprite Optimization

As reminder you should verify what colors and resolution is supported on the mobile handsets you are going to deploy to as well all production graphics should be optimized. Image optimization is usually the process of removing colors the human eye cannot see. The benefit of optimization is of course reduction in image size, which means over all reduction of the jar file.

Basic Sprite Example

The following is a simple sprite example, which builds on the last example from the GameCanvas section; you will notice you can only move the center sprite the sprite in the corner is there demonstrate what happens if transparency is not used with the image:

Main Game Canvas with Sprites:

```
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;

public class ExampleGameCanvas extends GameCanvas implements Runnable {
  private boolean isPlay;   // Game Loop runs when isPlay is true
  private long delay;       // To give thread consistency
  private int currentX, currentY;  // To hold current position of the 'X'
  private int width;        // To hold screen width
  private int height;       // To hold screen height

  // Sprites to be used
  private Sprite sprite;
  private Sprite nonTransparentSprite;

  // Constructor and initialization
  public ExampleGameCanvas() throws Exception {
    super(true);
    width = getWidth();
    height = getHeight();
    currentX = width / 2;
    currentY = height / 2;
    delay = 20;

    // Load Images to Sprites
    Image image = Image.createImage("/transparent.png");
    sprite = new Sprite (image,32,32);

    Image imageTemp = Image.createImage("/nontransparent.png");
    nonTransparentSprite = new Sprite (imageTemp,32,32);
```

```
}

// Automatically start thread for game loop
public void start() {
  isPlay = true;
  Thread t = new Thread(this);
  t.start();
}

public void stop() { isPlay = false; }

// Main Game Loop
public void run() {
  Graphics g = getGraphics();
  while (isPlay == true) {

    input();
    drawScreen(g);
    try { Thread.sleep(delay); }
    catch (InterruptedException ie) {}
  }
}

// Method to Handle User Inputs
private void input() {
  int keyStates = getKeyStates();

  sprite.setFrame(0);

  // Left
  if ((keyStates & LEFT_PRESSED) != 0) {
    currentX = Math.max(0, currentX - 1);
    sprite.setFrame(1);
  }

  // Right
  if ((keyStates & RIGHT_PRESSED) !=0 )
    if ( currentX + 5 < width) {
      currentX = Math.min(width, currentX + 1);
      sprite.setFrame(3);
    }

  // Up
  if ((keyStates & UP_PRESSED) != 0) {
    currentY = Math.max(0, currentY - 1);
    sprite.setFrame(2);
```

```
    }

    // Down
    if ((keyStates & DOWN_PRESSED) !=0)
      if ( currentY + 10 < height) {
        currentY = Math.min(height, currentY + 1);
        sprite.setFrame(4);
      }
  }

  // Method to Display Graphics
  private void drawScreen(Graphics g) {
    g.setColor(0xFF0000);
    g.fillRect(0, 0, getWidth(), getHeight());
    g.setColor(0x0000ff);

    // display sprites
    sprite.setPosition(currentX,currentY);
    sprite.paint(g);
    nonTransparentSprite.paint(g);

    flushGraphics();
  }

}
```

Main Midlet:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class ExampleGameSpriteMidlet extends MIDlet {
  private Display display;

  public void startApp() {
    try {
      display = Display.getDisplay(this);
      ExampleGameCanvas gameCanvas = new ExampleGameCanvas();
      gameCanvas.start();
      display.setCurrent(gameCanvas);
    } catch (Exception ex) {
      System.out.println(ex);
    }
  }

  public Display getDisplay() {
```

```
    return display;
  }

  public void pauseApp() {
  }

  public void destroyApp(boolean unconditional) {
    exit();
  }

  public void exit() {
    System.gc();
    destroyApp(false);
    notifyDestroyed();
  }
}
```

Figure 12 - Basic Sprite Example Source Code

Figure 13 - Simple Sprite Example Emulator Screen Shot

Extending the Sprite Class

There are few reasons why you might consider inheriting the Sprite class:

1. Add addition Sprite functionality
2. Use the Sprite class as your base Sprite class for all game sprites
3. Encapsulating functionality

Referring to the first point, you may find that the current Sprite class supplied by sun may not have all the necessary features you need for example you might be making a race car game. In most racing games velocity is a factor, so you may want to inherit the Sprite class and add in the necessary variables and methods you need to use.

Another example where you may want to inherit the Sprite class is say for example you have several different types of sprites but each type has their own set of characteristics. For example, say the player sprite has attributes such as strength; agility, speed, intelligence and the enemy sprites have attributes indicating what kind of enemy it is and a life bar. Both sprites still share the same kinds of characteristics such as position and name. You can now create a base sprite class that contains the common features for all sprites, as well create child sprites that contain both their own characteristics and inherently still retain the common characteristics.

You probably have noticed in the Simple Sprite Example the only sprites movements are left, right, up and down. If you try to press 2 keys at once like right and up to produce an angled direction it simply defaults to one of the other basic directions. To produce a SW, SE, NW or NE movement you need to define both the proper sprites and the proper calculations. This is just a matter of some basic math manipulation, but more importantly where should this be done? To keep you code a little more organized and clean you may consider encapsulating this work in a custom Sprite.

Making Your Own Sprite Class

If the Sprite class for whatever reason totally does not suit your needs you have the option to inherit the Layer class or completely start from scratch and implement your very own Sprite class, of course you will have to name it something else.

The following is an incomplete example of what Sprite class may look like if implemented from scratch, you can take this example an complete it to your customized requirements.

```
import java.microedition.lcdui.*;l

public class MySprite {
  private Image image;
  private int positionX;
  private int positionY;
  private int frameWidth;
  private int frameHeight;
  private int numFrames;
  private int currentFrame;
  private boolean visible;

  public MySprite(Image image, int frameWidth, int frameHeight, int numFrames)
                        throws Exception {
    this.image = image;
    this.frameWidth = frameWidth;
    this.frameHeight = frameHeight;
    this.numFrames = numFrames;
  }

  public int getX() {
    return this.positionX;
  }
  public void setX(int positionX) {
    this.positionX = positionX;
  }
  public int getY() {
   return this.positionY;
  }
  public void setY(int positionY) {
    this.positionY = position Y;
  }
   //  Continue Your Code here
}
```

Where to Find Sprites

Well coding your game to handle sprites is one thing but producing or finding sprites is another. Like most developers you probably do not have the skills to produce nice looking sprites.

Some well-known web links with sprites are:

- http://www.arifeldman.com
- http://www.idevgames.com
- http://www.spriteworks.com

Like anything else please read and abide by the terms of use before using any sprites you find for free.

Your other option is to simply hire professional sprite artists sometimes called pixel pushers. You may want to contract freelance graphic artists and find out their rates sometimes they will do it for free depending on what type the project is, especially if it is an open source project. You will need to do some investigation in where to find these types of freelancers. Just as developers hang out at developer sites, artists hang together in there own world.

Lastly you can always learn to make your own sprites. This may not be such as bad idea if you plan to make a simple 2D game and if you don't mind spending the time to learn more about sprite drawing. Granted if you do not have the artistic talent you are probably better off with the first 2 suggestions. But remember graphics are important but getting the game to work is more important. So in the beginning you may just want to use simple but not so pretty graphics until you have a fully function game. Then you can look at improving the graphics.

LayerManager

What is the LayerManager

In last section of this chapter we were dealing with one sprite, of course, we can easily put more then one sprite on the screen but what about the scenery. Virtually all games will have some sort of background whether it is animated, still or scrolling there is usually something there to give the game more visual appeal. This is where the LayerManager comes in, the LayerManager does exactly what it is named as, manages graphic layers. It will render all the layers under its control in the correct area and in sequential order. In other words it overlaps images over one another in an easily manageable fashion.

You can think of the ordering of layers as a 3rd dimension commonly called the z.

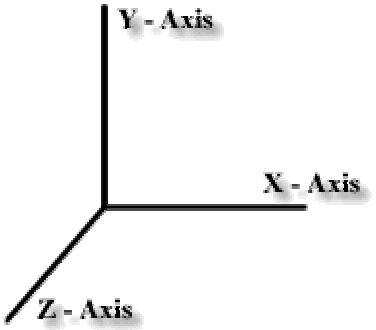


Figure 14- Z Axis Illustration

These layers on z-axis are indexed in a numeric order starting from zero. Index 0 is the layer closest to the user; likewise the layer with the highest index value is furthest from the user. If there are any adjustments to the number of layers in the LayerManager the numbering of indexes with respect to each layer is re-numbered according to keep a continuous sequential order. This would occur if layers are added or removed dynamically throughout the game.

How to use the LayerManager

To add an layer the method *append(Layer layer)* is available

```
LayerManager = new LayerManager();
layerManager.append(layer);
```

To retrieve a layer a specific index use the *getLayerAt(int index)* method. To obtain total number of layers currently in the LayerManager use the *getSize()* method. To remove a layer use the *remove(Layer layer)* method. To add a specific layer at specific index the following method *insert(Layer layer, int index)* will allow you to do this. To display the layers call the paint(Graphics g, int x, int y) method.

Beyond the Basics

In the last section the mentioned methods are fairly straightforward and really you could easily implement this yourself without using the LayerManager. However, there is one more method that provides a very convenient feature, the setViewWindow(int x, int y, int width, int height) method. This feature allows you to set both the actual size of the window the user can view and what part of the layer to display in the view. Yes this sounds a bit confusing at first but is more easily understood through the following diagram.

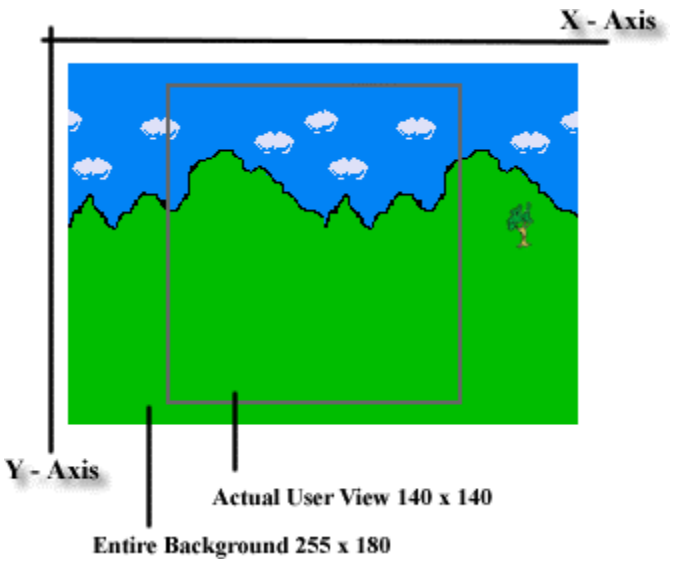


Figure 15 - setViewWindowIlustration

The actual method implementation for the above diagram would be setViewWindow(55,20,140,140). Where the first set of numbers is where the top left corner of the User View is located with respect to the entire background, (55,20). The second set of number 140 x 140 is the actual width and height for the User view. The demo code and actual screen shot of this will be demonstrated in the next section.

Basic LayerManager Example

Game Canvas using the LayerManager:

```
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;

public class ExampleGameCanvas extends GameCanvas implements Runnable {
  private boolean isPlay;   // Game Loop runs when isPlay is true
  private long delay;       // To give thread consistency
  private int currentX, currentY;  // To hold current position of the 'X'
  private int width;        // To hold screen width
  private int height;       // To hold screen height

  // Sprites to be used
  private Sprite playerSprite;
  private Sprite backgroundSprite;

  // Layer Manager
  private LayerManager layerManager;

  // Constructor and initialization
  public ExampleGameCanvas() throws Exception {
    super(true);
    width = getWidth();
    height = getHeight();

    currentX = width / 2;
    currentY = height / 2;
    delay = 20;

    // Load Images to Sprites
    Image playerImage = Image.createImage("/transparent.png");
    playerSprite = new Sprite (playerImage,32,32);

    Image backgroundImage = Image.createImage("/background.png");
    backgroundSprite = new Sprite(backgroundImage);

    layerManager = new LayerManager();
    layerManager.append(playerSprite);
    layerManager.append(backgroundSprite);

  }

  // Automatically start thread for game loop
  public void start() {
    isPlay = true;
```

```
  Thread t = new Thread(this);
  t.start();
}

public void stop() { isPlay = false; }

// Main Game Loop
public void run() {
  Graphics g = getGraphics();
  while (isPlay == true) {

    input();
    drawScreen(g);
    try { Thread.sleep(delay); }
    catch (InterruptedException ie) {}
  }
}

// Method to Handle User Inputs
private void input() {
  int keyStates = getKeyStates();

  playerSprite.setFrame(0);

  // Left
  if ((keyStates & LEFT_PRESSED) != 0) {
    currentX = Math.max(0, currentX - 1);
    playerSprite.setFrame(1);
  }

  // Right
  if ((keyStates & RIGHT_PRESSED) !=0 )
    if ( currentX + 5 < width) {
      currentX = Math.min(width, currentX + 1);
      playerSprite.setFrame(3);
    }

  // Up
  if ((keyStates & UP_PRESSED) != 0) {
    currentY = Math.max(0, currentY - 1);
    playerSprite.setFrame(2);
  }

  // Down
  if ((keyStates & DOWN_PRESSED) !=0)
    if ( currentY + 10 < height) {
```

```
        currentY = Math.min(height, currentY + 1);
        playerSprite.setFrame(4);
      }
    }

    // Method to Display Graphics
    private void drawScreen(Graphics g) {

      //g.setColor(0x00C000);
      g.fillRect(0, 0, getWidth(), getHeight());
      g.setColor(0x0000ff);

      // updating player sprite position
      playerSprite.setPosition(currentX,currentY);

      // display all layers
      layerManager.paint(g,0,0);

      flushGraphics();
    }

}
```

Main Midlet:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class ExampleLayerManagerMidlet extends MIDlet {
  private Display display;

  public void startApp() {
    try {
      display = Display.getDisplay(this);
      ExampleGameCanvas gameCanvas = new ExampleGameCanvas();
      gameCanvas.start();
      display.setCurrent(gameCanvas);
    } catch (Exception ex) {
      System.out.println(ex);
    }
  }

  public Display getDisplay() {
    return display;
  }

  public void pauseApp() {
```

```
  }

  public void destroyApp(boolean unconditional) {
    exit();
  }

  public void exit() {
    System.gc();
    destroyApp(false);
    notifyDestroyed();
  }
}
```

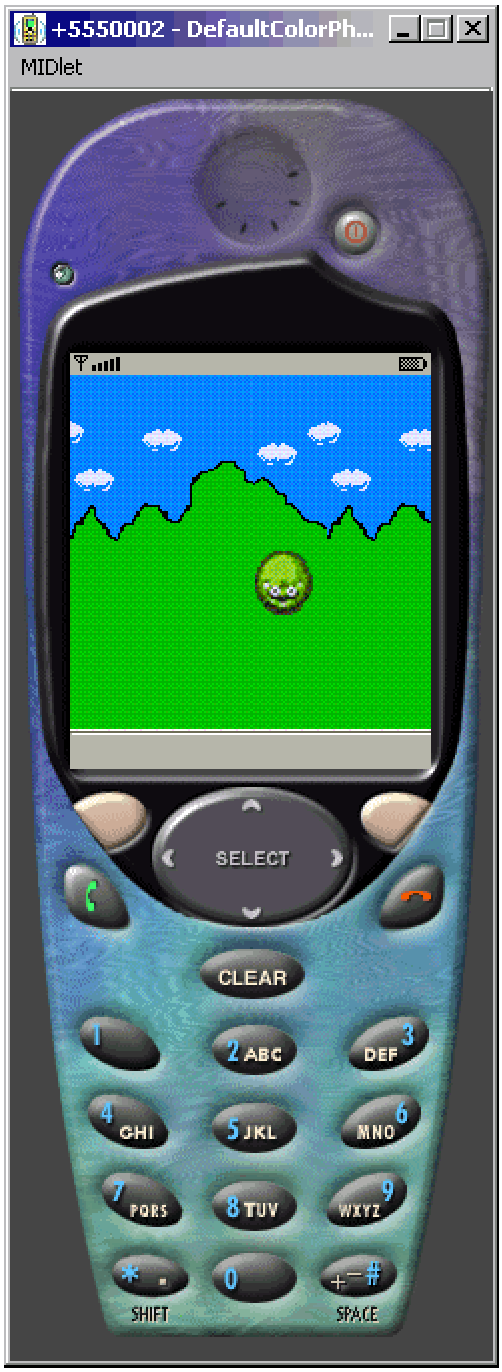


Figure 16 - Simple LayerManager Screen Shot

Figure 17 - LayerManager using setViewWindow method Screen Shot

In figure 15 demonstrates the use of the setViewWindow, the following is the source snippet to do this (replace the current drawScreen method).

```
 // Method to Display Graphics
 private void drawScreen(Graphics g) {
   g.setColor(0xffffff);
   g.fillRect(0, 0, getWidth(), getHeight());
   g.setColor(0x0000ff);

   // updating player sprite position
   playerSprite.setPosition(currentX,currentY);

   // display all layers
   layerManager.setViewWindow(55,20,140,140);
   layerManager.paint(g,20,20);

   flushGraphics();
 }
```

One more note you will notice just using the setViewWindow method the user viewing area will appear offer center. If you want to adjust this you will need to put in the appropriate values in the paint method in this case to center the view the values are 20-pixel a-axis and 20-pixel y-axis. This point is offset from the actual screen itself. Do not confuse this with the first values in the setViewWindow method. Remember these two numbers are offset from the layers being used.

Now that you have some space at the top and bottom of the screen you can now add additional feed back to the user. For example, you can now display the high score, current user score, level and number of lives.

Figure 18 - Dislay Extra Display Features Demo Screen Shot

LayerManager and Scrolling Background

You've probably noticed the background image we have been using is bigger then the display screen itself.

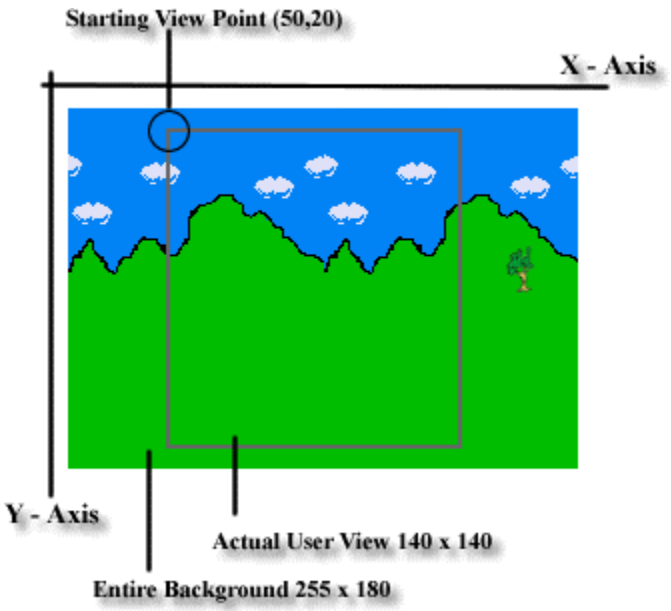


Figure 19- Scrolling View

In this particular example the starting point is (50,20). All you have to do is make the value 50 a dynamic value so that it decreases if you want the background to scroll to the left or increase if you want the background to scroll to the right.

The following code illustrates how this can be done, for simplicity the green smiley sprite has been removed and now if you press the right game key the image will scroll right and if you press the left game key the image will scroll left.

Note in this example, only the horizontal value is made dynamic nothing will happen if you decide to push the down or up key. Which in this scenario makes sense, but of course you can easily change this to scroll up/down or even scroll in all 4 directions.

Scrolling Game Canvas:

```
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;

public class ExampleGameCanvas extends GameCanvas implements Runnable {
  private boolean isPlay;   // Game Loop runs when isPlay is true
  private long delay;       // To give thread consistency
  private int width;        // To hold screen width
  private int height;       // To hold screen height
```

```
private int scnX, scnY;   // To hold screen starting viewpoint


// Sprites to be used
Image backgroundImage;
private Sprite backgroundSprite;

// Layer Manager
private LayerManager layerManager;

// Constructor and initialization
public ExampleGameCanvas() throws Exception {
  super(true);
  width = getWidth();
  height = getHeight();

  scnX = 55;
  scnY = 20;
  delay = 20;

  // Load Images to Sprites
  backgroundImage = Image.createImage("/background.png");
  backgroundSprite = new Sprite(backgroundImage);

  layerManager = new LayerManager();
  layerManager.append(backgroundSprite);

}

// Automatically start thread for game loop
public void start() {
  isPlay = true;
  Thread t = new Thread(this);
  t.start();
}

public void stop() { isPlay = false; }

// Main Game Loop
public void run() {
  Graphics g = getGraphics();
  while (isPlay == true) {

    input();
    drawScreen(g);
    try { Thread.sleep(delay); }
```

```
      catch (InterruptedException ie) {}
    }
  }

  // Method to Handle User Inputs
  private void input() {
    int keyStates = getKeyStates();


    if ((keyStates & LEFT_PRESSED) != 0) {
      if (scnX - 1 > 0)
        scnX--;
    }
    if ((keyStates & RIGHT_PRESSED) != 0) {
      if (scnX + 1 + 140 < backgroundImage.getWidth())
        scnX++;
    }
  }

  // Method to Display Graphics
  private void drawScreen(Graphics g) {

    //g.setColor(0x00C000);
    g.setColor(0xffffff);
    g.fillRect(0, 0, getWidth(), getHeight());
    g.setColor(0x0000ff);

    // display all layers
    layerManager.setViewWindow(scnX,scnY,140,140);
    layerManager.paint(g,20,20);

    flushGraphics();
  }

}
```

Main Midlet:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class SimpleScrollingLayerManger extends MIDlet {
  private Display display;

  public void startApp() {
    try {
```

```
      display = Display.getDisplay(this);
      ExampleGameCanvas gameCanvas = new ExampleGameCanvas();
      gameCanvas.start();
      display.setCurrent(gameCanvas);
    } catch (Exception ex) {
      System.out.println(ex);
    }
  }

  public Display getDisplay() {
    return display;
  }

  public void pauseApp() {
  }

  public void destroyApp(boolean unconditional) {
    exit();
  }

  public void exit() {
    System.gc();
    destroyApp(false);
    notifyDestroyed();
  }
}
```

Figure 20 - Simple Scrolling Background using LayerManager Screen Shot

Press the left or right game keys to view the scrolling effect

TiledLayer

What is TiledLayer?

TiledLayer is the last but certainly not the least significant class that is available in the game package. Games with large backgrounds / virtual areas the TiledLayer is the perfect candidate in dealing with this. It allows you to simply to define all the unique areas of the background and reuse them as many times as need to produce a complete image. Take for example the following image:

Figure 21 - TiledLayer Example Screen Shot

In figure 19, if you study the image you will notice there several areas that are similar and in fact you can break the image up into 6 distinct areas.

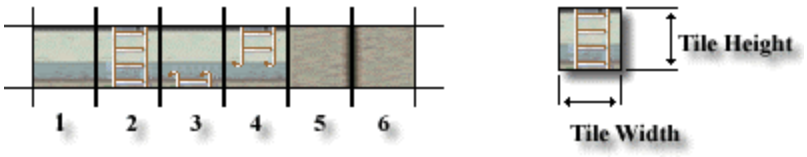


Figure 22 - Example Tile Sheet

You will notice all tiles have the same dimensions 32 x 32 pixels as required by the TiledLayer class. Each tile is given an index numeric value starting from 1. The numbering occurs from left to right and top to bottom. In other words numbers are assigned row-by-row bases. You can lay the tiles in any configuration and assuming you kept the same tile order the assigned index values will not change.

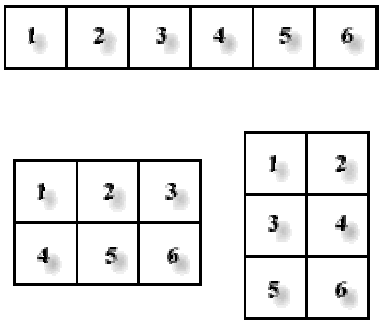


Figure 23 - TiledLayer Assigned Index

In the above example these tiles are considered static tiles because of the one to one fixed relationship from tile to index value. If the index contains the value zero this indicates that cell is empty, meaning any cell with the value zero nothing will drawn in that area the cell occupies.

TiledLayer Constructor

To created a TiledLayer simply instantiate the constructor
TiledLayer(int columns, int rows, Image image, int tileWidth, int tileHeight)

The first 2 values refer to the number of columns and rows that make up the entire final image. The third parameter is the image of tiles / tile sheet used to map against the index values. The last 2 parameters is the actual width and height for one specific tile.

TiledLayer Manipulation

To map a tile image to a specific cell you need to use the *setCell(int col, int row, int tileIndex)* method. Column and row is the location where you want the tile image to be rendered at and the last parameter maps the tile to be render at that cell location.

To replace the entire tile set / tile sheet after you have called the constructor simply use the *setStaticTileSet(Image image, int tileWidth, tileHeight)* method. If the new tile set is the same or contains more tiles then the original tile set the animated tiles and cell contents will remain the same. However, if the new set contains less tiles then the previous tile set then the mapping will be reset; all indexes will contain the value zero and all animated tiles will be deleted.

To fill specified region of cells with a specific tile you can use the fillCells(int col, int row, int numCols, int numRows, int tileIndex) method. The first to values indicate the column and row for the top-left cell in the specified region. Next 2 define the number of columns and rows for the region. The last parameter is the index you want to map/place in the region.

Display TiledLayer

Like the other game objects all you need to do is call the paint method directly or use the LayerManager.

Retrieve Current TiledLayer Settings

There are several self-explanatory methods available to obtain information on the current TiledLayer in use:

- getCell(int col, int row)
- getCellHeight()
- getCellWidth()
- getColumns()
- getRows()

Animated Cells

There is one more feature TiledLayer has to offer which is animated tiles. Animated tiles are indicated by negative index values. Each animated cell is dynamically associated to a static tile. This allows us to easily animate a group of cells simply by changing the associated static cell. This is great for background animation like a cheering crowd, moving clouds and/or rippling water. Aside from the previously mentioned methods there are three additional methods specific to animated cells:

- *createAnimatedTile(int staticTileIndex)* – creates a new animated tile a the specified index and returns the next consecutive negative index for the animated tile. By default it will contain a static tile (positive number) or the value zero.
- *getAnimatedTile(int animatedTileIndex)* – retrieves the tile map to the animated tile index.
- *setAnimatedtile(int animatedTileIndex, int staticTileIndex)* – links an animated tile to a static tile

Non-Animated TiledLayer Example

GameCanvas Source Code:

```
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;

public class ExampleGameCanvas extends GameCanvas implements Runnable {
  private boolean isPlay;   // Game Loop runs when isPlay is true
  private long delay;       // To give thread consistency
  private int width;        // To hold screen width
  private int height;       // To hold screen height


  // Layer Manager
  private LayerManager layerManager;

  // TiledLayer
  private TiledLayer tiledBackground;

  // Constructor and initialization
  public ExampleGameCanvas() throws Exception {
    super(true);
    width = getWidth();
    height = getHeight();
    delay = 20;

    tiledBackground = initBackground();
```

```
  layerManager = new LayerManager();
  layerManager.append(tiledBackground);
}

// Automatically start thread for game loop
public void start() {
  isPlay = true;
  Thread t = new Thread(this);
  t.start();
}

public void stop() { isPlay = false; }

// Main Game Loop
public void run() {
  Graphics g = getGraphics();

  while (isPlay == true) {
    input();
    drawScreen(g);
    try {
      Thread.sleep(delay);
    } catch (InterruptedException ie) {
    }
  }
}

// Method to Handle User Inputs
private void input() {
   // no inputs
}

// Method to Display Graphics
private void drawScreen(Graphics g) {
  g.setColor(0xffffff);
  g.fillRect(0, 0, getWidth(), getHeight());
  g.setColor(0x0000ff);

  layerManager.paint(g,0,0);
  flushGraphics();
}

private TiledLayer initBackground() throws Exception {
  Image tileImages = Image.createImage("/tiles.png");
  TiledLayer tiledLayer = new TiledLayer(10,10,tileImages,32,32);
```

```
    int[] map = {
      5,  1,  1,  4,  1,  1,  1,  1,  1,  6,
      5,  1,  3,  1,  1,  3,  1,  1,  1,  6,
      5,  1,  2,  1,  1,  2,  1,  1,  1,  6,
      5,  1,  2,  3,  1,  2,  1,  1,  1,  6,
      5,  1,  4,  2,  1,  2,  1,  1,  1,  6,
      5,  1,  1,  4,  1,  2,  1,  1,  1,  6,
      5,  1,  1,  1,  1,  4,  1,  1,  1,  6,
      5,  1,  1,  1,  1,  1,  1,  1,  1,  6,
      5,  1,  1,  1,  1,  1,  1,  1,  1,  6,
      5,  1,  1,  1,  1,  1,  1,  1,  1,  6
    };
    for (int i=0; i < map.length; i++) {
      int column = i % 10;
      int row = (i - column) / 10;
      tiledLayer.setCell(column,row,map[i]);
    }

    return tiledLayer;
  }
}
```

Main Midlet Source Code:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class ExampleTiledLayer extends MIDlet {
  private Display display;

  public void startApp() {
    try {
      display = Display.getDisplay(this);
      ExampleGameCanvas gameCanvas = new ExampleGameCanvas();
      gameCanvas.start();
      display.setCurrent(gameCanvas);
    } catch (Exception ex) {
      System.out.println(ex);
    }
  }

  public Display getDisplay() {
    return display;
  }

  public void pauseApp() {
  }
```

```
  public void destroyApp(boolean unconditional) {
    exit();
  }

  public void exit() {
    System.gc();
    destroyApp(false);
    notifyDestroyed();
  }
}
```

Screen Shot Output:
Refer to figure 19

Animated TiledLayerExample

The following example illustrates a simple example of to animated tile, though logically the tile that is being animated makes no sense but that is not the point. The point is to clearly show an animated example.

GameCanvas Source Code:

```
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;

public class ExampleGameCanvas extends GameCanvas implements Runnable {
  private boolean isPlay;   // Game Loop runs when isPlay is true
  private long delay;       // To give thread consistency
  private int currentX, currentY;  // To hold current position of the 'X'
  private int width;        // To hold screen width
  private int height;       // To hold screen height


  // Layer Manager
  private LayerManager layerManager;

  // TiledLayer
  private TiledLayer tiledBackground;

  // Flag to indicate tile switch
  private boolean switchTile;

  // To hold the animated tile index
  private int animatedIdx;

  // Constructor and initialization
  public ExampleGameCanvas() throws Exception {
    super(true);
    width = getWidth();
    height = getHeight();

    currentX = width / 2;
    currentY = height / 2;
    delay = 20;

    tiledBackground = initBackground();
    layerManager = new LayerManager();
    layerManager.append(tiledBackground);
  }
```

```
// Automatically start thread for game loop
public void start() {
  isPlay = true;
  Thread t = new Thread(this);
  t.start();
}

public void stop() { isPlay = false; }

// Main Game Loop
public void run() {
  Graphics g = getGraphics();

  while (isPlay == true) {
    input();
    drawScreen(g);
    try {
      Thread.sleep(delay);
    } catch (InterruptedException ie) {
    }
  }
}

// Method to Handle User Inputs
private void input() {
   // no inputs
}

// Method to Display Graphics
private void drawScreen(Graphics g) {
  g.setColor(0xffffff);
  g.fillRect(0, 0, getWidth(), getHeight());
  g.setColor(0x0000ff);

  // Determine which tile to show
  if (switchTile) {
    tiledBackground.setAnimatedTile(animatedIdx,3);
  } else {
    tiledBackground.setAnimatedTile(animatedIdx,4);
  }

  // Set tile file to opposite boolean value
  switchTile = !switchTile;

  layerManager.paint(g,0,0);
```

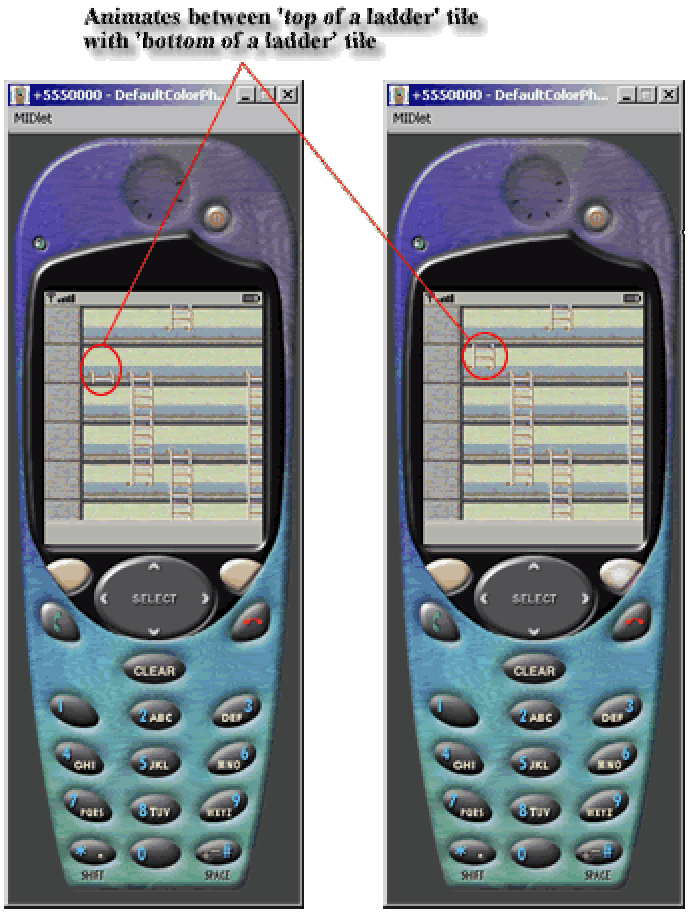
```
    flushGraphics();
  }

  private TiledLayer initBackground() throws Exception {
    Image tileImages = Image.createImage("/tiles.png");
    TiledLayer tiledLayer = new TiledLayer(10,10,tileImages,32,32);

    int[] map = {
      5,  1,  1,  4,  1,  1,  1,  1,  1,  6,
      5,  1,  3,  1,  1,  3,  1,  1,  1,  6,
      5,  1,  2,  1,  1,  2,  1,  1,  1,  6,
      5,  1,  2,  3,  1,  2,  1,  1,  1,  6,
      5,  1,  4,  2,  1,  2,  1,  1,  1,  6,
      5,  1,  1,  4,  1,  2,  1,  1,  1,  6,
      5,  1,  1,  1,  1,  4,  1,  1,  1,  6,
      5,  1,  1,  1,  1,  1,  1,  1,  1,  6,
      5,  1,  1,  1,  1,  1,  1,  1,  1,  6,
      5,  1,  1,  1,  1,  1,  1,  1,  1,  6
    };
    for (int i=0; i < map.length; i++) {
      int column = i % 10;
      int row = (i - column) / 10;
      tiledLayer.setCell(column,row,map[i]);
    }

    // Created animate tile and hold animated tile index
    animatedIdx = tiledLayer.createAnimatedTile(5);

    // Set Cell with animated tile index
    tiledLayer.setCell(1,1,animatedIdx);

    return tiledLayer;
  }
}
```

Main Midlet Source Code:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class ExampleTiledLayerAnimated extends MIDlet {
  private Display display;

  public void startApp() {
    try {
      display = Display.getDisplay(this);
      ExampleGameCanvas gameCanvas = new ExampleGameCanvas();
```

```
      gameCanvas.start();
      display.setCurrent(gameCanvas);
    } catch (Exception ex) {
      System.out.println(ex);
    }
  }

  public Display getDisplay() {
    return display;
  }

  public void pauseApp() {
  }

  public void destroyApp(boolean unconditional) {
    exit();
  }

  public void exit() {
    System.gc();
    destroyApp(false);
    notifyDestroyed();
  }
}
```

Output Screen Shot:



Chapter 5 – Math Constraints

Overview

When developing games in particular genres that require more interactive movement namely arcade like games will require more realism. One method of achieving that goal is adding into your game some basic math concepts, basic with respect to 2D games. This chapter doesn't go into the math concepts in game programming because this topic alone requires an entire book in itself. Instead this chapter shows you how to get around the J2ME math constraints. The main math constraint is the lack of support for decimal numbers and because there is not support for decimal numbers other math functions are not available such as trigonometry.

Fixed Point Math

What is fixed point math? Fixed-point math is the implementation of fractional numbers such as float or double. Why is this necessary? Well for one there is no math support in either MIDP 1.0 or MIDP 2.0. You are not able to do the following:

2.245 multiple 5.423 is not valid

No math support is not entirely true there is a very limited basic math support but nothing you can do calculations with, as well they only support either integer or long. These methods are:

- *abs(int a)* - returns absolute value
- *abs(long a)* - returns absolute value
- *max(int a, int b)* - returns the greater value of the two values
- *max(long a, long b)* - returns the greater value of the two values
- *min(int a, int b)* - returns the lowest value of the two values
- *min(long a, long b)* - returns the lowest value of the two values

There are two other advantages to use fixed-point math over fractional numbers. One of which is the fact that with fixed point all numbers have an equal distance to each other. Unlike factional numbers the distance increases from number to number when exponent increases. The other is assuming both implementations of fixed point numbers and fractional numbers are implemented on the same platform with the same language fixed point is faster.

Ok now that you have basics of fixed-point numbers the question how do we use it and/or do we get fixed point numbers. It is really just manipulation of numbers around fixed decimal point. Take for example, if you want accuracy to the thousandth position, which in most cases is good enough for J2ME™ development, ever number regardless it is fractional or not will contain fixed amount numbers behind the decimal point in this case 3 numbers. For example the numeric value 1 would look like this 1.000, but of course we don't have decimal available so the actually value would be 1000. You need keep track

of the fixed decimal point. So really what you will be doing is calculations with integer or long values. Going back to the previous examples, they would now look like the following:

> 2245 multiple 5423 is valid however when you display it you must remember the location of the fixed decimal point

You can now implement an entire fractional math class yourself, or use one that someone else's implemented, of course, remembers to adhere to the license use when using other peoples code. Some available links on the web are:

- http://bearlib.sourceforge.net/
- http://home.rochester.rr.com/ohommes/MathFP/
- http://henson.newmail.ru/j2me/Float.htm

Some points about optimization you may want to consider using bit shifting when manipulating the numbers. As well whether it is you own customized class or someone else's you may want to modify the original version for every game you use it in. In other words, only keep the code/methods that you are using because of the limited amount of space on a mobile handset it makes sense not to keep methods you are not going to use.

Basic Trigonometry and Sprite Movement

Yes, in the last section you are able to implement sin, cosine, and/or tan. However, it maybe perceived inefficient to use simply because it has to go through that other implementations of math functions before calculating the trigonometry function. You may have not choice if the range of angles is totally unknown, but with mobile gaming and constraints of the screen a sprite be made to move fluently in all directions by defining 16 directions, actually 15 if you count 0 degrees and 360 degrees as the same direction. These directions are easily calculable and can be stored in a lookup table. To save you some time in figuring out the values the following table outlines the degrees with the corresponding decimal trigonometry values. Do note we've applied the fixed point numbering to the values. For example $\sin(22.5^o) = 0.383$, with fixed point math it is adjusted to 383. You may want to do more research on the topic 2D Transformation or 2D Sprite Transformations, to gain more knowledge on how to flip and mirror images.

Degrees	Sin (Decimal Value x 1000)	Cosine (Decimal Value x 1000)
0	0	1000
22.5	383	924
45	707	707
67.5	924	383
90	1000	0
112.5	924	-383
135	707	-707
157.5	383	-924
180	0	-1000
202.5	-383	-924
225	-707	-707
247.5	-924	-383
270	-1000	0
292.5	-924	383
315	-707	707
337.5	-383	924
360	0	1000

Sprite Movement Example

Graphics Credit

Before we start with the following examples I would like to fully credit Ari Feldman, who has released several sprites under the General Public License. For more information about Air Feldman and the sprites he has released visit his website at http://www.arifeldman.com/

Without Fixed Point Math

In this example, a simple sprite, a tank's movement is based on integer values. To advance the tank you need press the forward key and to reverse the tank press the back arrow key. To turn the tank you need to either press right or left.

With Fixed Point Math

With 2D Transformation

Chapter 6 – Eliminator: Introduction

Overview

In the following chapters to come we will walk through sample vertical Scroller shoot-em up game. The main focus is to obtain an understanding of what the code looks like for a typical game. As well to pull together topics learned in the previous topics including the MIDP 2 classes and how to deal with the challenges of mobile constraints. Though this game isn't exactly the next killer game it will show some the fundamentals of what complete game involves from start to finish.

In saying that the source code it mainly target to use a J2ME Emulator, see below the *Tools Used* section for more information on the tools being used to develop the game. You will have to make the appropriate adjustments when deploying your own game onto a real mobile handset.

As well the code in general is written in a more user-friendly tutorial aspect, there may be area for improvement in design, memory usage and performance.

Tools Used

Mainly we will be using the J2ME Wireless Toolkit version 2.0 from Sun Microsystems; however, J2ME Wireless Toolkit version 2.0 has problems when rendering a scrolling TiledLayer. Nokia Developer's Suite for J2ME handles scrolling TiledLayer more efficiently. For best results it is recommended you use the Nokia Developer's Suite for J2ME to run the Eliminator game. At the time of this writing rumor has it the next version released by Sun Microsystems will resolve these issues.

Brief Game Description

The name of game is Eliminator. Eliminator will resemble you stand shoot-em up game like Raiden or 1942. The key features of the game are:

- 3 Levels based on predefined map using TiledLayer
- Boss at each Level
- Selection of 2 Ships
- Auto Save
- High Score (local or sync)
- Different Weapons (Wing Man, Bombs, Spread, Laser, shield)

Finished Product
/// Put screen shots here

Chapter 7 – Eliminator: Splash Screen

Overview

The splash screen is usually appears momentarily before the main menu appears for the game. Though a splash screen isn't necessary it adds to the over all appeal to the game. As well it allows for showing off some graphics, possible giving out some company information and/or re-enforces company branding.

Splash screens can be as simple as a Alert box or a more customize splash screen that incorporates the user of a timer and canvas. In either case, the splash screen should only appear for short amount time, approximately 3 seconds. In a customized splash screen there should be an option to allow the user to skip the splash screen and immediately get to the main game menu.

Source Code

Here is a straightforward use of both the Canvas and Timer class. However the Timer count down control is done in another class named CountDown. This is done for any future uses or complicated counters that maybe needed; re-use of the same object. For the remainder of this book the same splash screen code will be used and will not be printed out again.

Of course, the splash screen by itself is meaningless it needs to be invoked by the main midlet. This will be demonstrated in the Game Menu Chapter along with an downloadable source code.

Source code for SplashScreen.java:

```
import java.util.Timer;
import javax.microedition.lcdui.*;

public final class SplashScreen extends Canvas {
  private Display display;
  private Displayable next;
  private Timer timer;
  private Image image;
  private int dismissTime;

  public SplashScreen(Display display, Displayable next, Image image,int dismissTime) {
    timer = new Timer();
    this.display = display;
    this.next = next;
    this.image = image;
    this.dismissTime = dismissTime;
    display.setCurrent(this);
  }
```

```
  static void access(SplashScreen splashScreen) {
    splashScreen.dismiss();
  }

  private void dismiss() {
    timer.cancel();
    display.setCurrent(next);
  }

  protected void keyPressed(int keyCode) {
    dismiss();
  }

  protected void paint(Graphics g) {
    g.setColor(0x00FFFFFF);
    g.fillRect(0, 0, getWidth(), getHeight());
    g.setColor(0x00000000);
    g.drawImage(image, getWidth() / 2, getHeight() / 2 - 5, 3);
  }

  protected void pointerPressed(int x, int y) {
    dismiss();
  }

  protected void showNotify() {
    if(dismissTime > 0)
      timer.schedule(new CountDown(this), dismissTime);
  }
}
```

Source Code for CountDown.java:

```
import java.util.TimerTask;

class CountDown extends TimerTask {
  private final SplashScreen splashScreen;

  CountDown(SplashScreen splashScreen) {
    this.splashScreen = splashScreen;
  }

  public void run() {
    SplashScreen.access(this.splashScreen);
  }
}
```

Splash Screen Screen Shot:

Figure 24 - "Splash Screen" Screen Shot

Chapter 8 – Eliminator: Game Menu

Overview

The menu system to game or to any application is important especially for mobile devices. Aside from the all the factors that are needed in making a successful game emphasis needs to put on the design of the menu. The key word is usability.

The usability of the menu should be efficient, easy to use, easy to learn and possibly easy to memorize. If an application such as an email client is hard to use or non-user friendly a user will tend avoid using the application at all cost. This is worse for game that is non-user friendly because the user will simply will not play the game again if he/she finds it difficult to use even if the game is fun. So it is critical that a usable menu system is made, in other words the menu system must be included in the game design. Though if done right the first time, more then likely you will be able to use the same menu framework for other future game development.

Menu Look and Feel

Well you basically have two options, option one use the high level GUI components implemented in MIDP or implement your own customized graphically menu system. There are disadvantages and advantages in using either option.

High Level GUI Component

Advantages	Disadvantage
Easier to implement	Not visual appealing
Conforms with native mobile handset look and feel	Unable to do certain effects such as rollover image change effects
Possibly less code thus small overall jar size	More then likely the user will have to scroll down for further options. It is highly encouraged to prevent the user from scrolling
High Level GUI is more portable from model to model and manufacturer to manufacturer	
Easy to maintain and change	
Easy to replicate / keep consistent in other games	

Customized Graphic Menu

Advantages	Disadvantage
Visually more appealing	More design and implementation
Re-enforces company branding	More resources, ie: may require my graphics to be done
Re-enforces client branding	Possibly more code, thus large overall jar size
More control and customization. For example small fonts and closer font spacing helps prevent the menu from expanding beyond one screen; therefore the user doesn't have to scroll	If over customized to fit a certain *look and feel* the menu may not be re-useable in another game
	May require more resources to when changes are made to the menu, ie new icons are needed for new menu options.

Basic Main Menu

The following diagram best illustrates a very basic menu flow.

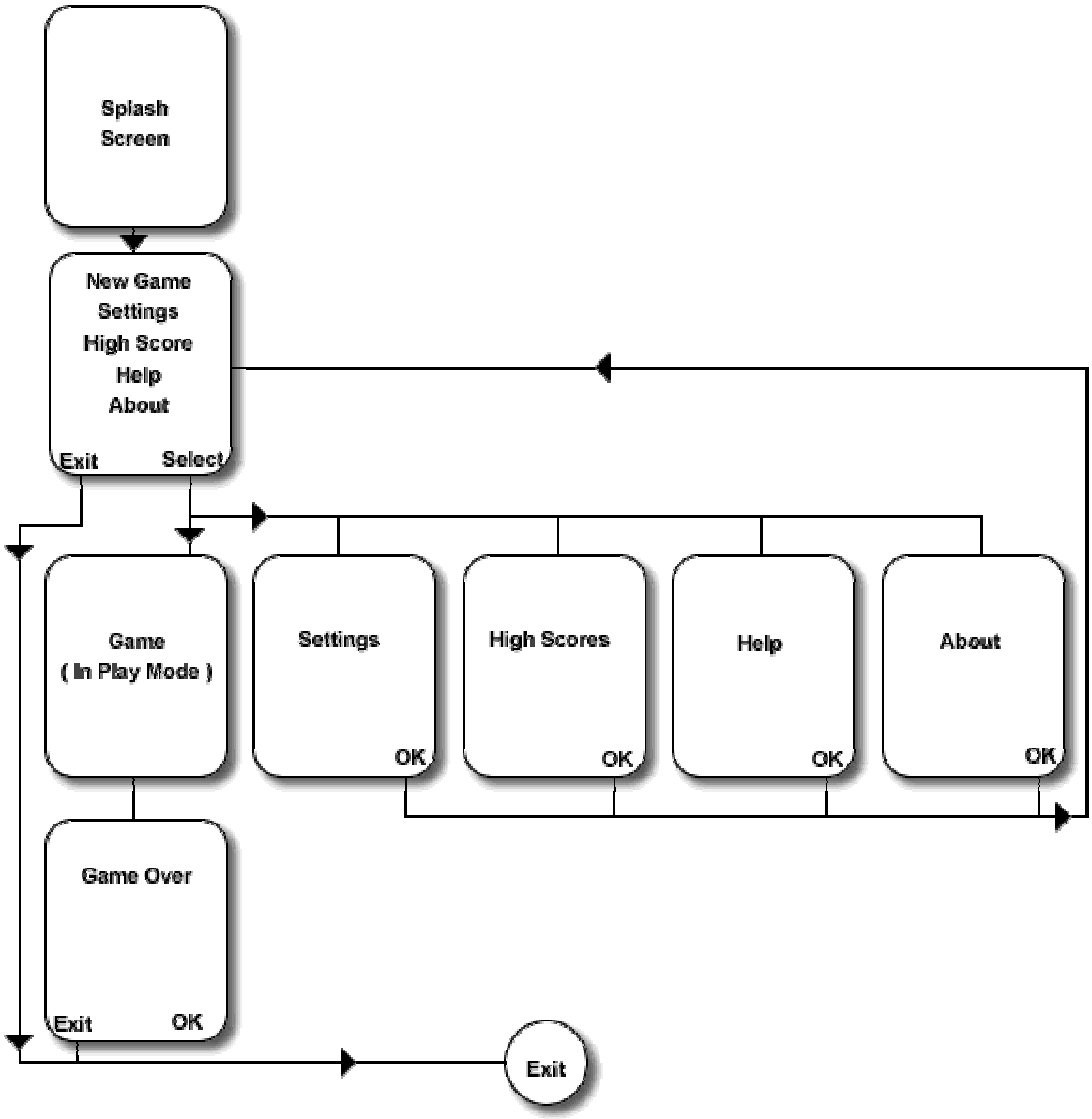


Figure 25 - Basic Menu Flow Diagram

Settings, *High Scores*, *Help*, and *About* in this example are Alert boxes. Though it might make sense for *Help* and *About* to use an Alert box it is most likely not logical to use an Alert box for *Settings* or *High Scores*. This is done for simplicity and not to over whelm you with the entire menu system all at once. The next section uses sub menus, which both Settings and High Scores would use. Though a simple game like Tic-Tac-Toe without the option to view *High Scores* nor the ability to set any *Settings* this basic menu flow would be satisfactory.

Basic Menu Source Code:

```
import javax.microedition.lcdui.*;

public class MainMenuScreen  extends List implements CommandListener {
  private Eliminator midlet;

  private Command selectCommand = new Command("Select", Command.ITEM,1);
  private Command exitCommand = new Command("Exit", Command.EXIT,1);

  private Alert alert;

  public MainMenuScreen(Eliminator midlet) {
    super("Eliminator",Choice.IMPLICIT);
    this.midlet = midlet;
    append("New Game",null);
    append("Settings",null);
    append("High Scores", null);
    append("Help",null);
    append("About",null);
    addCommand(exitCommand);
    addCommand(selectCommand);
    setCommandListener(this);
  }

  public void commandAction(Command c, Displayable d) {
    if (c == exitCommand) {
      midlet.mainMenuScreenQuit();
      return;
    } else if (c == selectCommand) {
      processMenu();  return;
    } else {
      processMenu();  return;
    }
  }

  private void processMenu() {
    try {
       List down = (List)midlet.display.getCurrent();
       switch (down.getSelectedIndex()) {
         case 0: scnNewGame(); break;
         case 1: scnSettings(); break;
         case 2: scnHighScores(); break;
         case 3: scnHelp(); break;
         case 4: scnAbout(); break;
```

```
    };
  } catch (Exception ex) {
    // Proper Error Handling should be done here
    System.out.println("processMenu::"+ex);
  }
}

private void scnNewGame() {
  midlet.mainMenuScreenShow(null);
}

private void scnSettings() {
  alert = new Alert("Settings"
                ,"Settings....."
                ,null
                ,null);
  alert.setTimeout(Alert.FOREVER);
  alert.setType(AlertType.INFO);
  midlet.mainMenuScreenShow(alert);
}

private void scnHighScores() {
  alert = new Alert("High Scores"
                ,"High Scores....."
                ,null
                ,null);
  alert.setTimeout(Alert.FOREVER);
  alert.setType(AlertType.INFO);
  midlet.mainMenuScreenShow(alert);
}

private void scnHelp() {
  alert = new Alert("Help"
                ,"Help....."
                ,null
                ,null);
  alert.setTimeout(Alert.FOREVER);
  alert.setType(AlertType.INFO);
  midlet.mainMenuScreenShow(alert);
}

private void scnAbout() {
  alert = new Alert("About"
                ,"Eliminator\nVersion 1.0.0\nby Jason Lam"
                ,null
                ,null);
```

```
    alert.setTimeout(Alert.FOREVER);
    alert.setType(AlertType.INFO);
    midlet.mainMenuScreenShow(alert);
  }
}
```

Main Midlet Source Code:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Eliminator extends MIDlet {
  protected Display display;

  private Image splashLogo;
  private boolean isSplash = true;

  MainMenuScreen mainMenuScreen;

  public Eliminator() {
  }

  public void startApp() {
    display = Display.getDisplay(this);
    mainMenuScreen = new MainMenuScreen(this);
    if(isSplash) {
      isSplash = false;
      try {
        splashLogo = Image.createImage("/splash.png");
        new SplashScreen(display, mainMenuScreen, splashLogo,3000);
      } catch(Exception ex) {
        mainMenuScreenShow(null);
      }
    } else {
      mainMenuScreenShow(null);
    }

  }

  public Display getDisplay() {
    return display;
  }

  public void pauseApp() {
  }
```

```
  public void destroyApp(boolean unconditional) {
    System.gc();
    notifyDestroyed();
  }

  private Image createImage(String filename) {
    Image image = null;
    try {
      image = Image.createImage(filename);
    } catch (Exception e) {
    }
    return image;
  }

  public void mainMenuScreenShow(Alert alert) {
    if (alert==null)
      display.setCurrent(mainMenuScreen);
    else
      display.setCurrent(alert,mainMenuScreen);
  }

  public void mainMenuScreenQuit() {
    destroyApp(true);
  }
}
```

Notice the Main Midlet is the one that calls the Splash Screen. As well the Main Midlet controls all screens, this will become more apparent in the next section, *Main Menu with Sub Menus*. The one thing that may seem odd is the following:

```
  public void mainMenuScreenShow(Alert alert) {
    if (alert==null)
      display.setCurrent(mainMenuScreen);
    else
      display.setCurrent(alert,mainMenuScreen);
  }
```

Why not just the following:

```
  public void mainMenuScreenShow() {
      display.setCurrent(alert,mainMenuScreen);
  }
```

This alert is placed there incase you want to output any prior messages before going the next screen. Another use would be say the user select High Score but an error occurred during the load of high scores, you can now output an alert error message and have the screen display go back to the main menu.

Basic Menu Screen Shot:

Figure 26 - Basic Menu Screen Shot

Main Menu with Sub Menus

In this section we replaced all Alert messages for High Score, Help and About with form components. There are two sub menus where *Settings* uses a new List Component and *Game Over* is really the GameCanvas with additional menu items assigned the soft key. The screens are currently dummy screens; the implementation will be done in the coming chapters. The goal of this section is get the basic understanding of the menu flow for the game Eliminator. As well this menu can be used as is or modified for other games. The following diagram best illustrates the menu flow including submenus.

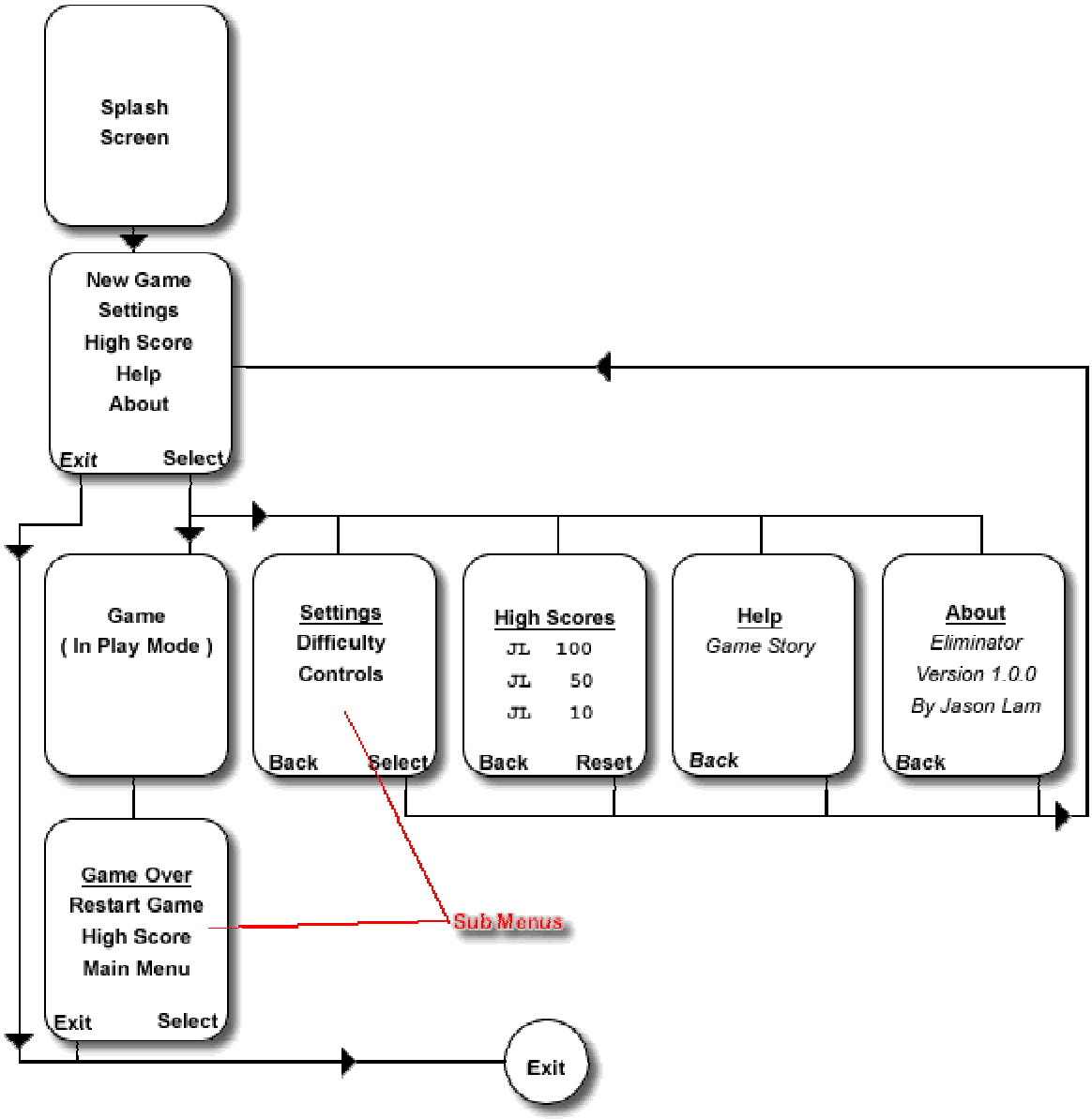


Figure 27 - Main Menu with Sub Menus Flow Diagram

The following is the stub source code for each the screens and changes need in the main midlet and main menu.

Main Midlet Source Code:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Eliminator extends MIDlet {
  protected Display display;
  private Image splashLogo;
  private boolean isSplash = true;

  private MainMenuScreen mainMenuScreen;
  private SettingsScreen settingsScreen;
  private HighScoreScreen highScoreScreen;
  private HelpScreen helpScreen;
  private AboutScreen aboutScreen;

  public Eliminator() {
  }

  public void startApp() {
    display = Display.getDisplay(this);
    mainMenuScreen = new MainMenuScreen(this);
    settingsScreen = new SettingsScreen(this);
    highScoreScreen = new HighScoreScreen(this);
    helpScreen = new HelpScreen(this);
    aboutScreen = new AboutScreen(this);
    if(isSplash) {
      isSplash = false;
      try {
        splashLogo = Image.createImage("/splash.png");
        new SplashScreen(display, mainMenuScreen, splashLogo,3000);
      } catch(Exception ex) {
        mainMenuScreenShow();
      }
    } else {
      mainMenuScreenShow();
    }

  }

  public Display getDisplay() {
    return display;
```

```
  }

  public void pauseApp() {
  }

  public void destroyApp(boolean unconditional) {
    System.gc();
    notifyDestroyed();
  }

  private Image createImage(String filename) {
    Image image = null;
    try {
      image = Image.createImage(filename);
    } catch (Exception e) {
    }
    return image;
  }

  public void mainMenuScreenShow() {
    display.setCurrent(mainMenuScreen);
  }

  public void settingsScreenShow() {
    display.setCurrent(settingsScreen);
  }

  public void highScoreScreenShow() {
    display.setCurrent(highScoreScreen);
  }

  public void helpScreenShow() {
    display.setCurrent(helpScreen);
  }

  public void aboutScreenShow() {
    display.setCurrent(aboutScreen);
  }

  public void mainMenuScreenQuit() {
    destroyApp(true);
  }
}
```

Main Menu with Sub Menus Source Code:

```
import javax.microedition.lcdui.*;

public class MainMenuScreen  extends List implements CommandListener {
  private Eliminator midlet;
  private Command selectCommand = new Command("Select", Command.ITEM,1);
  private Command exitCommand = new Command("Exit", Command.EXIT,1);

  public MainMenuScreen(Eliminator midlet) {
    super("Eliminator",Choice.IMPLICIT);
    this.midlet = midlet;
    append("New Game",null);
    append("Settings",null);
    append("High Scores", null);
    append("Help",null);
    append("About",null);
    addCommand(exitCommand);
    addCommand(selectCommand);
    setCommandListener(this);
  }

  public void commandAction(Command c, Displayable d) {
    if (c == exitCommand) {
      midlet.mainMenuScreenQuit();
    } else if (c == selectCommand) {
      processMenu();
    } else {
      processMenu();
    }
  }

  private void processMenu() {
    try {
       List down = (List)midlet.display.getCurrent();
       switch (down.getSelectedIndex()) {
         case 0: scnNewGame(); break;
         case 1: scnSettings(); break;
         case 2: scnHighScore(); break;
         case 3: scnHelp(); break;
         case 4: scnAbout(); break;
       };
    } catch (Exception ex) {
      // Proper Error Handling should be done here
      System.out.println("processMenu::"+ex);
```

```
    }
  }

  private void scnNewGame() {
    midlet.mainMenuScreenShow();
  }

  private void scnSettings() {
    midlet.settingsScreenShow();
  }

  private void scnHighScore() {
    midlet.highScoreScreenShow();
  }

  private void scnHelp() {
    midlet.helpScreenShow();
  }

  private void scnAbout() {
    midlet.aboutScreenShow();
  }
}
```

Settings Screen Source Code:

```
import javax.microedition.lcdui.*;

public class SettingsScreen extends List implements CommandListener {
  private Eliminator midlet;
  private Command backCommand = new Command("Back", Command.BACK, 1);
  private Command selectCommand = new Command("Select", Command.SCREEN,1);

  public SettingsScreen (Eliminator midlet) {
    super("Settings",Choice.IMPLICIT);
    append("Difficulty",null);
    append("Controls",null);
    this.midlet = midlet;
    addCommand(backCommand);
    addCommand(selectCommand);
    setCommandListener(this);
  }
  public void commandAction(Command c, Displayable d) {
    if (c == backCommand) {
      midlet.mainMenuScreenShow();
    } else if (c == selectCommand) {
      processMenu();
    } else {
      processMenu();
    }
  }
  private void processMenu() {
    try {
       List down = (List)midlet.display.getCurrent();
       switch (down.getSelectedIndex()) {
         case 0: setDifficulty(); break;
         case 1: setControls(); break;
       };
    } catch (Exception ex) {
      // Proper Error Handling should be done here
      System.out.println("processMenu::"+ex);
    }
  }
  private void setDifficulty() {
    System.out.println("Difficutly Settings Not Implemented Yet");
  }
  private void setControls() {
    System.out.println("Controls Settings Not Implemented Yet");
  }
}
```

High Score Screen Source Code:

```
import javax.microedition.lcdui.*;

public class HighScoreScreen extends Form implements CommandListener {
  private Eliminator midlet;
  private Command backCommand = new Command("Back", Command.BACK, 1);
  private Command resetCommand = new Command("Rest", Command.SCREEN,1);

  public HighScoreScreen (Eliminator midlet) {
    super("High Score");
    this.midlet = midlet;
    StringItem stringItem = new StringItem(null,"JL    100\nJL      50\nJL      10");
    append(stringItem);
    addCommand(backCommand);
    addCommand(resetCommand);
    setCommandListener(this);
  }

  public void commandAction(Command c, Displayable d) {
    if (c == backCommand) {
      midlet.mainMenuScreenShow();
      return;
    }
    if (c == resetCommand) {
      // not implemented yet
      System.out.println("Reset High Scores Not Implemented Yet");
    }
  }
}
```

Help Screen Source Code:

```
import javax.microedition.lcdui.*;

public class HelpScreen extends Form implements CommandListener {
  private Eliminator midlet;

  private Command backCommand = new Command("Back", Command.BACK, 1);

  public HelpScreen (Eliminator midlet) {
    super("Help");
    this.midlet = midlet;
    StringItem stringItem = new StringItem(null,
    "It is the year 3023, many things have changed over the years " +
    "including the Great Migration to Earth II.  " +
    "But the one thing that hasn't changed is the struggle for " +
    "Power and Wealth.  You are one the last remaning hopes " +
    "to defend Earth II from total dominance by Evil Ruler named " +
    "Zikron.  You mission is to eliminate Zikron and his " +
    "minions from the face of the Earth II"
                                        );
    append(stringItem);
    addCommand(backCommand);
    setCommandListener(this);
  }

  public void commandAction(Command c, Displayable d) {
    if (c == backCommand) {
      midlet.mainMenuScreenShow();
      return;
    }
  }
}
```

About Screen Source Code:

```
import javax.microedition.lcdui.*;

public class AboutScreen extends Form implements CommandListener {
  private Eliminator midlet;
  private Command backCommand = new Command("Back", Command.BACK, 1);

  public AboutScreen (Eliminator midlet) {
    super("About");
    this.midlet = midlet;
    StringItem stringItem = new StringItem(null,"Eliminator\nVersion 1.0.0\nBy Jason
Lam");
    append(stringItem);
    addCommand(backCommand);
    setCommandListener(this);
  }

  public void commandAction(Command c, Displayable d) {
    if (c == backCommand) {
      midlet.mainMenuScreenShow();
      return;
    }
  }
}
```

Main Menu with Sub Menus Screen Shot:

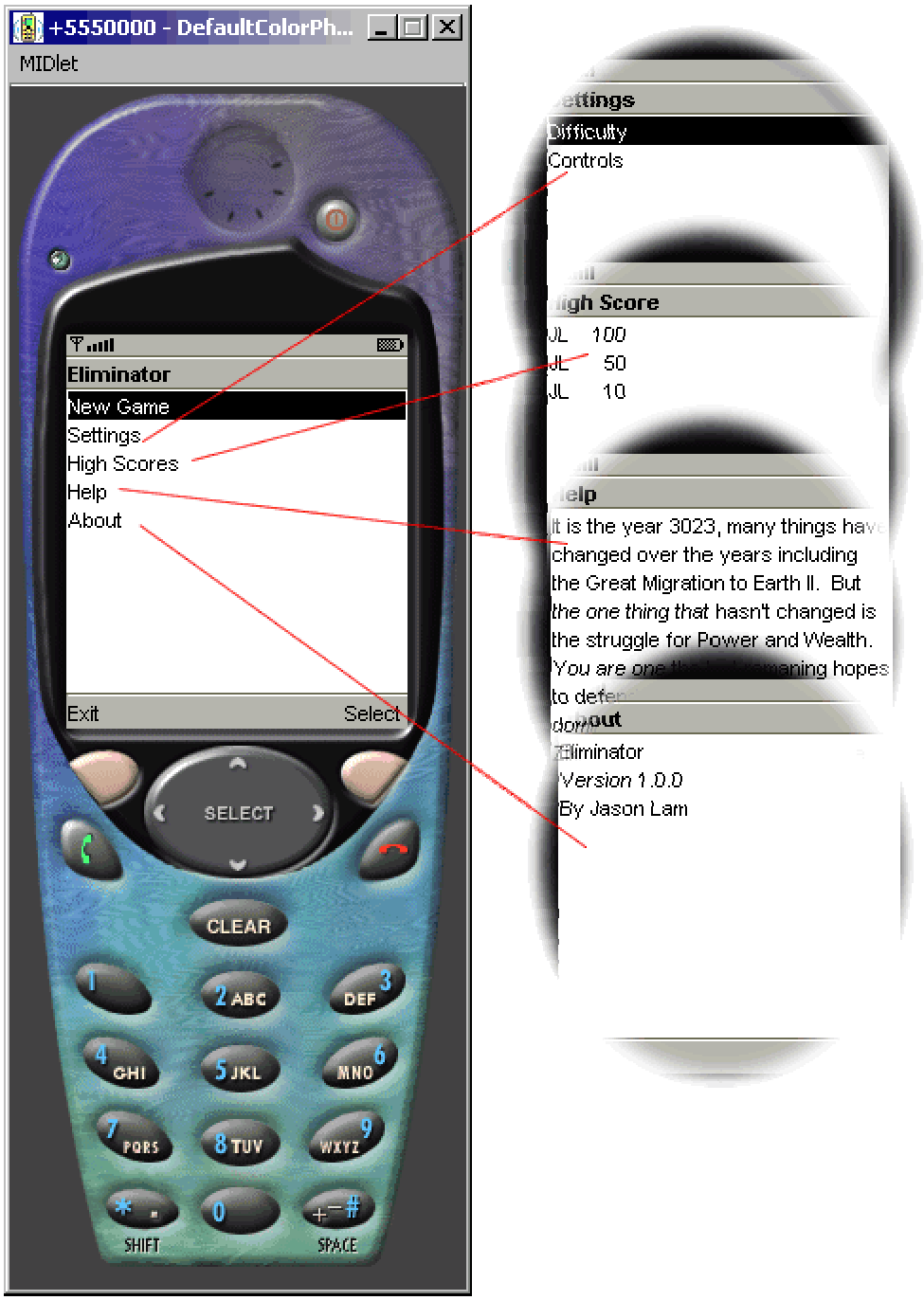


Figure 28- Main Menu wiht Sub Menus Screen Shot

Menu Suggestions and Enhancements

The previous sections are only suggested solutions for presenting a menu. There are many ways of doing this but always keep in mind usability. Some suggested enhancements or changes you may want to apply to the above menu code are:

1. For Splash Screen you may want to add soft key buttons with the options to Skip to make it more obvious that user can skip the splash screen. As well add an Exit button incase for any reason the user needs to exit immediately.

2. The soft buttons in the above example may or may not be appropriate for certain mobile handsets. You should consult the carrier, manufacture or aggregator you are going to deploy the game to about the correct labeling of buttons, menus and positioning of buttons. For example, the above code has the *Back* and *Exit* button the left soft button but for the Nokia Series 40 the *Back* and *Exit* button are often found on the right soft button. Another example is some aggregators prefer the word *Options* over *Settings*.

3. You've probably noticed the menu system processing is done in procedure like approach. Given that menu options probably will not extend beyond what is listed in the above examples this maybe satisfactory. In fact there should not be that many game options because this forces the user to scroll and adds frustration. However, to make your game more robust and maintainable you may consider using patterns and techniques such as the MVC pattern, Command Pattern and/or Reflection for menu processing. But do not forgot by adding in more robust code the trade off is the use of more memory and we haven't even started coding the game yet.

   Currently, if you need to add or remove a menu item you would have to edit Elminator.java, MainMenuScreen.java and add or remove the *[name]*Screen.java file. If done right all you would have to do is add or *[name]*Screen.java file.

4. If you decide not to use the high-level GUI and decide to a customized menu the flow the menu should not change much. Once again make sure it is easy to learn and easy to use.

Summary

You should now have fairly good grasp of both the flow of menu options and what the source code looks like for the menu system. Once again keep in mind the menu is a very important factor that makes up a game or any other application. The menu should be efficient, easy to use, easy to learn and possibly easy to memorize. Not only should the game be fun; but also, getting there should be painless.

Chapter 9 – Eliminator: Exception Handling

As we add more and more code to complete the final version of Eliminator we need some support features built into the program such as handling errors and exceptions. There are numerous ways of doing this. For simplicity we are going to have the main midlet handle all exceptions thrown. In other words all other classes need propagate the thrown exception.

We are only covering some the more trivial exceptions that can be thrown throughout the game execution. You will have to do you own extensive research to handle all possible exception before deploying to production. Exception handling is beyond the scope of this book.

Please refer to sample code in Chapter 9 for the updated source code. The primary change is in the main midlet the code now displays an alert box when needed. In this case the user then would see the error message before the main menu appears. Of course the assumption here is that main menu doesn't have a problem. Anywhere where the game gets an exception whether it is loading *High Scores*, while game is in Play or user is updating *Settings* the user is shown the error message and is brought back the main menu.

Alert Method Source Code:

```
  protected void showErrorMsg(String alertMsg) {
    if (alertMsg == null || CONST_DEBUG == false) {
      alertMsg = "Error Starting Elminator, may or may not function correctly.  Please
contact support.";
    }
    alert = new Alert("Eliminator ERROR",alertMsg,null,null);
    alert.setTimeout(Alert.FOREVER);
    alert.setType(AlertType.ERROR);
    this.mainMenuScreenShow(alert);
  }
```

Updated Main Menu Show Method Source Code:

```
  protected void mainMenuScreenShow(Alert alert) {
    if (alert==null)
      display.setCurrent(mainMenuScreen);
    else
      display.setCurrent(alert,mainMenuScreen);
  }
```

You will notice in the method showErrorMsg the is a check for CONST_DEBUG == false, this enables you to output customized developer messages for development but production all you have to do is change the boolean value and the user gets the generic message. Of course, there is debate whether or not you should output meaning messages

to the user such as error codes, this will have to be decided by the game developer or as instructed by the manufacture, carrier or aggregator.

Exception Example Screen Shot:
In this example, to display the error message the *splash.png* file was removed from the jar file.

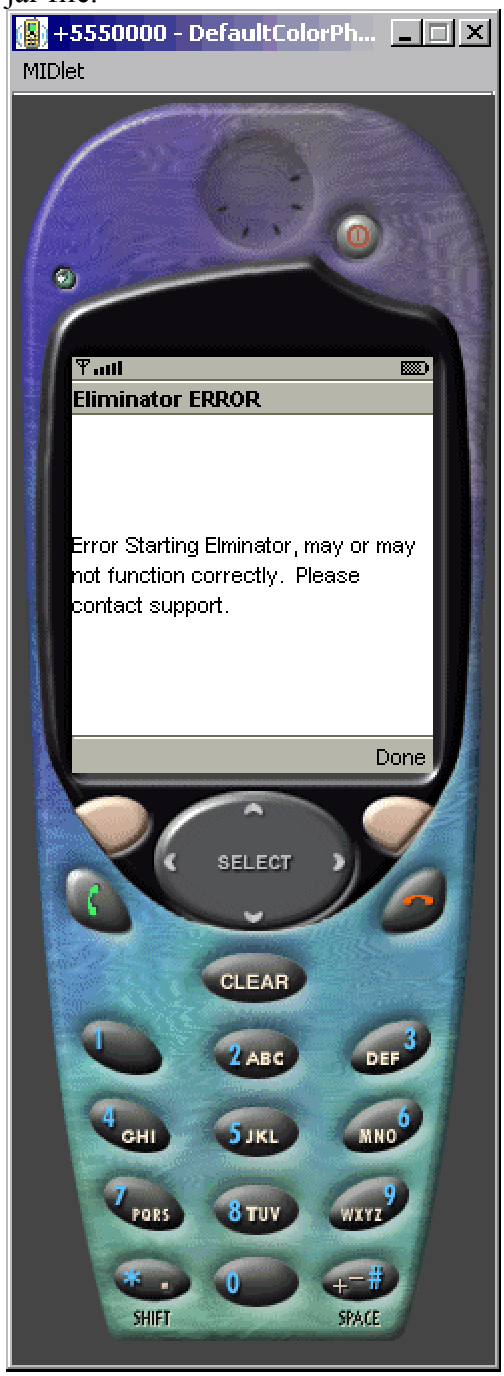


Figure 29 - Eliminator Error Screen Shot

Chapter 10 – Eliminator: Settings & High Score

Overview

There are few more things we need to setup before we get to coding the game itself this includes game difficulty settings, control setting and high score. All of which require persistence data in other words we need to use the Record Management System, RMS. If you are not familiar with RMS, please do so before continue with the rest of the chapter.

Both settings and high score will be stored in two separate recordstores. For simplicity control settings will only have one option, the option to set auto fire on or off. Because there is only one option the setting submenu will be changed to have the following to options *Difficulty* and *Auto Fire* instead of *Difficulty* and *Controls*.

A side note for demonstration purposes we've included the option to set difficulty; however, in general when there is an option to set game modes there truly should be a significant value added when such an option is given to a user. But as mentioned before the focus of these chapters is demonstrate more what a J2ME game code looks like rather then what contributes to making a successful game.

RMS Data Structure

The storage format for Settings will be two fields type integer making up one record/rows.

Name	Data Type	Value	Comment
Difficulty	integer	1,2 or 3	1 = easy, 2 = medium, 3 = hard
Auto Fire	integer	1 or 0	1 = on, 0 = off

The storage format for High Score will be 2 fields making up 3 records/rows. Where record 1 is the 1st place, 2nd record is 2nd place and the 3rd record is 3rd place.

Name	Data Type	Value	Comment
High Score Initials/Name	String	3 Char	
High Score Value	integer		

Settings and High Score Design

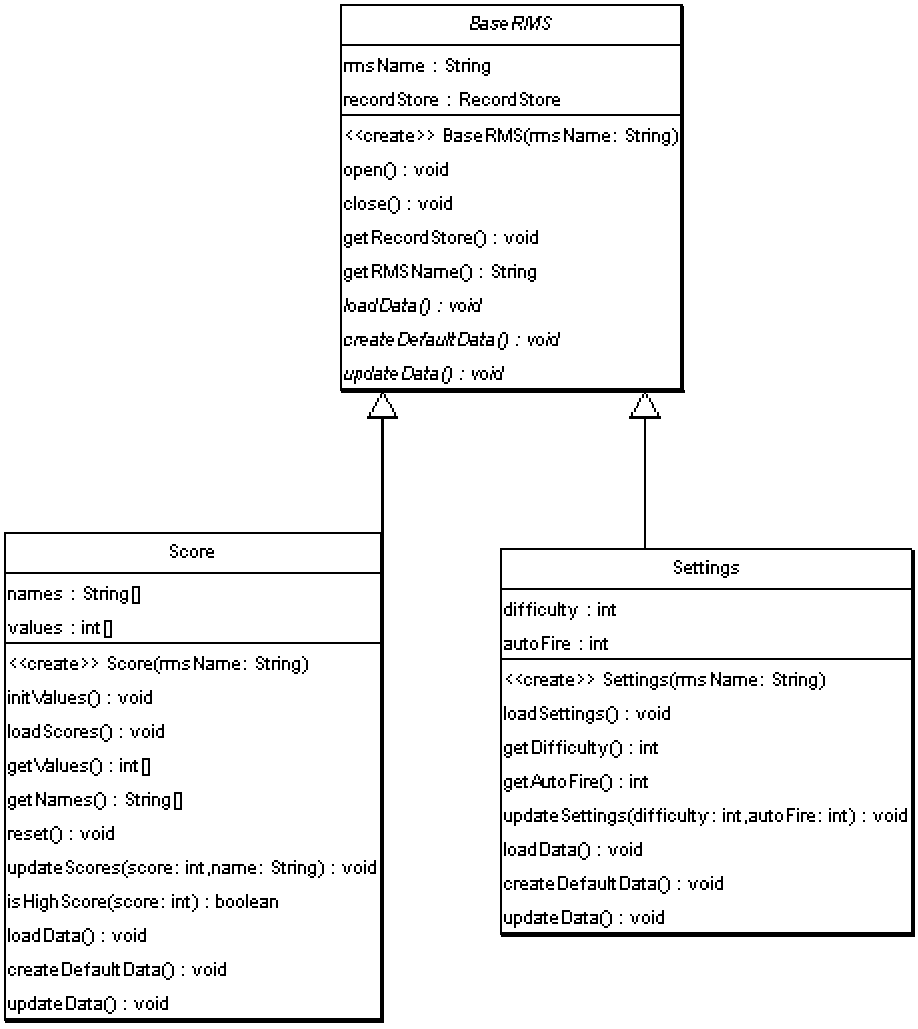


Figure 30 – Settings and High Score UML Class diagrams

Because there are common functionalities between Settings and High Score, a possible design solution is to use inheritance. This not only allows you to easily add addition sub classes for any future features that may need to use RMS; but also, allows you to easily reuse selected components in other projects. For example, you wanted to make another game that does require the use of Settings. All you have to do is copy the BaseRMS and Score class and simply do not use the Settings Class. This also allows you to simply edit or totally replace certain components without affecting other areas of the code.

Though all the functionality could have easily been done in one class instead of three it is well worth the trade off for more code maintainability versus gain in memory space.

Source Code

There are three new source files as indicated in the class diagram BaseRMS.java, Score.java and Settings.java. As well Eliminator.java, SettingsScreen.java and HighScore.java have been updated appropriately to use the Settings and Score. The remaining classes have been unchanged.

See Chapter 10 resources for the complete source code

BaseRMS

BaseRMS is the Base class or super class to handle RMS functionality. It implements two methods open and close. There are three other methods that are defined abstract. Close is pretty straight forward it closes the connection. The open method does more then more then open the recordstore; if it is unable to find an existing recordstore it goes out and creates a default recordstore. If it finds an existing recordstore, the recordstore is loaded.

```
import javax.microedition.rms.*;
import java.util.*;
import java.io.*;

abstract public class BaseRMS {
  private String rmsName;
  private RecordStore recordStore;

  BaseRMS(String rmsName) {
    this.rmsName = rmsName;
  }

  public void open() throws Exception {
    try {
      recordStore = RecordStore.openRecordStore(this.rmsName,true);
      if (recordStore.getNumRecords() > 0) {
        loadData();
      } else {
        createDefaultData();
      }
    } catch (Exception e) {
      throw new Exception(this.rmsName+"::open::"+e);
    }
  }

  public void close() throws Exception {
```

```
    if (recordStore != null) {
      try {
        recordStore.closeRecordStore();
      } catch(Exception e) {
          throw new Exception(this.rmsName+"::close::"+e);
      }
    }
  }

  public RecordStore getRecordStore() {
    return this.recordStore;
  }

  public String getRMSName() {
    return this.rmsName;
  }

  abstract void loadData() throws Exception;
  abstract void createDefaultData() throws Exception;
  abstract void updateData() throws Exception;
}
```

Score

Score is inherits BaseRMS and implements the abstract methods. The source code is fairly straight forward.

```
import javax.microedition.rms.*;
import java.util.*;
import java.io.*;

public class Score extends BaseRMS {
  private String[] names = new String[3];
  private int[] values = new int[3];

  public Score(String rmsName) {
    super(rmsName);
    initValues();
  }

  private void initValues() {
    names[0] = "JCL";
    names[1] = "YYK";
    names[2] = "RKL";
    values[0] = 100;
```

```
  values[1] = 50;
  values[2] = 10;
}

public void loadScores() throws Exception {
  try {
    // Will call either loadData() or createDefaultData()
    this.open();

    // Add any other neccessary processing, in this case there is none

    // Close
    if (this.getRecordStore() != null)
      this.close();
  } catch (Exception e) {
    throw new Exception("Error loading Scores" + e);
  }
}

public int[] getValues() {
  return this.values;
}

public String[] getNames() {
  return this.names;
}

public void reset() throws Exception {
  this.open();
  initValues();
  updateData();
  if (this.getRecordStore() != null)
    this.close();
}

public void updateScores(int score, String name) throws Exception {
  try {
    for (int i = 0; i < names.length; i++) {
      if (score >= values[i]) {
        // load current scores
        this.open();

        // Shift the score table.
        for (int j = names.length - 1; j > i; j--) {
          values[j] = values[j - 1];
          names[j] = names[j - 1];
```

```
        }

        // Insert the new score.
        this.values[i] = score;
        this.names[i] = name;

        // Update High Scores
        updateData();

        // close
        if (this.getRecordStore() != null)
         this.close();
        break;
       }
      }
    } catch (Exception e) {
      throw new Exception(this.getRMSName() + "::updateScores::" + e);
    }
  }

  public boolean isHighScore(int score) throws Exception {
    boolean isHighScore = false;
    try {
     for (int i = 0; i < names.length; i++) {
       if (score >= values[i])
         isHighScore = true;
     }
    } catch (Exception e) {
      throw new Exception(this.getRMSName() + "::isHighScore::" + e);
    }
    return isHighScore;
  }

  protected void loadData() throws Exception {
    try {
      for (int i = 0; i < names.length; i++) {
        byte[] record = this.getRecordStore().getRecord(i + 1);
        DataInputStream istream = new DataInputStream(new
ByteArrayInputStream(record,0,record.length));
        values[i] = istream.readInt();
        names[i] = istream.readUTF();
      }
    } catch (Exception e) {
      throw new Exception (this.getRMSName() + "::loadData::" + e);
    }
  }
```

```
  protected void createDefaultData() throws Exception {
    try {
      for (int i = 0; i < names.length; i++) {
        ByteArrayOutputStream bstream = new ByteArrayOutputStream(12);
        DataOutputStream ostream = new DataOutputStream(bstream);
        ostream.writeInt(values[i]);
        ostream.writeUTF(names[i]);
        ostream.flush();
        ostream.close();
        byte[] record = bstream.toByteArray();
        this.getRecordStore().addRecord(record, 0, record.length);
      }
    } catch (Exception e) {
      throw new Exception(this.getRMSName() + "::createDefaultData::" + e);
    }
  }

  protected void updateData() throws Exception {
    try {
      for (int i = 0; i < names.length; i++) {
        ByteArrayOutputStream bstream = new ByteArrayOutputStream(12);
        DataOutputStream ostream = new DataOutputStream(bstream);
        ostream.writeInt(values[i]);
        ostream.writeUTF(names[i]);
        ostream.flush();
        ostream.close();
        byte[] record = bstream.toByteArray();
        this.getRecordStore().setRecord(i + 1, record, 0, record.length);
      }
    } catch(Exception e) {
      throw new Exception(this.getRMSName() + "::updateData::" + e);
    }
  }
}
```

Settings

Settings is inherits BaseRMS and implements the abstract methods. The source code is fairly straightforward.

```
import javax.microedition.rms.*;
import java.util.*;
import java.io.*;

public class Settings extends BaseRMS {
  private int difficulty = 1;
  private int autoFire = 1;

  public Settings(String rmsName) {
    super(rmsName);
  }

  public void loadSettings() throws Exception {
    try {
      // Will call either loadData() or createDefaultData()
      this.open();

      // Add any other neccessary processing, in this case there is none

      // Close
      if (this.getRecordStore() != null)
        this.close();
    } catch (Exception e) {
      throw new Exception("Error loading Settings" + e);
    }
  }

  public int getDifficulty() {
    return this.difficulty;
  }

  public int getAutoFire() {
    return this.autoFire;
  }

  public void updateSettings(int difficulty,int autoFire) throws Exception {
    try {
      // load current scores
      this.open();
```

```
    // update data
    this.difficulty = difficulty;
    this.autoFire = autoFire;

    // Update High Scores
    updateData();

    // close
    if (this.getRecordStore() != null)
      this.close();
   } catch (Exception e) {
    throw new Exception(this.getRMSName() + "::updateSettings::" + e);
   }
  }

  protected void loadData() throws Exception {
   try {
      byte[] record = this.getRecordStore().getRecord(1);
      DataInputStream istream = new DataInputStream(new
ByteArrayInputStream(record,0,record.length));
      difficulty = istream.readInt();
      autoFire = istream.readInt();
   } catch (Exception e) {
    throw new Exception (this.getRMSName() + "::loadData::" + e);
   }
  }

  protected void createDefaultData() throws Exception {
   try {
    ByteArrayOutputStream bstream = new ByteArrayOutputStream(12);
    DataOutputStream ostream = new DataOutputStream(bstream);
    ostream.writeInt(difficulty);
    ostream.writeInt(autoFire);
    ostream.flush();
    ostream.close();
    byte[] record = bstream.toByteArray();
    this.getRecordStore().addRecord(record, 0, record.length);
   } catch (Exception e) {
    throw new Exception(this.getRMSName() + "::createDefaultData::" + e);
   }
  }

  protected void updateData() throws Exception {
   try {
    ByteArrayOutputStream bstream = new ByteArrayOutputStream(12);
```

```
      DataOutputStream ostream = new DataOutputStream(bstream);
      ostream.writeInt(difficulty);
      ostream.writeInt(autoFire);
      ostream.flush();
      ostream.close();
      byte[] record = bstream.toByteArray();
      this.getRecordStore().setRecord(1, record, 0, record.length);
    } catch(Exception e) {
      throw new Exception(this.getRMSName() + "::updateData::" + e);
    }
  }
}
```

Main Midlet

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Eliminator extends MIDlet {
  protected Display display;
  private Image splashLogo;
  private boolean isSplash = true;

  private MainMenuScreen mainMenuScreen;
  private SettingsScreen settingsScreen;
  private HighScoreScreen highScoreScreen;
  private HelpScreen helpScreen;
  private AboutScreen aboutScreen;

  private Alert alert;

  private static Score score;
  private static final String scoreRMSName = "EliminatorSettings";
  private static Settings settings;
  private static final String settingsRMSName = "EliminatorScore";

  private static final boolean CONST_DEBUG = true;

  public Eliminator() {
  }

  public void startApp() {
    display = Display.getDisplay(this);
    if(isSplash) {
      isSplash = false;
```

```
    try {


      settings = new Settings(settingsRMSName);
      settings.loadSettings();

      score = new Score(scoreRMSName);
      score.loadScores();


      mainMenuScreen = new MainMenuScreen(this);
      settingsScreen = new SettingsScreen(this,settings);
      highScoreScreen = new HighScoreScreen(this,score);
      helpScreen = new HelpScreen(this);
      aboutScreen = new AboutScreen(this);


      splashLogo = Image.createImage("/splash.png");
      new SplashScreen(display, mainMenuScreen, splashLogo,3000);
    } catch(Exception ex) {
      showErrorMsg(null);
    }
  } else {
    mainMenuScreenShow(null);
  }

}

public Display getDisplay() {
  return display;
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional) {
  System.gc();
  notifyDestroyed();
}

private Image createImage(String filename) {
  Image image = null;
  try {
    image = Image.createImage(filename);
  } catch (Exception e) {
```

```
    }
    return image;
  }

  protected void mainMenuScreenShow(Alert alert) {
    if (alert==null)
      display.setCurrent(mainMenuScreen);
    else
      display.setCurrent(alert,mainMenuScreen);
  }

  protected void settingsScreenShow() {
    try {
      settingsScreen.init();
      display.setCurrent(settingsScreen);
    } catch (Exception e) {
      showErrorMsg(null);
    }

  }

  protected void highScoreScreenShow() {
    try {
      highScoreScreen.init();
      display.setCurrent(highScoreScreen);
    } catch (Exception e) {
      showErrorMsg(null);
    }
  }

  protected void helpScreenShow() {
    display.setCurrent(helpScreen);
  }

  protected void aboutScreenShow() {
    display.setCurrent(aboutScreen);
  }

  protected void mainMenuScreenQuit() {
    destroyApp(true);
  }

  protected void showErrorMsg(String alertMsg) {
    if (alertMsg == null || CONST_DEBUG == false) {
      alertMsg = "Error Starting Elminator, may or may not function correctly.  Please
contact support.";
```

```
    }
    alert = new Alert("Eliminator ERROR",alertMsg,null,null);
    alert.setTimeout(Alert.FOREVER);
    alert.setType(AlertType.ERROR);
    this.mainMenuScreenShow(alert);
  }
}
```

Settings Screen

Previously SettingsScreen.java inherited the List component but because the options on the Settings screen involve 2 different lists of options it has been changed to inherit a Form component. This form component includes appropriate buttons and choices for the both Settings and Auto Fire.

```
import javax.microedition.lcdui.*;

public class SettingsScreen extends Form implements CommandListener {
  private Eliminator midlet;
  private Command backCommand = new Command("Back", Command.BACK, 1);
  private Command okCommand = new Command("OK", Command.SCREEN,1);


  private Settings settings;
  private static final String[] difficultyOptions = {"Easy","Medium","Hard"};
  private static final String[] autoFireOptions = {"Off","On"};
  private ChoiceGroup difficulty;
  private ChoiceGroup autoFire;


  public SettingsScreen (Eliminator midlet, Settings settings) throws Exception {
    super("Settings");
    this.midlet = midlet;
    this.settings = settings;
    difficulty = new ChoiceGroup("Difficulty", ChoiceGroup.POPUP,
difficultyOptions,null);
    autoFire = new ChoiceGroup("Auto Fire", ChoiceGroup.POPUP,
autoFireOptions,null);
    append(difficulty);
    append(autoFire);
    addCommand(backCommand);
    addCommand(okCommand);
    setCommandListener(this);
  }

  public void commandAction(Command c, Displayable d) {
```

```
    if (c == backCommand) {
      midlet.mainMenuScreenShow(null);
    } else if (c == okCommand) {
      processMenu();
    }
  }

  public void init() throws Exception {
    settings.loadSettings();
    difficulty.setSelectedIndex(settings.getDifficulty()-1,true);
    autoFire.setSelectedIndex(settings.getAutoFire(),true);
  }

  private void processMenu() {
    try {
      settings.updateSettings(difficulty.getSelectedIndex()+1,autoFire.getSelectedIndex());
      midlet.mainMenuScreenShow(null);
    }  catch (Exception ex) {
      midlet.showErrorMsg("null");
    }
  }
}
```

High Score Screen

```
import javax.microedition.lcdui.*;

public class HighScoreScreen extends Form implements CommandListener {
  private Eliminator midlet;
  private Command backCommand = new Command("Back", Command.BACK, 1);
  private Command resetCommand = new Command("Reset", Command.SCREEN,1);
  private Score score;
  StringItem stringItem;

  public HighScoreScreen (Eliminator midlet,Score score) throws Exception {
    super("High Score");
    this.midlet = midlet;
    this.score = score;
    stringItem = new StringItem(null,"");
    append(stringItem);
    addCommand(backCommand);
    addCommand(resetCommand);
    setCommandListener(this);
  }
```

```
  public void commandAction(Command c, Displayable d) {
    if (c == backCommand) {
      midlet.mainMenuScreenShow(null);
      return;
    }
    if (c == resetCommand) {
      processMenu();
    }
  }

  public void init() throws Exception {
    score.loadScores();
    stringItem.setText(buildHighScore());
  }

  private void processMenu() {
    try {
      score.reset();
      midlet.mainMenuScreenShow(null);
    }  catch (Exception ex) {
      midlet.showErrorMsg("null");
    }
  }

  private String buildHighScore() {
    String result = "";
    String[] names = score.getNames();
    int[] values = score.getValues();
    for (int i=0; i<names.length; i++) {
      result = result + names[i] + "    " + values[i] + "\n";
    }
    return result;
  }
}
```

Screen Shot

Settings screen

Figure 31 - Settings Screen Shot

High Score Screen Shot

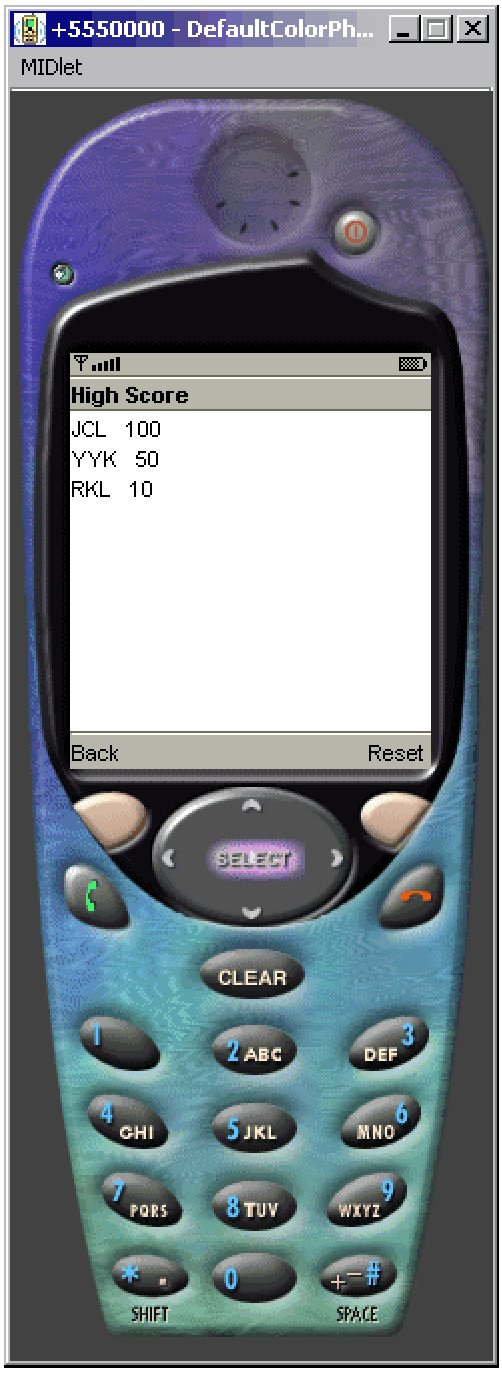


Figure 32 – High Score Screen Shot

Chapter 11 – Eliminator: Terrain (Scrolling Background)

Overview

Finally, we get to start developing the main part, the game itself. Let us start off with the background scenery.

As stated earlier Eliminator is a vertical shooter, to make the game visually more appealing the background is dynamic, in other words it scrolls. You basically have two choices either a random generated background that really serves no other purpose but visual effects or a background that contributes to the game itself. An example of a random generated background is a random set of falling stars. A background with more purpose would be a clearly defined background that is the same every time the user plays that game/stage. This background can act as indicator to the user how he/she progresses throughout the game and allows you to progressively place certain items or enemies at certain locations throughout the background. You can think of this more like a map.

It probably isn't the best choice to simply insert a large graphic background for obvious memory constraints mobile devices have. This is where the TiledLayer comes in handy to produce large levels made up of re-useable graphic tiles.

Keep mind now that we are now developing the game itself we will be using the concepts explained in Chapter 4.

GameScreen Itself

Because this is the first chapter the deals with the GameScreen we will briefly go over the changes needed to invoke this screen, which really is not any different from how the other screens are called from the main menu. See Chapter 11 source code for the complete listing.

Eliminator.java

In the main midlet, Elminator.java the GameScreen object is added

```
private GameScreen gameScreen;
```

As well the method used to invoke a new game. A new GameScreen is instantiated passing in the midlet itself, settings object and the score object. At the same time the game thread is started, start().

```
protected void gameScreenShow() {
  try {
    gameScreen = null;
    gameScreen = new GameScreen(this,settings,score);
    gameScreen.start();
    display.setCurrent(gameScreen);
  } catch (Exception e) {
        System.out.println(e);
    showErrorMsg(null);
  }
}
```

MainMenuScreen.java

The new game option now calls the gameScreenShow method in main midlet.

```
private void scnNewGame() {
  midlet.gameScreenShow();
}
```

GameScreen.java

The GameScreen itself consists of the basic game loop and game components. It inherits the GameCanvas class, explained in more detail back in Chapter 4. It implements the Runnable interface to provide a thread for main game loop as well it implements CommandListener for soft button feedback

The main loop looks like the following:

```
// Main Game Loop
public void run() {
  Graphics g = getGraphics();

  Thread currentThread = Thread.currentThread();

  try {
   while (currentThread == gameThread) {
        long startTime = System.currentTimeMillis();
        if (isShown()) {
          if (isPlay) {
            tick();
          }
       render(g);
        }
        long timeTake = System.currentTimeMillis() - startTime;
        if (timeTake < MILLIS_PER_TICK) {
          synchronized (this) {
            wait(MILLIS_PER_TICK - timeTake);
          }
        } else {
          currentThread.yield();
        }
   }
  } catch (InterruptedException ex) {
    // won't be thrown
  }

}
```

The main area where graphics are drawn looks like the following:

```
// Method to Display Graphics
private void render(Graphics g) {

  // Set Background color to beige
  g.setColor(0xF8DDBE);
  g.fillRect(0,0,width,height);
  g.setColor(0x0000ff);

  layerManager.paint(g,0,0);

  flushGraphics();
}
```

The main area where graphics are update, such as movement and scrolling:

```
// Handle dynamic changes to game including user input
public void tick() {
  // update code goes here
}
```

Though these are essential pieces to a game there really isn't much to show if there isn't anything to animate, draw or execute. In the next section you will be able to see the GameScreen in action when it animates the scrolling background terrain.

Simple One Terrain

Overview

Using a TiledLayer isn't anything new we haven't explained in Chapter 4, the scrolling effect itself is simply the adjustment to the relative position of where to show the TiledLayer image; this is done by using the setPosition method.

Constructor

In the constructor we make a call to the initTerrain method and add the terrain to the LayerManager

```
initTerrain();

layerManager = new LayerManager();
layerManager.append(terrain);
```

Initialize Terrain

The initTerrain Method loads the TiledLayer managing the terrain and sets the initial TiledLayer position based on the height of the tile, number of tiles and the screen height view port.

```
// Init Terrain and set Terrain at the bottom of the map
private void initTerrain() throws Exception {
  try {
    terrain = loadTerrain();
    terrainScroll = 1 - (TILE_HEIGHT * terrain.getRows()) + scnViewHeight;
    terrain.setPosition(0,terrainScroll);
  } catch (Exception e) {
    throw new Exception("GameScreen::initTerrain::" + e);
  }
}
```

Load Terrain

You probably noticed in the last section mapping of the TiledLayer with graphics wasn't really done in that method but instead in the loadTerrain() method. A nature question maybe why separate this to a different task, with a few adjustments to this method it will be more clear why we have done this. These adjustments will be illustrated in the next section with multiple terrains.

```
  private TiledLayer loadTerrain() throws Exception {
    Image tileImages = Image.createImage("/terrain.png");
    TiledLayer tiledLayer = new
TiledLayer(TILE_NUM_COL,TILE_NUM_ROW,tileImages,TILE_WIDTH,TILE_HEI
GHT);

    // Define Terrain Map
    int[][] map = {
      {0,0,0,0,0,0},
      {3,0,0,0,0,0},
      {6,0,0,0,0,0},
      {6,0,0,0,1,2},
      {6,0,0,0,4,5},
      {6,0,0,0,7,8},
      {6,0,0,0,0,0},
      {9,0,1,2,3,0},
      {0,0,4,5,6,0},
      {0,0,7,8,9,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {3,0,0,0,0,0},
      {6,0,0,0,0,0},
      {6,0,0,0,1,2},
      {6,0,0,0,4,5},
      {6,0,0,0,7,8},
      {6,0,0,0,0,0},
      {9,0,1,2,3,0},
      {0,0,4,5,6,0},
      {0,0,7,8,9,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {3,0,0,0,0,0},
      {6,0,0,0,0,0},
      {6,0,0,0,1,2},
```

```
    {6,0,0,0,4,5},
    {6,0,0,0,7,8},
    {6,0,0,0,0,0},
    {9,0,0,0,0,0},
    {0,0,0,0,0,0},
    {0,0,0,0,0,0},
    {0,0,0,0,0,0},
    {3,0,0,0,0,1}
  };

  // Map Terrain Map with actual graphic from terrain.png
  for (int row=0; row<TILE_NUM_ROW; row++) {
    for (int col=0; col<TILE_NUM_COL; col++) {
        tiledLayer.setCell(col,row,map[row][col]);
    }
  }
  return tiledLayer;
}
```

Scroll Terrain

As mentioned earlier there really isn't anything to it when scrolling a TiledLayer image it is just a matter of showing the image at different points based on the screen view port. We simply increment the terrain y position by 2 for each game loop; we did not set it to one 1 pixel increments because it would scroll too slowly. Because the scrollTerrain is an update to the graphics it is called by the tick() method. This technique differs from the way scrolled the background in Chapter 4. Remember in chapter 4 we scrolled through the LayerManager. The problem with scrolling the LayerManager itself is everything else that is appended to the LayerManager is affected as well. To keep other graphics such as Sprites unaffected by the scrolling we simply only adjust the position of the TiledLayer itself.

```
// Scroll Terrain
private void scrollTerrain() {
  if (terrainScroll < 0) {
    terrainScroll += 2;
    terrain.setPosition(0,terrainScroll);
  }
}
```

Terrain Graphic

The terrain graphic is made up 32 x 32 tiles with total of 9 tiles.

Figure 33 – Terrain in defined pieces

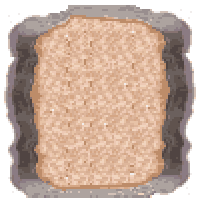


Figure 34 – Terrain

Source Code

Please see Chapter 11 source code for a full listing.

GameScreen.java:

```
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;

public class GameScreen extends GameCanvas implements Runnable, CommandListener
{
  private static final int MILLIS_PER_TICK = 50;
  private static final int TILE_WIDTH = 32;
  private static final int TILE_HEIGHT = 32;
  private static final int TILE_NUM_COL = 6;
  private static final int TILE_NUM_ROW = 36;


  private Eliminator midlet;  // Hold the Main Midlet
  private Settings settings;  // Hold Game Settings
  private Score score;        // Hold Game Score
  private Command backCommand = new Command("Back", Command.BACK,1);

  private boolean isPlay;   // Game Loop runs when isPlay is true
  private int width;        // To hold screen width
  private int height;       // To hold screen height

  private int scnViewWidth; // Hold Width Screen View Port
  private int scnViewHeight; // Hold Height Screen View Port

  private Thread gameThread = null;

  // Layer Manager to manager background (terrain)
  private LayerManager layerManager;

  // TiledLayer - Terrain
  private TiledLayer terrain;
  private int terrainScroll;   // Hold Y position for scrolling

  // Constructor and initialization
  public GameScreen(Eliminator midlet,Settings settings,Score score) throws Exception {
    super(true);
    this.midlet = midlet;
    addCommand(backCommand);
    setCommandListener(this);
```

```
  width = getWidth();  // get screen width
  height = getHeight();  // get screen height
  scnViewWidth = width; // Set View Port width to screen width
  scnViewHeight = height; // Set View Port height to screen height

  isPlay = true;

  initTerrain();

  layerManager = new LayerManager();
  layerManager.append(terrain);

}

// Start thread for game loop
public void start() {
  gameThread = new Thread(this);
  gameThread.start();
}

// Stop thread for game loop
public void stop() {
  gameThread = null;
}

// Main Game Loop
public void run() {
  Graphics g = getGraphics();

  Thread currentThread = Thread.currentThread();

  try {
    while (currentThread == gameThread) {
        long startTime = System.currentTimeMillis();
        if (isShown()) {
          if (isPlay) {
            tick();
          }
       render(g);
        }
        long timeTake = System.currentTimeMillis() - startTime;
        if (timeTake < MILLIS_PER_TICK) {
          synchronized (this) {
            wait(MILLIS_PER_TICK - timeTake);
          }
```

```
          } else {
            currentThread.yield();
          }
      }
    } catch (InterruptedException ex) {
      // won't be thrown
    }

  }

  // Handle dynamic changes to game including user input
  public void tick() {
    scrollTerrain();
  }

  public void commandAction(Command c, Displayable d) {
    if (c == backCommand) {
      midlet.mainMenuScreenShow(null);
    }
  }

  // Method to Display Graphics
  private void render(Graphics g) {

    // Set Background color to beige
    g.setColor(0xF8DDBE);
    g.fillRect(0,0,width,height);
    g.setColor(0x0000ff);

    // LayerManager Paint Graphics
    layerManager.paint(g,0,0);

    flushGraphics();
  }

  private TiledLayer loadTerrain() throws Exception {
    Image tileImages = Image.createImage("/terrain.png");
    TiledLayer tiledLayer = new
TiledLayer(TILE_NUM_COL,TILE_NUM_ROW,tileImages,TILE_WIDTH,TILE_HEI
GHT);

    // Define Terrain Map
    int[][] map = {
       {0,0,0,0,0,0},
       {3,0,0,0,0,0},
       {6,0,0,0,0,0},
```

```
    {6,0,0,0,1,2},
    {6,0,0,0,4,5},
    {6,0,0,0,7,8},
    {6,0,0,0,0,0},
    {9,0,1,2,3,0},
    {0,0,4,5,6,0},
    {0,0,7,8,9,0},
    {0,0,0,0,0,0},
    {0,0,0,0,0,0},
    {0,0,0,0,0,0},
    {3,0,0,0,0,0},
    {6,0,0,0,0,0},
    {6,0,0,0,1,2},
    {6,0,0,0,4,5},
    {6,0,0,0,7,8},
    {6,0,0,0,0,0},
    {9,0,1,2,3,0},
    {0,0,4,5,6,0},
    {0,0,7,8,9,0},
    {0,0,0,0,0,0},
    {0,0,0,0,0,0},
    {0,0,0,0,0,0},
    {3,0,0,0,0,0},
    {6,0,0,0,0,0},
    {6,0,0,0,1,2},
    {6,0,0,0,4,5},
    {6,0,0,0,7,8},
    {6,0,0,0,0,0},
    {9,0,0,0,0,0},
    {0,0,0,0,0,0},
    {0,0,0,0,0,0},
    {0,0,0,0,0,0},
    {3,0,0,0,0,1}
  };

  // Map Terrain Map with actual graphic from terrain.png
  for (int row=0; row<TILE_NUM_ROW; row++) {
    for (int col=0; col<TILE_NUM_COL; col++) {
        tiledLayer.setCell(col,row,map[row][col]);
    }
  }
  return tiledLayer;
}

// Scroll Terrain
private void scrollTerrain() {
```

```
    if (terrainScroll < 0) {
      terrainScroll += 2;
      terrain.setPosition(0,terrainScroll);
    }
  }

  // Init Terrain and set Terrain at the bottom of the map
  private void initTerrain() throws Exception {
    try {
      terrain = loadTerrain();
      terrainScroll = 1 - (TILE_HEIGHT * terrain.getRows()) + scnViewHeight;
      terrain.setPosition(0,terrainScroll);
    } catch (Exception e) {
      throw new Exception("GameScreen::initTerrain::" + e);
    }
  }
}
```

Map Class for Multiple Terrains

Overview

It is most likely impractical to only have one level especially for a vertical shooter. So in this chapter with a few modifications we will now have the ability to easily modify the maps/levels in the game.

To keep things simple there will only be a total of 2 terrains, the first terrain is the same one we used in the "Simple One Terrain" section. The second terrain is a grassy hill with grass background.

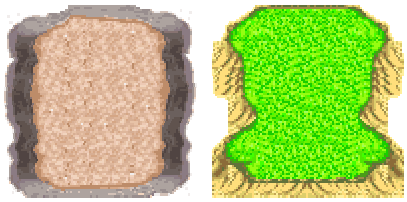


Figure 35 - Multiple Terrains

We now extract the terrain related settings to a separate class. The GameMap handles the following:

- Terrain selection
- Terrain Scrolling
- Terrain Initialization
- Terrain Background/Floor color

This class is now a reusuable component you can apply to other vertical scrollers and with a little adjustment you can reuse it for horizontal scrollers.

Constructor, Initialize and Load Terrain

By invoking the GameMap constructor the terrain is automatically loaded and initialized A simple call to the getter obtains the current terrain.

```
gameMap = new GameMap(scnViewHeight);
terrain = gameMap.getTerrain();
```

Scroll Terrain

To scroll the terrain the method scrollTerrain on the GameMap class is called.

gameMap.scrollTerrain();

As well to render terrain the GameScreen class will need to call the terrain getter to obtain the latest terrain after scrolling as been applied to it.

terrain = gameMap.getTerrain();

Source Code

GameMap.java:

```
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;

public class GameMap {

  // Terrain 1
  private static final int[][] map1 = {
      {0,0,0,0,0,0},
      {3,0,0,0,0,0},
      {6,0,0,0,0,0},
      {6,0,0,0,1,2},
      {6,0,0,0,4,5},
      {6,0,0,0,7,8},
      {6,0,0,0,0,0},
      {9,0,1,2,3,0},
      {0,0,4,5,6,0},
      {0,0,7,8,9,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {3,0,0,0,0,0},
      {6,0,0,0,0,0},
      {6,0,0,0,1,2},
      {6,0,0,0,4,5},
      {6,0,0,0,7,8},
      {6,0,0,0,0,0},
      {9,0,1,2,3,0},
      {0,0,4,5,6,0},
      {0,0,7,8,9,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {0,0,0,0,0,0},
      {3,0,0,0,0,0},
      {6,0,0,0,0,0},
      {6,0,0,0,1,2},
      {6,0,0,0,4,5},
      {6,0,0,0,7,8},
      {6,0,0,0,0,0},
      {9,0,0,0,0,0},
      {0,0,0,0,0,0},
```

```
    {0,0,0,0,0,0},
    {0,0,0,0,0,0},
    {3,0,0,0,0,1}
};

// Terrain 2
private static final int[][] map2 = {
    {0,0,0,0,0,0},
    {3,0,0,0,0,0},
    {6,0,0,0,0,0},
    {6,0,0,0,1,2},
    {6,0,0,0,4,5},
    {6,0,0,0,7,8},
    {6,0,0,0,0,0},
    {9,0,1,2,3,0},
    {0,0,4,5,6,0},
    {0,0,7,8,9,0},
    {0,0,0,0,0,0},
    {0,0,0,0,0,0},
    {0,0,0,0,0,0},
    {3,0,0,0,0,0},
    {6,0,0,0,0,0},
    {6,0,0,0,1,2},
    {6,0,0,0,4,5},
    {6,0,0,0,7,8},
    {6,0,0,0,0,0},
    {9,0,1,2,3,0},
    {0,0,4,5,6,0},
    {0,0,7,8,9,0},
    {0,0,0,0,0,0},
    {0,0,0,0,0,0},
    {0,0,0,0,0,0},
    {3,0,0,0,0,0},
    {6,0,0,0,0,0},
    {6,0,0,0,1,2},
    {6,0,0,0,4,5},
    {6,0,0,0,7,8},
    {6,0,0,0,0,0},
    {9,0,0,0,0,0},
    {0,0,0,0,0,0},
    {0,0,0,0,0,0},
    {0,0,0,0,0,0},
    {3,0,0,0,0,1}
};

// Set Constant values, the values are direct
```

```
// relation to the actually tile sizes
// defined for the terrain graphics
private static final int TILE_WIDTH = 32;
private static final int TILE_HEIGHT = 32;
private static final int TILE_NUM_COL = 6;
private static final int TILE_NUM_ROW = 36;

// To hold the current map
private int[][] currentMap;

// To hold the current terrain
private TiledLayer terrain;

// To hold the current background/floor color
private int groundColor;

// To hold the current screen, value neeeded for scrolling calculation
private int screenHeight;

// To hold Y position for scrolling
private int terrainScroll;

public GameMap(int screenHeight) throws Exception {
  this.screenHeight = screenHeight;
  setMap(1);  // default to set to terrain 1
}

// Set Appropriate Terrain and Map
public void setMap(int level) throws Exception {
  Image tileImages = null;

  switch (level) {
    case 1:  tileImages = Image.createImage("/terrain1.png");
             currentMap = map1;
             groundColor = 0xF8DDBE;
             break;
    case 2:  tileImages = Image.createImage("/terrain2.png");
             currentMap = map2;
             groundColor = 0xDECE6B;
             break;
    default: tileImages = Image.createImage("/terrain1.png");
             currentMap = map1;
             groundColor = 0xF8DDBE;
             break;
  }
```

```
    terrain = new
TiledLayer(TILE_NUM_COL,TILE_NUM_ROW,tileImages,TILE_WIDTH,TILE_HEI
GHT);

    // Map Terrain Map with actual graphic from terrain.png
    for (int row=0; row<TILE_NUM_ROW; row++) {
      for (int col=0; col<TILE_NUM_COL; col++) {
           terrain.setCell(col,row,currentMap[row][col]);
      }
    }

    terrainScroll = 1 - (terrain.getCellHeight() * terrain.getRows()) + screenHeight;
    terrain.setPosition(0,terrainScroll);
  }

  public void scrollTerrain() {
    if (terrainScroll < 0) {
      terrainScroll += 2;
      terrain.setPosition(0,terrainScroll);
    }
  }

  // Terrain Getter
  public TiledLayer getTerrain() {
    return terrain;
  }

  // Ground/Floor color Getter
  public int getGroundColor() {
    return groundColor;
  }
}
```

GameScreen.java:

The GameScreen class is now simplified with the use of the GameMap class

```
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;

public class GameScreen extends GameCanvas implements Runnable, CommandListener
{
 private static final int MILLIS_PER_TICK = 50;

 private Eliminator midlet;  // Hold the Main Midlet
 private Settings settings;  // Hold Game Settings
 private Score score;        // Hold Game Score
 private Command backCommand = new Command("Back", Command.BACK,1);
 private GameMap gameMap;

 private boolean isPlay;   // Game Loop runs when isPlay is true
 private int width;        // To hold screen width
 private int height;       // To hold screen height

 private int scnViewWidth; // Hold Width Screen View Port
 private int scnViewHeight; // Hold Height Screen View Port

 private Thread gameThread = null;

 // Layer Manager to manager background (terrain)
 private LayerManager layerManager;

 // TiledLayer - Terrain
 private TiledLayer terrain;
 private int terrainScroll;   // Hold Y position for scrolling

 // Constructor and initialization
 public GameScreen(Eliminator midlet,Settings settings,Score score) throws Exception {
   super(true);
   this.midlet = midlet;
   addCommand(backCommand);
   setCommandListener(this);

   width = getWidth();  // get screen width
   height = getHeight();  // get screen height
   scnViewWidth = width; // Set View Port width to screen width
   scnViewHeight = height; // Set View Port height to screen height

   isPlay = true;
```

```
  gameMap = new GameMap(scnViewHeight);
  terrain = gameMap.getTerrain();

  layerManager = new LayerManager();
  layerManager.append(terrain);

}

// Start thread for game loop
public void start() {
  gameThread = new Thread(this);
  gameThread.start();
}

// Stop thread for game loop
public void stop() {
  gameThread = null;
}

// Main Game Loop
public void run() {
  Graphics g = getGraphics();

  Thread currentThread = Thread.currentThread();

  try {
    while (currentThread == gameThread) {
        long startTime = System.currentTimeMillis();
        if (isShown()) {
          if (isPlay) {
            tick();
          }
       render(g);
        }
        long timeTake = System.currentTimeMillis() - startTime;
        if (timeTake < MILLIS_PER_TICK) {
          synchronized (this) {
            wait(MILLIS_PER_TICK - timeTake);
          }
        } else {
          currentThread.yield();
        }
    }
  } catch (InterruptedException ex) {
```

```
      // won't be thrown
    }

  }

  // Handle dynamic changes to game including user input
  public void tick() {
    // Scroll Terrain
    gameMap.scrollTerrain();
  }

  public void commandAction(Command c, Displayable d) {
    if (c == backCommand) {
      midlet.mainMenuScreenShow(null);
    }
  }

  // Method to Display Graphics
  private void render(Graphics g) {

    // Set Background color to beige
    //g.setColor(0xF8DDBE);
    g.setColor(gameMap.getGroundColor());
    g.fillRect(0,0,width,height);
    g.setColor(0x0000ff);

    // Get Current Map
    terrain = gameMap.getTerrain();

    // LayerManager Paint Graphics
    layerManager.paint(g,0,0);

    flushGraphics();
  }
}
```

Chapter 12 – Eliminator: Player and Bullets

Overview

Player

Now lets add the main sprite the player that the user controls. As we develop the game further the player can have several attributes such as number of lives left, energy bar, number of bombs left and different types of weapons. Player attributes can get very complicated especially in role playing games where there can be literally dozens and dozens of attributes, ranging anywhere from intelligence, agility, strength, stamina to type of Armour. It may then make sense to create a custom sprite class that inherits the Sprite class from the MIDP 2 library.

The PlayerSprite.java is a fairly straightforward class wrapping most the available public interfaces from the Sprite class. The only method that may seem odd is the fire method which is used to fire bullets.

```
public Sprite fire(Image bullets) {
  Sprite bullet = new Sprite(bullets,2,2);
  bullet.setFrame(0);
  getXY();
  bullet.setPosition(x - bullet.getWidth() / 2 + this.getWidth() / 2, y);
  return bullet;
}
```

What this method does is obtain the current players position and uses those co-ordinates to create a new Sprite which actually is a bullet sprite in the game. The use of this will be more evident in the GameScreen.java file

Bullets

Because there really isn't much functionality needed for a bullet using the default Sprite class is good enough. But in most games there will always be more then one bullet on the screen, a vector is then used to hold multiple bullets.

To reduce the use of resources every Sprite that fires shares this same vector including the player class. The question is how do we easily determine who fires what? We simply use more then one frame; see the next graphic section, the first frame the bullet is the color green to represent a player bullet. The next frame is color red which represents an enemy bullet.

This is flexible enough to add additional bullet types and even dedicated certain frames to certain sets of bullets to create animation if necessary.

Graphics

Player

Here is the player graphic sheet from the file named player.png

Figure 36 - Player Sprite Graphic Sheet

There is only one frame to represent the player when the player is a live; the remaining three frames represent the player when being blown up. To improve the graphics slightly and to give the user better feedback two more frames should be added to the live state, one for turning left and one for turning right.

Bullet

Here is the bullet graphic sheet from the file named bullets.png, this diagram is enlarged the actual size for each frame is 2X2 pixels, this case where there are only two types of bullets the entire size is 4X2 pixels.

Figure 37 - Bullet Sprites (Enlarged)

Source

Player Sprite

```
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;

/**
 * Main Sprite / Player
 * Inherits the MIDP 2.0 Sprite class
 */
public class PlayerSprite extends Sprite {
  private static final int MOVE = 3;
  private int x,y;
  private int scnWidth,scnHeight;
  private int frameWidth, frameHeight;
  private int frame;
  private int lives;

  public PlayerSprite(Image image, int frameWidth, int frameHeight, int scnWidth, int
scnHeight) throws Exception {
    super(image, frameWidth, frameHeight);
    x = frameWidth/2;
    y = frameHeight/2;
    this.scnWidth = scnWidth;
    this.scnHeight = scnHeight;
    this.frameWidth = frameWidth;
    this.frameHeight = frameHeight;
    this.frame = 1;
    this.lives = 3;
  }

  public void startPosition() {
    setPosition(scnWidth/2,scnHeight/2);
  }

  public void moveLeft() {
    getXY();
    if (x - MOVE > 0)
      move(MOVE * -1,0);
  }

  public void moveRight() {
    getXY();
    if (x + MOVE + frameWidth < scnWidth)
```

```
    move(MOVE,0);
  }

  public void moveUp() {
    getXY();
    if (y - MOVE > 0)
      move(0,MOVE * -1);
  }

  public void moveDown() {
    getXY();
    if (y + MOVE + frameHeight < scnHeight)
      move(0,MOVE);
  }

  public Sprite fire(Image bullets) {
    Sprite bullet = new Sprite(bullets,2,2);
    bullet.setFrame(0);
    getXY();
    bullet.setPosition(x - bullet.getWidth() / 2 + this.getWidth() / 2, y);
    return bullet;
  }

  public void display(Graphics g) {
    this.setFrame(frame);
    this.paint(g);
  }

  public int getLives() {
    return lives;
  }

  public void setLives(int lives) {
    this.lives = lives;
  }

  private void getXY() {
    x = getX();
    y = getY();
  }
}
```

Bullets

The bullets are initialized like any other Sprite; however instead of being added to the LayerManager in the constructor it is added dynamically whenever the player fires and in later chapters when ever any Sprite fires.

```
// Player Fires
if ((keyStates & FIRE_PRESSED) != 0) {
  Sprite bullet = player.fire(bulletImages);
  if (bullet != null) {
    bullets.addElement(bullet);
    layerManager.insert(bullet,1);
  }
}
```

As during the tick() method all bullet movement is update, in this case we only have logic to remove bullets from the vector and LayerManager once it reaches the top of the screen. Additional logic will be needed to handle bullets moving of the display at any point and of course bullets need to be removed if they collide into a valid Sprite.

```
// Update Bullet(s) Movement
for (int i = 0; i < bullets.size(); ++i)  {
  for (int j = 0; j < 2; ++j) {
    Sprite bullet = (Sprite)(bullets.elementAt(i));
    bullet.move(0, -1);
    if (bullet.getY() < 0) {
      bullets.removeElementAt(i);
      layerManager.remove(bullet);
      i--;
      break;
     }
  }
}
```

Game Screen

Putting it altogether, the bullet logic was explained in the last section. As for the player sprite everything should be self-explanatory, the bulk of the player code in the tick() method.

```
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;
import java.util.*;

public class GameScreen extends GameCanvas implements Runnable, CommandListener
{
  private static final int MILLIS_PER_TICK = 50;

  private Eliminator midlet;  // Hold the Main Midlet
  private Settings settings;  // Hold Game Settings
  private Score score;        // Hold Game Score
  private Command backCommand = new Command("Back", Command.BACK,1);
  private GameMap gameMap;

  private boolean isPlay;   // Game Loop runs when isPlay is true
  private int width;        // To hold screen width
  private int height;       // To hold screen height

  private int scnViewWidth; // Hold Width Screen View Port
  private int scnViewHeight; // Hold Height Screen View Port

  private Thread gameThread = null;

  // Layer Manager to manager background (terrain)
  private LayerManager layerManager;

  // TiledLayer - Terrain
  private TiledLayer terrain;
  private int terrainScroll;   // Hold Y position for scrolling

  // Sprites
  private PlayerSprite player;

  // Variables to hold bullet info
  private Vector bullets;
  private Image bulletImages;


  // Constructor and initialization
```

```
  public GameScreen(Eliminator midlet,Settings settings,Score score) throws Exception {
    super(true);
    this.midlet = midlet;
    addCommand(backCommand);
    setCommandListener(this);

    width = getWidth();  // get screen width
    height = getHeight();  // get screen height
    scnViewWidth = width; // Set View Port width to screen width
    scnViewHeight = height; // Set View Port height to screen height

    isPlay = true;

    // setup map
    gameMap = new GameMap(scnViewHeight);
    terrain = gameMap.getTerrain();

    // setup player sprite
    Image image = Image.createImage("/player.png");
    player = new PlayerSprite (image,24,18,width,height);  // 24 = width of sprite in pixels,
18 is height of sprite in pixels
    player.startPosition();

    // init bullets
    bullets = new Vector();
    bulletImages = midlet.createImage("/bullets.png");

    layerManager = new LayerManager();
    layerManager.append(player);
    layerManager.append(terrain);

  }

  // Start thread for game loop
  public void start() {
    gameThread = new Thread(this);
    gameThread.start();
  }

  // Stop thread for game loop
  public void stop() {
    gameThread = null;
  }

  // Main Game Loop
  public void run() {
```

```
  Graphics g = getGraphics();

  Thread currentThread = Thread.currentThread();

  try {
    while (currentThread == gameThread) {
          long startTime = System.currentTimeMillis();
          if (isShown()) {
            if (isPlay) {
              tick();
            }
         render(g);
          }
          long timeTake = System.currentTimeMillis() - startTime;
          if (timeTake < MILLIS_PER_TICK) {
            synchronized (this) {
              wait(MILLIS_PER_TICK - timeTake);
            }
          } else {
            currentThread.yield();
          }
    }
  } catch (InterruptedException ex) {
    // won't be thrown
  }

}

// Handle dynamic changes to game including user input
public void tick() {
  // Scroll Terrain
  gameMap.scrollTerrain();

  // Player Actions
  int keyStates = getKeyStates();

  // Player Moves
  if ( (keyStates & LEFT_PRESSED) != 0)  {
    player.moveLeft();
  } else if ((keyStates & RIGHT_PRESSED) !=0 )  {
    player.moveRight();
  } else if ((keyStates & UP_PRESSED) != 0) {
    player.moveUp();
  } else if ((keyStates & DOWN_PRESSED) != 0) {
    player.moveDown();
  }
```

```
  // Player Fires
  if ((keyStates & FIRE_PRESSED) != 0) {
    Sprite bullet = player.fire(bulletImages);
    if (bullet != null) {
      bullets.addElement(bullet);
      layerManager.insert(bullet,1);
    }
  }

  // Update Bullet(s) Movement
  for (int i = 0; i < bullets.size(); ++i)  {
    for (int j = 0; j < 2; ++j) {
      Sprite bullet = (Sprite)(bullets.elementAt(i));
      bullet.move(0, -1);
      if (bullet.getY() < 0) {
        bullets.removeElementAt(i);
        layerManager.remove(bullet);
        i--;
        break;
       }
    }
  }

}

public void commandAction(Command c, Displayable d) {
  if (c == backCommand) {
    midlet.mainMenuScreenShow(null);
  }
}
```

```
  // Method to Display Graphics
  private void render(Graphics g) {

    // Set Background color to beige
    //g.setColor(0xF8DDBE);
    g.setColor(gameMap.getGroundColor());
    g.fillRect(0,0,width,height);
    g.setColor(0x0000ff);

    // Get Current Map
    terrain = gameMap.getTerrain();

    // LayerManager Paint Graphics
    layerManager.paint(g,0,0);

    flushGraphics();
  }
}
```

Chapter 13 – Eliminator – Change of Scenery

Code CleanUp and Slim Down

Well before continue with the development of the actual game it time to do some house cleaning. As mentioned earlier before good programming and design practices in mobile development is sometimes aretraded for more efficient or at least for code that takes up less space.

Our first slim down is to remove the BaseRMS class, albeit there maybe some duplicate code by having to separate class not inherting a base class the bytes gained is significant, to further improve this once could merge the 2 classes together. I'll leave this up to you, remember though this can become more difficult for code maintenance. My reason for keeping them separate is because some games may not have a need for high score or settings; or possibly highscore is processing is completely done on the server instead of the client. Which ever case the seperation of the two different tasks allows you to easily rip out either feature.

See source code files for the changes: *Settings.java* and *Score.java*. (refer to Chapter 14)

Our second slim down is the remove of the MainMenuScreen class and moved the menu processing to the main midlet, the trade off here is the increased difficulty of replacing the menu with say customized canvas menu. Refer to chapter 14 and examine the main midlet class, *Elminator.java*.

The third code slim down is the merge of all high level gui screens into one class each screen is invoked through the use of a *switch*, see *Gui.java*. This removes separate classes for screens like About, High Score, Settings and Help.

Reasons for doing the above merges is to free up some byte storage for additional classes need for the game itself.

In addition, you will notice all text outputs now use a constant, for example is is the snippet of the main menu

```
mainMenu.append(Utils.getStr(Utils.ID_NEW),null);
```

Instead of using the actual strings "New", Utils.ID_NEW is used, the actuall string is stored in the class Utils, this class handles locale support, allowing for easy implementation in other languages. See Chapter 17 section *Unicode and World-Wide Language Support* for more information.

Change of Scenery

To complete the game more 4 levels are now defined in the GameMap class and the graphics have been improved by Leeman Cheng and Carlo Casimiro.

Here are the new graphics that will be used for the game:

Figure 38 – In Game Graphics

Chapter 14 – Eliminator: Game Items

Enemies

Enemies unlike the player sprite can move in all directions and turn in all directions. Remember the player can only move side to side and in doing so always faces forward. Well no point in re-inventing the wheel, we will borrow the code from the article Creating 2D Action Game Actions with the Game API by Jonathan Knudsen hosted at Sun Microsystems, http://developers.sun.com/techtopics/mobility/midp/articles/game/. His code demonstrates the use of transformation and mirroring images, I highly suggest ou read this article before continuing on.

With a few motification we can now incorpate support fo several different types of NPCs

```
public static final int NPC_CANON1 = 1;
  public static final int NPC_CANON2 = 2;
  public static final int NPC_CANON3 = 3;
  public static final int NPC_FLYER1 = 4;
  public static final int NPC_FLYER2 = 5;
  public static final int NPC_FLYER3 = 6;
  public static final int NPC_FLYER4 = 7;
  public static final int NPC_TANK1 = 8;
  public static final int NPC_TANK2 = 9;
  int npcType;
```

Because all the NPCs are defined in one graphic sheet we need to make the appropriate adjustments, admittly the following is some what of a hack:

```
    // Hack 1:1 relation to the PNG file 3 frames of cannons first then a flyer... as we add
    //more NPCs this switch needs to be updated accordingly
    switch(npcType) {
      case NPC_CANON1: this.frameOffset = 0; this.setFrame(0); break;
      case NPC_CANON2: this.frameOffset = 3; this.setFrame(3); break;
      case NPC_CANON3: this.frameOffset = 6; this.setFrame(6); break;
      case NPC_FLYER1: this.frameOffset = 9; this.setFrame(9); break;
      case NPC_FLYER2: this.frameOffset = 12; this.setFrame(12); break;
      case NPC_FLYER3: this.frameOffset = 15; this.setFrame(15); break;
      case NPC_FLYER4: this.frameOffset = 18; this.setFrame(18); break;
      case NPC_TANK1: this.frameOffset = 21; this.setFrame(21); break;
      case NPC_TANK2: this.frameOffset = 24; this.setFrame(24); break;
    }
```

To add some life to the NPCs we add a firing method is some what similar to the firing method we produced for the main player but with 2 additional conditions. One we need to determine which direction the bullet is firing unlike the player bullet which always

moves from the bottom of the screen to the top. Secondly, we need to control the firing rate of the bullet this is simply done by adding a delay and calculating the time firing time between calls when the method is called. Of course you can easily add additional contidions to make the game more realistic. There is actually one more thing we need to determine if the enemy is actually facing the player before it fires, it the NPC is not facing the player then the NPC should turn in the direction of the player.

The following methods are used to determine what quadrant the NPC is in and determines what direction it should move/turn in.

```
// Major hack and magic numbers here, sorry I'll try and make this more readable later
  private int getPlayerDirection(PlayerSprite player) {

    // Beside the NPC
    if (player.getMidY() <= (getMidY() + SPC) && player.getMidY() >= (getMidY()-
SPC) ) {
      if (player.getX() < getX())
        return 12;
      else
        return 4;
    } else if (player.getMidX() >= getMidX() - SPC && player.getMidX() <=
getMidX()+SPC ) {
      if (player.getY() > getY())
        return 8;
      else
        return 0;

    // Quad 4
    } else if (player.getMidX() <= getMidX() && player.getMidY() <= getMidY()   ) {

      int qC = quadCalc(player);
      if (qC <= SPC)
        return 14;
      else if ( Math.abs(player.getMidX() - getMidX()) > Math.abs(player.getY() -
getMidY())   )
        return 13;
      else
        return 15;

    // Quad 3
    } else if (player.getMidX() <= getMidX() && player.getMidY() >= getMidY()   ) {
      int qC = quadCalc(player);
      if (qC <= SPC)
        return 10;
      else if ( Math.abs(player.getMidX() - getMidX()) > Math.abs(player.getY() -
getMidY())   )
```

```
        return 11;
      else
        return 9;

    // Quad 2
    } else if (player.getMidX() >= getMidX() && player.getMidY() >= getMidY()   ) {
      int qC = quadCalc(player);
      if (qC <= SPC)
        return 6;
      else if ( Math.abs(player.getMidX() - getMidX()) > Math.abs(player.getY() -
getMidY())   )
        return 5;
      else
        return 7;

    // Quad 1
    } else if (player.getMidX() >= getMidX() && player.getMidY() <= getMidY()   ) {
      int qC = quadCalc(player);
      if (qC <= 5)
        return 2;
      else if ( Math.abs(player.getMidX() - getMidX()) > Math.abs(player.getY() -
getMidY())   )
        return 3;
      else
        return 1;
    }

    // Should never reach here
    return -1;
  }

  private int quadCalc(PlayerSprite player) {
    return Math.abs((Math.abs(player.getMidX() - getMidX())) - (Math.abs(player.getY() -
getMidY())));
  }
```

Power Ups

Chapter 15 – Eliminator: Boss

Chapter 16 – Eliminator: Game Extras

Sound and Vibration

Though this Eliminator will not have any sound and vibration, it does not mean these features are not important. Any features that can enhance the user experience is definitely good. However, keep in mind the both sound and vibration will use up more resources such as memory and battery life. As well you need to consider also that fact that mobile games may be played when sound may actually be annoying. For example, waiting in the doctor's office for an appointment, playing in the library, killing time on the bus or slacking off during classroom lecture. These are all times when one would like an option to either have the sound on or off.

You will have to consult the manufacturer for specific mobile handsets you want to implement sound and vibration for. However, there is now sound support in MIDP 2.0 under the javax.microedition.media package.

Binary Data

storing data, maps, images as binary….. etc

Data Downloads / Updates

- levels
- graphics
- … etc

Chapter 17 – Improving Usability

Pause, Resume and Save

Adding the ability to pause the game is extremely important especially when the user can at any moment be interrupted by an incoming phone call, SMS message, MMS message or some other incoming alert. Sometimes the user may have to quit his/her game abruptly, for example the bus arrives at the bus stop and he/she needs to dig out some change. In either case there should be an option to pause and resume the game. If the user decides to quite the game, a nice feature would be an auto-save feature so he/she can continue the game where he/she last left off.

When the user is interrupted for example by an incoming call the hideNotify() method is called this where you would call the pause logic. Likewise showNotify() calls the appropriate logic to continue the game. Primarily the two things that will be affected in a game are the soft keys or custom keys that are labeled pause and resume. The other affected area is the main game loop/thread itself. Of course the user should be allowed to explicitly pause the resume the game by pressing the buttons assigned to pause and resume.

The following source code is a simple example of implementing pause and stub method for implementing an auto-save.

Example Pause and Resume Using MIDP 2.0

ExamplePauseResumeMIDP2.java (main midlet):

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class ExamplePauseResumeMIDP2 extends MIDlet {
 private Display display;

 private GameScreen gameScreen;


 public ExamplePauseResumeMIDP2() {
  display = Display.getDisplay(this);
 }

 public void startApp() {
   try {
     this.gameScreen = new GameScreen(this);
     this.gameScreen.start();
     display.setCurrent(this.gameScreen);
   } catch (Exception ex) {
     System.out.println("error: " + ex);
   }
 }

 public void pauseApp() {  }
 public void destroyApp(boolean unconditional) {
   gameScreen.autoSave();
 }

 public void exitGame() {
   destroyApp(false);
   notifyDestroyed();
 }
}
```

GameScreen.java:

```
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;
import java.util.*;

public class GameScreen extends GameCanvas implements Runnable, CommandListener
{
  private static final int MILLIS_PER_TICK = 50;

  private Command exitCommand = new Command("Exit", Command.BACK,1);
  private Command pauseCommand = new Command("Pause", Command.SCREEN,1);
  private Command resumeCommand = new Command("Resume",
Command.SCREEN,1);

  private ExamplePauseResumeMIDP2 midlet;

  private boolean isPlay;   // Game Loop runs when isPlay is true
  private int width;        // To hold screen width
  private int height;       // To hold screen height

  private Thread gameThread = null; // Main Game Thread/loop

  // Layer Manager
  private LayerManager layerManager;

  // Pause flag
  private boolean isPause;

  // Variables used for demo animation
  private int posX;
  private int posY;
  private boolean isUp;

  public GameScreen(ExamplePauseResumeMIDP2 midlet) throws Exception {
    super(true);
    this.midlet = midlet;
    addCommand(exitCommand);
    addCommand(pauseCommand);
    setCommandListener(this);

    width = getWidth();  // get screen width
    height = getHeight();  // get screen height

    isPlay = true;
```

```
  isPause = false;

  posX = width/2;
  posY = height/2;
  isUp = true;

  layerManager = new LayerManager();
}

// Start thread for game loop
public void start() {
  gameThread = new Thread(this);
  gameThread.start();
}

// Stop thread for game loop
public void stop() {
  gameThread = null;
}

// Main Game Loop
public void run() {
  Graphics g = getGraphics();

  Thread currentThread = Thread.currentThread();

  try {
    while (currentThread == gameThread) {
        long startTime = System.currentTimeMillis();
        if (isShown()) {
          if (isPlay) {
            tick();
          }
       render(g);
        }
        long timeTake = System.currentTimeMillis() - startTime;
        if (timeTake < MILLIS_PER_TICK) {
          synchronized (this) {
            wait(MILLIS_PER_TICK - timeTake);
          }
        } else {
          currentThread.yield();
        }
    }
  } catch (InterruptedException ex) {
    // won't be thrown
```

```
  }

}

// Handle dynamic changes to game including user input
public void tick() {
  try {
    if (isUp) {
         if (posY-3 > 0) {
           posY-=3;
         } else {
           isUp = false;
      }
    }

    if (!isUp) {
         if (posY+3 < height) {
           posY+=3;
         } else {
          isUp = true;
         }
    }

  } catch (Exception e) {
    stop();
  }
}

// Handle Soft-Key reponses
public void commandAction(Command c, Displayable d) {
  if (c == exitCommand) {
    midlet.exitGame();
  }

  if (c == pauseCommand) {
    pauseGame();
  }

  if (c == resumeCommand) {
    resumeGame();
  }
}

// Method to Display Graphics
private void render(Graphics g) {
  g.setColor(0x00FFFFFF);
```

```
    g.fillRect(0,0,width,height);
    g.setColor(0x000000FF);
    g.drawString("X",posX,posY,Graphics.TOP|Graphics.LEFT);
    layerManager.paint(g,0,0);
    flushGraphics();
  }

  // This where you should handle game auto-save, save to either network server
  // and/or local RMS
  public void autoSave() {
    // Auto Save Stuff should be here
    System.out.println("Auto Saving Logic Should Be Here");
  }

  public void pauseGame() {
    // You may or may not want to have auto-save done at Pause, that is why when you
exit
    // the system console outpus and extra 2 auto saving messages, total 3 including the
    // the call to autosave when exiting
    autoSave();
    removeCommand(pauseCommand);
    addCommand(resumeCommand);
    isPause=true;
    stop();
  }

  public void resumeGame() {
    removeCommand(resumeCommand);
    addCommand(pauseCommand);
    isPause=false;
    start();
  }

  public void hideNotify() {
    pauseGame();
  }

  public void showNotify() {
    resumeGame();
  }
}
```

Example Pause and Resume Using MIDP 1.0

ExamplePauseResumeMIDP2.java (main midlet):

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class ExamplePauseResumeMIDP1 extends MIDlet {
 private Display display;

 private GameCanvas gameCanvas;

 public ExamplePauseResumeMIDP1() {
  display = Display.getDisplay(this);
 }

 public void startApp() {
   try {
     this.gameCanvas = new GameCanvas(this);
     this.gameCanvas.start();
     display.setCurrent(this.gameCanvas);
   } catch (Exception ex) {
     System.out.println("error: " + ex);
   }
 }

 public void pauseApp() {  }
 public void destroyApp(boolean unconditional) {
   gameCanvas.autoSave();
 }

 public void exitGame() {
   destroyApp(false);
   notifyDestroyed();
 }
}
```

GameScreen.java:

```
import javax.microedition.lcdui.*;
import java.util.*;

public class GameCanvas extends Canvas implements Runnable, CommandListener {
  private static final int MILLIS_PER_TICK = 50;

  private Command backCommand = new Command("Exit", Command.BACK,1);
  private Command pauseCommand = new Command("Pause", Command.SCREEN,1);
  private Command resumeCommand = new Command("Resume",
Command.SCREEN,1);

  private ExamplePauseResumeMIDP1 midlet;

  private boolean isPlay;
  private int width;
  private int height;

  private Thread gameThread = null;
  private Image buffer;

  // Pause flag
  private boolean isPause;

  // Variables used for demo animation
  private int posX;
  private int posY;
  private boolean isUp;

  public GameCanvas(ExamplePauseResumeMIDP1 midlet) throws Exception {
    this.midlet = midlet;
    addCommand(backCommand);
    addCommand(pauseCommand);
    setCommandListener(this);

    width = getWidth();  // get screen width
    height = getHeight();  // get screen height

    isPlay = true;
    isPause = false;

    posX = width/2;
    posY = height/2;
    isUp = true;
```

```
  if (!isDoubleBuffered())
    buffer = Image.createImage(width,height);
}

// Start thread for game loop
public void start() {
  gameThread = new Thread(this);
  gameThread.start();
}

// Stop thread for game loop
public void stop() {
  gameThread = null;
}

// Main Game Loop
public void run() {

  Thread currentThread = Thread.currentThread();

  try {
    while (currentThread == gameThread) {
        long startTime = System.currentTimeMillis();
        if (isShown()) {
          if (isPlay) {
            tick();
          }
       repaint(0,0,width,height);
       serviceRepaints();
        }
        long timeTake = System.currentTimeMillis() - startTime;
        if (timeTake < MILLIS_PER_TICK) {
          synchronized (this) {
            wait(MILLIS_PER_TICK - timeTake);
          }
        } else {
          currentThread.yield();
        }
    }
    synchronized(this){
      if(gameThread == Thread.currentThread()) {
        gameThread = null;
      }
    }
  } catch (InterruptedException ex) {
    // won't be thrown
```

```
  }

}

// Handle dynamic changes to game including user input
public void tick() {
  try {
    if (isUp) {
         if (posY-3 > 0) {
           posY-=3;
         } else {
           isUp = false;
      }
    }

    if (!isUp) {
         if (posY+3 < height) {
           posY+=3;
         } else {
          isUp = true;
         }
    }

  } catch (Exception e) {
    stop();
  }
}

public void keyPressed(int keyCode)  {}

// Handle Soft-Key reponses
public void commandAction(Command c, Displayable d) {
  if (c == backCommand) {
    midlet.exitGame();
  }

  if (c == pauseCommand) {
    pauseGame();
  }

  if (c == resumeCommand) {
    resumeGame();
  }
}

// Method to Display Graphics
```

```
  public void paint(Graphics g) {
    Graphics gr = g;
    try {
      if (buffer != null)
        g = buffer.getGraphics();
        g.setColor(0x00FFFFFF);
        g.fillRect(0,0,width,height);
        g.setColor(0x000000FF);
        g.drawString("X",posX,posY,Graphics.TOP|Graphics.LEFT);
      if(g!= gr)
        gr.drawImage(buffer,0,0,20);
    } catch (Exception e) {
      System.out.println("Animation Error");
    }
  }

  // This where you should handle game auto-save, save to either network server
  // and/or local RMS
  public void autoSave() {
    // Auto Save Stuff should be here
    System.out.println("Exiting and Auto Saving");
  }

  public void pauseGame() {
    autoSave();
    removeCommand(pauseCommand);
    addCommand(resumeCommand);
    isPause=true;
    stop();
  }

  public void resumeGame() {
    removeCommand(resumeCommand);
    addCommand(pauseCommand);
    isPause=false;
    start();
  }

  public void hideNotify() {
    pauseGame();
  }

  public void showNotify() {
    resumeGame();
  }
}
```

Unicode and World-Wide Language Support

What and Why Unicode

Unicode is a unique representation of a character; these characters are anything from the Latin language to Hebrew, Japanese to unique symbols. This character set is a universal standard called the ISO/IEC 10646. You can compare it to using ASCII, like ASCII, a number represents each character; however, ASCII has only 7 bits or in other words 128 character limitations. This covered everything in English like languages and a few other symbols but what about other languages and other symbols. As computer industry grew especially in other countries where English like languages where not the dominant character set new encoding schemes where invented. This potentially causes problems when older systems on different encoding scheme interface with newer systems, or when systems from one country interfaces with another system from another country. The systems either become corrupt or simply crash due to the inability to communicate with each other. That is why Unicode was invented, the following statement is taken from http://www.unicode.org it summarizes best what Unicode is and why we can benefit from it:

> *Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language. The Unicode Standard has been adopted by such industry leaders as Apple, HP, IBM, JustSystem, Microsoft, Oracle, SAP, Sun, Sybase, Unisys and many others. Unicode is required by modern standards such as XML, Java, ECMAScript (JavaScript), LDAP, CORBA 3.0, WML, etc., and is the official way to implement ISO/IEC 10646. It is supported in many operating systems, all modern browsers, and many other products. The emergence of the Unicode Standard, and the availability of tools supporting it, are among the most significant recent global software technology trends.*

As stated in the Unicode summary statement is supported on a wide variety of languages and platforms this includes Java and more specifically J2ME. Fortunately when Java was made/invented they include Unicode support, to use this handy feature simply use \u escape sequences.

Enabling J2ME Emulator with Unicode

There is one more step to properly display Unicode characters that is to ensure the display itself can support Unicode in this case the emulator screen. You will need to obtain the appropriate fonts, for example if you want to display Japanese characters you will need the MS Mincho font. After installing the font on your system you will now need to edit the properties file, we will go over that in next section. This can be very tedious if you wanted to support dozens and dozens of languages, you would have to install and edit dozens of property files. Of course, an easy solution to this is to obtain a font set that contains multiple language supports, one of which is the Arial Unicode MS

font. Unfortunately Microsoft no longer supplies this font on a free basis. You will either have to purchase it or find an alternative Unicode font set.

Now with the new fonts installed go to home directory of Sun Wireless ToolKit, go into the directory \wtk\devices. Now copy DefaultColorPhone and paste in the same directory and rename to UnicodePhone. As well rename the properties file under UnicodePhone directory to UnicodePhone.properties. Using any editor open the UnicdoePhone.properties file and find the font section it should look similar to the following

font.default=SansSerif-plain-10
font.softButton=SansSerif-plain-11

font.system.plain.small: SansSerif-plain-9
font.system.plain.medium: SansSerif-plain-11
font.system.plain.large: SansSerif-plain-14

font.system.bold.small: SansSerif-bold-9
font.system.bold.medium: SansSerif-bold-11
font.system.bold.large: SansSerif-bold-14

font.system.italic.small: SansSerif-italic-9
font.system.italic.medium: SansSerif-italic-11
font.system.italic.large: SansSerif-italic-14

font.system.bold.italic.small: SansSerif-bolditalic-9
font.system.bold.italic.medium: SansSerif-bolditalic-11
font.system.bold.italic.large: SansSerif-bolditalic-14

font.monospace.plain.small: Monospaced-plain-9
font.monospace.plain.medium: Monospaced-plain-11
font.monospace.plain.large: Monospaced-plain-14

font.monospace.bold.small: Monospaced-bold-9
font.monospace.bold.medium: Monospaced-bold-11
font.monospace.bold.large: Monospaced-bold-14

font.monospace.italic.small: Monospaced-italic-9
font.monospace.italic.medium: Monospaced-italic-11
font.monospace.italic.large: Monospaced-italic-14
.
.
Continued

Now replace SansSerif and Monospaced with the following Arial\ Unicode\ MS, assuming you are using the Arial Unicode MS font set. The file now should look similar to the following

font.default=Arial\ Unicode\ MS-plain-10
font.softButton=Arial\ Unicode\ MS-plain-11

font.system.plain.small: Arial\ Unicode\ MS-plain-9
font.system.plain.medium: Arial\ Unicode\ MS-plain-11
font.system.plain.large: Arial\ Unicode\ MS-plain-14

font.system.bold.small: Arial\ Unicode\ MS-bold-9
font.system.bold.medium: Arial\ Unicode\ MS-bold-11
font.system.bold.large: Arial\ Unicode\ MS-bold-14

font.system.italic.small: Arial\ Unicode\ MS-italic-9
font.system.italic.medium: Arial\ Unicode\ MS-italic-11
font.system.italic.large: Arial\ Unicode\ MS-italic-14

font.system.bold.italic.small: Arial\ Unicode\ MS-bolditalic-9
font.system.bold.italic.medium: Arial\ Unicode\ MS-bolditalic-11
font.system.bold.italic.large: Arial\ Unicode\ MS-bolditalic-14

font.monospace.plain.small: Arial\ Unicode\ MS-plain-9
font.monospace.plain.medium: Arial\ Unicode\ MS-plain-11
font.monospace.plain.large: Arial\ Unicode\ MS-plain-14

font.monospace.bold.small: Arial\ Unicode\ MS-bold-9
font.monospace.bold.medium: Arial\ Unicode\ MS-bold-11
font.monospace.bold.large: Arial\ Unicode\ MS-bold-14

font.monospace.italic.small: Arial\ Unicode\ MS-italic-9
font.monospace.italic.medium: Arial\ Unicode\ MS-italic-11
font.monospace.italic.large: Arial\ Unicode\ MS-italic-14

font.monospace.bold.italic.small: Arial\ Unicode\ MS-bolditalic-9
font.monospace.bold.italic.medium: Arial\ Unicode\ MS-bolditalic-11
font.monospace.bold.italic.large: Arial\ Unicode\ MS-bolditalic-14
.
.
Continued

For more detailed installation instructions refer to Qusay H. Mahmoud excellent tutorial indicated in the reference section.

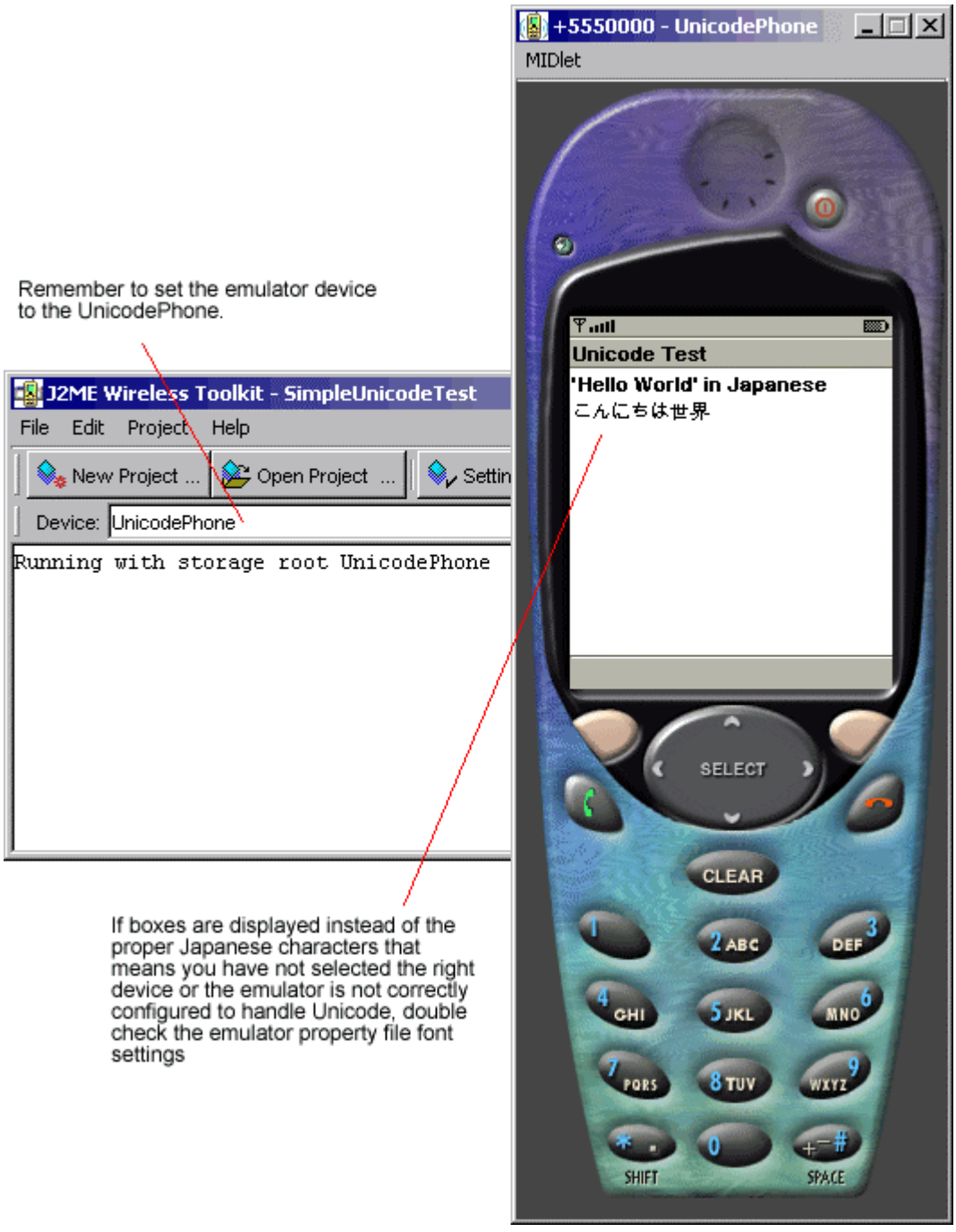
Simple Unicode Test

Now that everything is Unicode ready lets make the famous Hello World example:

Source Code:

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class SimpleUnicodeTest extends MIDlet {
  Display display;
  Form form = null;
  StringItem msg = null;

  public SimpleUnicodeTest() {
  }

  public void startApp() {
     display = Display.getDisplay(this);
     msg = new StringItem("'Hello World' in japanese",
                          "\u3053\u3093\u306B\u3061\u306F\u4E16\u754C");
     form = new Form("Unicode Test");
     form.append(msg);
     display.setCurrent(form);
  }

  public void pauseApp() {}
  public void destroyApp(boolean unconditional) {}
}
```

Output:



See Reference Section for the online web tools I used to convert Hello World to こんにちは世界 and from こんにちは世界 to \u3053\u3093\u306B\u3061\u306F\u4E16\u754C

Read Unicode File

The next step in module code separation would be load language definitions from external source. One option would be to read in language definitions from a text file and more appropriately a Unicode file. Here is a method you can use to read Unicode text file.

To create test file you can use the free application Simredo from http://www4.vc-net.ne.jp/~klivo/sim/simeng.htm

```
public String readUnicodeFile(String filename) {
  StringBuffer buffer = null;
  InputStream is = null;
  InputStreamReader isr = null;
  try {
    Class c = this.getClass();
    is = c.getResourceAsStream(filename);
    if (is == null)
      throw new Exception("File Does Not Exist");

    isr = new InputStreamReader(is,"UTF8");

    buffer = new StringBuffer();
    int ch;
    while ((ch = isr.read()) > -1) {
      buffer.append((char)ch);
    }
    if (isr != null)
      isr.close();
  } catch (Exception ex) {
    System.out.println(ex);
  }
  return buffer.toString();
}
```

Read Text file with Unicode codes

In the last section we went over reading a Unicode file, an alternative approach is to read text file with Unicode codes in them. Referring back to the Hello World example the following would be inside the text file \u3053\u3093\u306B\u3061\u306F\u4E16\u754C. One would think this is very straight forward and a simple load/read of text would do. However, when reading the file each character is treated as string. So in other words \u3053 reads in as '\', 'u', '3','0','5','3'. You will now have to detect and parse out the \u which indicates to you the next 4 characters is a Unicode character. The following method will help convert the string to the appropriate Unicode character, it assumes your are only passing in a valid 4 character Unicode, ie: 3053.

```
private String convertStrToUnicode(String value) {
  short valueAsShort = Short.parseShort(value.trim(),16);
  return String.valueOf((char)valueAsShort);
}
```

Locale Example Part 1

Now that you have understanding of Unicode we can move onto a full example of how one might implement Locale / Resource in an application or game.

In this example I've attempted to abstract / decouple the functionality and the locale data itself. Though this may not be the most efficient way with respect to mobile development, at the end of this section there will be suggestions on how to improve and/or enhance this example. However, the trade off here is that one can easily add and remove language support without recompiling the program. This may not be viewed as all the important because I've heard this many times, "oh easy just add it and recompile no big deal", but yes it is a big deal every time you change the code there is high possibility of introducing new bugs. At least if you are only editing the text files and something goes wrong, then you know it's the text file. To further improve date human entry errors you may want to consider developing an in house tool/application that writes out the locale files so verifies the file indeed is in the correct format. An example of this commonly occurring is say you have a game that has support for French and English in the UK and now the carrier requests that they want support for German and Spanish. All you have to do now is send them the appropriate new files or repackage the application with the supported locales reduces service level times.

The Locale class will read in 3 files based on the locale of the phone or if the code explicitly indicates the locale it wants to set. In the test it reads all the locales support from locale.support and echos out to the console all the keys defined for each locale based on the 1 to 1 mapping in the locale.keys file.

Test Driver Midlet:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class LocaleTest extends MIDlet {
   private LocaleLib localeLib;

   private Display display;
   TextBox box = null;

   public void startApp() {
     try {
          localeLib = LocaleLib.getInstance();
       //localeLib = LocaleLib.getInstance("en-us");
       //localeLib = LocaleLib.getInstance("de");

       String[] localeList = localeLib.getSupportedLocales();
       for (int i=0; i<localeList.length; i++) {
         localeLib.setLocaleKey(localeList[i]);
```

```
        localeLib.loadLocale();

        System.out.println("\n\n==== Locale " + localeList[i] + " ================");
        System.out.println(localeLib.getLocaleString("ID_HELLOWORLD"));
        System.out.println(localeLib.getLocaleString("ID_NEW"));
        System.out.println(localeLib.getLocaleString("ID_SETTINGS"));
        System.out.println(localeLib.getLocaleString("ID_HISCORE"));
        System.out.println(localeLib.getLocaleString("ID_HELP"));
        System.out.println(localeLib.getLocaleString("ID_ABOUT"));
        System.out.println(localeLib.getLocaleString("ID_EXIT"));
      }

      String temp2 = localeLib.getLocaleString("ID_HELLOWORLD");
      box = new TextBox("",temp2, 500, 0);
      display = Display.getDisplay(this);
      display.setCurrent(box);
    } catch (Exception ex) {
        System.out.println(ex);
    }
  }

  public void destroyApp(boolean b) {}
  public void pauseApp() {}
}
```

LocaleTest.java:

```
import java.io.*;
import java.util.*;

public class LocaleLib {
  private static LocaleLib instance = null;
  private static String localeKey = null;
  private static String[] idKeys = null;
  private static String[] supportedLocales = null;
  private static String[] translationText = null;
  private static Hashtable translation = new Hashtable();

  private LocaleLib() {
  }

  private LocaleLib(String localeKey) throws Exception {
    try {
      supportedLocales = loadTextFile("/locale.support");
      idKeys = loadTextFile("/locale.keys");
      setLocaleKey(localeKey);
```

```
    loadLocale();
  } catch (Exception e) {
    throw new Exception ("Error Loading Locale Resources::" + e);
  }
}

public static LocaleLib getInstance() throws Exception {
  return getInstance(System.getProperty("microedition.locale"));
}

public static LocaleLib getInstance(String localKey) throws Exception {
  if (instance == null)
    instance = new LocaleLib(localKey);
  return instance;
}

public String getLocaleKey() {
  return this.localeKey;
}

public String getLocaleString(String stringKey) {
  return this.parseUnicode((String)translation.get(stringKey));
}

public void setLocaleKey(String localeKey) throws Exception {
  this.localeKey = localeKey.toLowerCase();
  if (!validateLocaleKey())
    throw new Exception("Invalid Locale Key");
}

public String[] getSupportedLocales() {
  return this.supportedLocales;
}

public void loadLocale() throws Exception {
  translationText = loadTextFile("/locale." + this.localeKey.toLowerCase());
  if (translationText.length == idKeys.length) {
    for (int i=0; i<idKeys.length; i++) {
      translation.put(idKeys[i],translationText[i]);
    }
  } else {
    throw new Exception("Invalid Locale Files, data not matching keys");
  }
}

/**
```

```
 * Ensure the current Locale key exists in the supported local list
 * as listed in locale.support
 *
 * @param none
 * @return false is invalid and true is valid
 */
private boolean validateLocaleKey() {
  boolean result = false;
  for (int i=0; i<supportedLocales.length; i++) {
    if (localeKey.equals(supportedLocales[i].toLowerCase())) {
      result = true;
      break;
    }
  }
  return result;
}

/**
 * Read Text file deliminted by comma
 *
 * @param filename
 * @return Array with delimimted values
 */
private String[] loadTextFile(String filename) throws Exception {
  String [] result = null;
  int resultIdx = 0;
  StringBuffer sb = null;
  Vector strings = null;
  Enumeration enum = null;

  try {
    Class c = this.getClass();
    InputStream is = c.getResourceAsStream(filename);
    if (is == null)
      throw new Exception("File Does Not Exist");

    sb = new StringBuffer();
    strings = new Vector();
    byte b[] = new byte[1];

    while ( is.read(b) != -1 ) {
      if (b[0] != 44 && b[0] > 31 && b[0] < 255) {
        sb.append(new String(b));
      } else {
        if (sb.toString().trim().length() > 0) {
          strings.addElement(sb.toString().trim());
```

```
        }
        sb.delete(0,sb.length());
      }
    }
    is.close();
    if (sb.toString().trim().length() > 0) {
      strings.addElement(sb.toString().trim());
    } else {
        sb = null;
    }

    result = new String [strings.size()];
    enum = strings.elements();
    while (enum.hasMoreElements()) {
     result[resultIdx] = (String)enum.nextElement();
     resultIdx++;
    }
   } catch (Exception e) {
    throw new Exception ("ERROR: " + e);
   } finally {
    sb = null;
    strings = null;
    enum = null;
   }
   return result;
  }

  /**
   * Parse and translate unicode codes to the appropriate unicode symbol
   *
   * @param String value contain possibly both ASCII characters and unicode codes
   *        Unicode codes are signified by back slash then the character u
   * @return String value with unicode codes translated to thier appropriate unicode
symbols
   */
  private String parseUnicode(String value) {
   String result = "";

   // Ensures value has at least one character
   if (value == null || value.length() <= 0)
    return result;

   int idx = 0;
   int idxFound = -1;
   int findLength = 2;
   StringBuffer sb = new StringBuffer();
```

```
    while ((idxFound = value.indexOf("\\u",idx)) != -1) {
      sb.append(value.substring(idx,idxFound));
      sb.append(convertStrToUnicode(value.substring(idxFound+2,idxFound+6)));
      idx = idxFound + 6;
    }
    if(idx < value.length()) {
      sb.append(value.substring(idx));
    }

    return sb.toString();
  }

  /**
   * Converts a unicode string code to the unicode symbol
   *
   * @param String value representating a unicode code ie: 0669
   * @return unicode symbol as a String
   */
  private String convertStrToUnicode(String value) {
    short valueAsShort = Short.parseShort(value.trim(),16);
    return String.valueOf((char)valueAsShort);
  }
}
```

Local Example Part 2 - Suggestions

Now that we understand how the LocaleTest works, here are some suggestions on how to improve granted everyone will have different objectives so each suggestion may or may not suit your situation.

1. Add another layer of abstraction by adding in XML parser and having all the data files formatted as XML. This may be ideal for situations where standarization is more important such a case would be building a generic game server with the ability to plug multiple games from different vendors, Xadra at www.xadra.com does exactly this with their game server API.
2. You may not want to read the files locally but instead retrieve locale data from a server. This itself gives you a lot choices with respect to the commuication layer itself such as SOAP, Binary defined data…. Etc this will be further explained in another chapter.
3. Assuming you want non-network aware game and want to improve the efficieny of this code. You may want to consider removing all the data files and store all Locale data locally in the LocaleLib.java file as static strings. But if you are going to do this you may as well redefined the keys as integers rather then strings this will help out a bit too. The over all size of the program may be reduced as well. You wil have to do some experiementing on your own.

Chapter 18 – Adding Time Trial to Your Game

Overview

Time Trial is a method of allowing your potential customers to use your product, in this case a J2ME game or any other application, on limited basis. There are number of time trials commonly used in software, here are three common examples:

- **Date:** A specific date is used; when this date is met the game is expired. The assumption here is the correct time is always set on the users phone, otherwise the simple trick is to reset the time on the phone to an earlier date and the game will continue to work. In most cases users will find this annoying enough and will considering buying your game, especially when some mobile handsets with on all the time internet access will be automatically updated with the current date and time.

- **Number of Days**: When the user first uses the game the current date is recorded. From that date the number of days is recorded when the maximum specified days is reached the game is expired.

- **Number of Plays**: Every time the user plays the game the number of plays is recorded. Once the number of plays matches the maximum specified number of plays the game is expired.

The last two methods require the ability to persist either the number of days or number plays. One method is of course to use the local RMS system; however, some mobile handset are enabled with a feature to remove/clear all RMS records. Even if this feature isn't available one can easily obtain or write his/her own RMS utility to remove the RMS records or even modify the RMS records. An alternate solution would be to use the network and read the information from a server. To keep things simple we will be using the local RMS, which in most case is sufficient enough because in most cases casual users are not tech savvy enough to alter or remove the RMS entries themselves.

How

In each of the 3 methods listed in the last section a reference data is needed an expiry date, maximum number of days the game is valid or maximum number of plays. You have a few options where to store these initial values, one is to hard code the value in the code itself or in text file your can read during run time. The first case is not really all that feasible considering a recompile and repackaging is needed. Even the second method requires editing of the text file and repackaging of the JAR file. This maybe acceptable if you have total control of distribution but like many developers their games and applications are deployed by third parties, such as carriers, aggregators and other distribution channels. Asking them to recompile and repackage is more then likely not a possible option. However, editing the JAD file may be an option considering this JAD file is usually updated to the distributor's specific download URL. Of course you can have default time-trial values preset so the distrubutor does not need to be aware of what needs to be changed.

The three possible new parameters to be made to the JAD file are:

- Expire-Date: 20031201
- Expire-Days: 30
- Expire-Plays: 10

An example of what an actual JAD file might look like using Expiry-Date would be:

MIDlet-1: TimeTrialDemo, , TimeTrialDemo
MIDlet-Jar-Size: 49006
MIDlet-Jar-URL: TimeTrialDemo.jar
MIDlet-Name: TimeTrialDemo
MIDlet-Vendor: Sun Microsystems
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
Expire-Date: MjAwMzEwMTU=

You will notice the value for Expire-Date is not readable; it is encrypted using Base64 for increased security, albeit Base64 isn't a very strong encryption but serves a good enough purpose to deter someone simply changing the value assuming they can even get access to the JAD file. Though this may become a problem for the distributor but you can always resend an update JAD file that is still better then recompiling and repackaging.

To obtain a parameter value from a JAD file use the method getAppProperty(String key) from the MIDlet class, not the static method getProperty(String key) from the System class. An example

```
String expirePlays = getAppProperty("Expire-Date");
```

Once you have this value you can now check if the game time trial is expired, do not forget to validate the value. You may want to check if the value is given is indeed numeric, validate date, with the acceptable range. If any of these are broken the default result is possibly set game to expired mode.

Increase Protection and Improvements

Other approaches you may want to consider when protecting your software:

- Implement the time-trial using the server side component
- This may seem obvious but always obfuscate your code
- Add in DRM (Digital Rights Management). In the above example it doesn't stop anyone from transferring the current game to another phone. One prevention DRM offers is forward-lock which prevents unauthorized game distribution. Once the original game is encrypted with a key uniqued to the mobile handset, only the mobile handset is able to execute the game.
- Consider removing features you don't want the user to have available in a demo/time-trial game. For example, removal of levels or special power-ups; however, this may degrade the over experience of the game and discourages the customer to buy the full version.
- Use stronger encryption such as Bouncy Castle or Kerberos
- For your OTA download links make the temporary, a user can only download it once and/or expires after certain date
- OTA downloads should be available for members only

Is It Worth it?

Okay now that we've gone through the basics of how to implement time-trial to your game is it really worth the effort considering most games only sell on the average at $3.00 to $9.00 US per unit. This is something the business has to decide for itself. However, the carriers themselves may make the decisions for you. Some carriers require time-trial/demo versions.

Example

Main Midlet

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class TimeTrialDemo extends MIDlet implements CommandListener {
 private Display display;
 private Command cmdExit = new Command("Exit", Command.EXIT,1);;
 private Form formMain;
 private StringItem stringItem;

 private TimeTrial timeTrial;
 private static final String rmsName = "TimeTrialDemoRMS";
 private boolean result;

 public TimeTrialDemo() {
  try {

    // Manual Encrypt - You may need this to change the values in the JAD file
    //System.out.println("Manual Encryption: " + Base64.encodeToString("20031015"));


    // Time Trial - By Date
    //timeTrial = new TimeTrial(rmsName,TimeTrial.TT_DATE);
    //result = TimeTrial.isValid(getAppProperty("Expire-Date"));

    // Time Trial - Number of Days
    //timeTrial = new TimeTrial(rmsName,TimeTrial.TT_DAYS);
    //result = TimeTrial.isValid(getAppProperty("Expire-Days"));

    // Time Trial - Number of Plays
    timeTrial = new TimeTrial(rmsName,TimeTrial.TT_PLAYS);
    result = timeTrial.isValid(getAppProperty("Expire-Plays"));

    if (result)
      System.out.println("Game NOT Expired");
    else
      System.out.println("Game Expired");
  } catch (Exception ex) {
    System.out.println(ex);
  }
  display = Display.getDisplay(this);
  stringItem = new StringItem("","See Console");
```

```
  formMain = new Form("Time Trial Demo");
  formMain.addCommand(cmdExit);
  formMain.append(stringItem);
  formMain.setCommandListener(this);
 }

 public void startApp() {
  display.setCurrent(formMain);
 }

 public void pauseApp() {  }
 public void destroyApp(boolean unconditional) {}

 public void commandAction(Command c, Displayable s) {
  if (c == cmdExit) {
    destroyApp(false);
    notifyDestroyed();
  }
 }
}
```

Time Trial Class

```
import javax.microedition.rms.*;
import java.util.*;
import java.io.*;

public class TimeTrial extends BaseRMS {
  public static final int TT_DATE = 0;
  public static final int TT_DAYS = 1;
  public static final int TT_PLAYS = 2;

  public static final int MAX_NUM_DAYS = 30;
  public static final int MAX_NUM_PLAYS = 10;

  Calendar currentCalendar;
  Date currentDate;
  int  currentDateAsInt; // YYYYMMDD

  private int timeTrialType;
  private String value;
  private String decryptedValue;

  private int numDays;
  private int numPlays;

  /**
   * Constructor
   */
  public TimeTrial(String rmsName,int timeTrialType) throws Exception {
    super(rmsName);
    setTimeTrialType(timeTrialType);

    currentDate = new Date();
    currentCalendar = Calendar.getInstance();
    currentCalendar.setTime(currentDate);
    currentDateAsInt = getCurrentDateAsInt();

  }

  /**
   * Change TimeTrial Method
   */
  public void setTimeTrialType(int timeTrialType) throws Exception {
    if(timeTrialType == TT_DATE || timeTrialType == TT_DAYS || timeTrialType ==
TT_PLAYS) {
```

```
    this.timeTrialType = timeTrialType;
   } else {
    throw new Exception("Invalid Time Trial Option");
   }
  }

  /**
   * Check if App is Exired
   */
  public boolean isValid(String value) throws Exception {
   boolean result = false;
   this.value = value;
   decryptedValue = Base64.decodeToString(value);
   switch (this.timeTrialType) {
     case TT_DATE: result = isValidDate(); break;
     case TT_DAYS: result = isValidDays(); break;
     case TT_PLAYS: result = isValidPlays(); break;
   };
   return result;
  }

  /**
   * Time Trial by Date
   */
  private boolean isValidDate() throws Exception {
   boolean result = false;

   Date expireDate = new Date();
   Calendar expireCalendar = Calendar.getInstance();

   expireCalendar.set(Calendar.YEAR,Integer.parseInt(decryptedValue.substring(0,4)));

expireCalendar.set(Calendar.MONTH,Integer.parseInt(decryptedValue.substring(4,6)));

expireCalendar.set(Calendar.DAY_OF_MONTH,Integer.parseInt(decryptedValue.substri
ng(6,8)));
   expireDate = expireCalendar.getTime();

   if (currentDate.getTime() < expireDate.getTime())
     result = true;

   return result;
  }

  /**
   * Time Trial By Number of Days
```

```
 */
private boolean isValidDays() throws Exception {
  boolean result = false;

  loadTimeTrialData();
  if (currentDateAsInt - numDays < Integer.parseInt(decryptedValue)) {
    result = true;
  }
  return result;
}

/**
 * Time Trial By Number of Plays
 */

private boolean isValidPlays() throws Exception {
  boolean result = false;
  loadTimeTrialData();
  System.out.println("Play: " + decryptedValue + " Plays RMS: " + numPlays);
  if (numPlays <= Integer.parseInt(decryptedValue)) {
    updatePlays(++numPlays);
    result = true;
  }

  return result;
}

public void loadTimeTrialData() throws Exception {
  try {
    // Will call either loadData() or createDefaultData()
    this.open();

    if (this.getRecordStore() != null)
      this.close();
  } catch (Exception e) {
    throw new Exception("Error loading Settings" + e);
  }
}

private void updatePlays(int numPlays) throws Exception {
  try {
    // load current scores
    this.open();

    // Update
    this.numPlays = numPlays;
```

```
      updateData();

      // close
      if (this.getRecordStore() != null)
        this.close();
    } catch (Exception e) {
      throw new Exception(this.getRMSName() + "::updateTimeTrial::" + e);
    }
  }

  private int getCurrentDateAsInt() {
    StringBuffer sb = new StringBuffer();
    sb.append(String.valueOf(currentCalendar.get(Calendar.YEAR)));
    if (currentCalendar.get(Calendar.MONTH) < 10)
      sb.append("0" + String.valueOf(currentCalendar.get(Calendar.MONTH)));
    else
      sb.append(String.valueOf(currentCalendar.get(Calendar.MONTH)));
    sb.append(String.valueOf(currentCalendar.get(Calendar.DAY_OF_MONTH)));

    return Integer.parseInt(sb.toString());
  }

  ////////////////////////////////////////////////////////
  // RMS Related methods (loadData,createDefaultData, updateData) inherited from
BaseRMS

  protected void loadData() throws Exception {
    try {
        byte[] record = this.getRecordStore().getRecord(1);
        DataInputStream istream = new DataInputStream(new
ByteArrayInputStream(record,0,record.length));
        numDays = istream.readInt();
        numPlays = istream.readInt();
    } catch (Exception e) {
      throw new Exception (this.getRMSName() + "::loadData::" + e);
    }
  }

  /**
   * Default values for numDays will be currentDay (first day
   * the user plays the game
   *
   * numPlays is defaulted to zero number times played
   */
  protected void createDefaultData() throws Exception {
    try {
```

```
      numDays = currentDateAsInt;
      numPlays = 0;

      ByteArrayOutputStream bstream = new ByteArrayOutputStream(12);
      DataOutputStream ostream = new DataOutputStream(bstream);
      ostream.writeInt(numDays);
      ostream.writeInt(numPlays);
      ostream.flush();
      ostream.close();
      byte[] record = bstream.toByteArray();
      this.getRecordStore().addRecord(record,0,record.length);
    } catch (Exception e) {
      throw new Exception(this.getRMSName() + "::createDefaultData::" + e);
    }
  }

  protected void updateData() throws Exception {
    try {
      ByteArrayOutputStream bstream = new ByteArrayOutputStream(12);
      DataOutputStream ostream = new DataOutputStream(bstream);
      ostream.writeInt(numDays);
      ostream.writeInt(numPlays);
      ostream.flush();
      ostream.close();
      byte[] record = bstream.toByteArray();
      this.getRecordStore().setRecord(1, record, 0, record.length);
    } catch(Exception e) {
      throw new Exception(this.getRMSName() + "::updateData::" + e);
    }
  }

}
```

BaseRMS

See Chapter 10, re-use of the BaseRMS.java file

Base64 Class

The following code is not mine, I originally obtain the code from public domain source on the Internet, unfortunately I can not remember where exactly.

```
import java.io.*;

public class Base64 {
   protected static final char[] alphabet = {
         'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', // 0 to 7
         'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', // 8 to 15
         'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', // 16 to 23
         'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f', // 24 to 31
         'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', // 32 to 39
         'o', 'p', 'q', 'r', 's', 't', 'u', 'v', // 40 to 47
         'w', 'x', 'y', 'z', '0', '1', '2', '3', // 48 to 55
         '4', '5', '6', '7', '8', '9', '+', '/'  // 56 to 63
   };

   protected static final int[] decodeTable = {
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, // 0 to 9
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, // 10 to 19
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, // 20 to 29
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, // 30 to 39
         -1, -1, -1, 62, -1, -1, -1, 63, 52, 53, // 40 to 49
         54, 55, 56, 57, 58, 59, 60, 61, -1, -1, // 50 to 59
         -1, -1, -1, -1, -1,  0,  1,  2,  3,  4, // 60 to 69
          5,  6,  7,  8,  9, 10, 11, 12, 13, 14, // 70 to 79
         15, 16, 17, 18, 19, 20, 21, 22, 23, 24, // 80 to 89
         25, -1, -1, -1, -1, -1, -1, 26, 27, 28, // 90 to 99
         29, 30, 31, 32, 33, 34, 35, 36, 37, 38, // 100 to 109
         39, 40, 41, 42, 43, 44, 45, 46, 47, 48, // 110 to 119
         49, 50, 51                              // 120 to 122
   };

   public static String encodeToString(String s) {
     char[] encoded = encode(s);
     StringBuffer encodedSB = new StringBuffer();
     for (int i=0; i<encoded.length; i++) {
          encodedSB.append(encoded[i]);
     }
     return encodedSB.toString();
   }
```

```
public static char[] encode(String s) {
        return encode(s.getBytes());
}

public static char[] encode(byte[] bytes) {
        int sixbit;

        char[] output = new char[((bytes.length - 1) / 3 + 1) * 4];

        int outIndex = 0;
        int i = 0;

        while ((i + 3) <= bytes.length) {
           sixbit = (bytes[i] & 0xFC) >> 2;
           output[outIndex++] = alphabet[sixbit];
           sixbit = ((bytes[i] & 0x3) << 4) + ((bytes[i + 1] & 0xF0) >> 4);
           output[outIndex++] = alphabet[sixbit];
           sixbit = ((bytes[i + 1] & 0xF) << 2) + ((bytes[i + 2] & 0xC0) >> 6);
           output[outIndex++] = alphabet[sixbit];
           sixbit = bytes[i + 2] & 0x3F;
           output[outIndex++] = alphabet[sixbit];
           i += 3;
        }

        if (bytes.length - i == 2) {
           sixbit = (bytes[i] & 0xFC) >> 2;
           output[outIndex++] = alphabet[sixbit];
           sixbit = ((bytes[i] & 0x3) << 4) + ((bytes[i + 1] & 0xF0) >> 4);
           output[outIndex++] = alphabet[sixbit];
           sixbit = (bytes[i + 1] & 0xF) << 2;
           output[outIndex++] = alphabet[sixbit];
           output[outIndex++] = '=';
        } else if (bytes.length - i == 1) {
           sixbit = (bytes[i] & 0xFC) >> 2;
           output[outIndex++] = alphabet[sixbit];
           sixbit = (bytes[i] & 0x3) << 4;
           output[outIndex++] = alphabet[sixbit];
           output[outIndex++] = '=';
           output[outIndex++] = '=';
        }
        return output;
}

public static String decodeToString(String encoded) {
  byte[] decoded = decode(encoded);
  StringBuffer decodedSB = new StringBuffer();
```

```
   for (int i=0; i< decoded.length; i++) {
    decodedSB.append( (char)decoded[i]);
   }
   return decodedSB.toString();
}

public static byte[] decode(String encoded) {
        byte[] decoded = null;
        int decodedLength = (encoded.length() / 4 * 3);
        int invalid = 0;

        if (encoded.length() % 4 != 0) {
           System.err.println("It's not BASE64 encoded string.");
           return null;
        }
        if (encoded.charAt(encoded.length() - 2) == '=') {
           invalid = 2;
        } else if (encoded.charAt(encoded.length() - 1) == '=') {
           invalid = 1;
        }
        decodedLength -= invalid;
        decoded = new byte[decodedLength];

        int i = 0, di = 0;
        int sixbit0, sixbit1, sixbit2, sixbit3;

        for (; i < encoded.length() - 4; i += 4) {
           sixbit0 = decodeTable[encoded.charAt(i)];
           sixbit1 = decodeTable[encoded.charAt(i + 1)];
           sixbit2 = decodeTable[encoded.charAt(i + 2)];
           sixbit3 = decodeTable[encoded.charAt(i + 3)];

           decoded[di++] = (byte) ((sixbit0 << 2) + ((sixbit1 & 0x30) >> 4));
           decoded[di++] = (byte) (((sixbit1 & 0xF) << 4) + ((sixbit2 & 0x3C) >> 2));
           decoded[di++] = (byte) (((sixbit2 & 0x3) << 6) + sixbit3);
        }


        switch (invalid) {
        case 0 :
           sixbit0 = decodeTable[encoded.charAt(i)];
           sixbit1 = decodeTable[encoded.charAt(i + 1)];
           sixbit2 = decodeTable[encoded.charAt(i + 2)];
           sixbit3 = decodeTable[encoded.charAt(i + 3)];

           decoded[di++] = (byte) ((sixbit0 << 2) + ((sixbit1 & 0x30) >> 4));
```

```
            decoded[di++] = (byte) (((sixbit1 & 0xF) << 4) + ((sixbit2 & 0x3C) >> 2));
            decoded[di++] = (byte) (((sixbit2 & 0x3) << 6) + sixbit3);
            break;

        case 1 :
            sixbit0 = decodeTable[encoded.charAt(i)];
            sixbit1 = decodeTable[encoded.charAt(i + 1)];
            sixbit2 = decodeTable[encoded.charAt(i + 2)];

            decoded[di++] = (byte) ((sixbit0 << 2) + ((sixbit1 & 0x30) >> 4));
            decoded[di++] = (byte) (((sixbit1 & 0xF) << 4) + ((sixbit2 & 0x3C) >> 2));
            break;

        case 2 :
            sixbit0 = decodeTable[encoded.charAt(i)];
            sixbit1 = decodeTable[encoded.charAt(i + 1)];

            decoded[di++] = (byte) ((sixbit0 << 2) + ((sixbit1 & 0x30) >> 4));
            break;
        }

        return decoded;
    }

    static boolean bytesEquals(byte[] b1, byte[] b2) {
        if (b1.length != b2.length) {
            return false;
        }

        for (int i = b1.length - 1; i >= 0; i--) {
            if (b1[i] != b2[i]) {
                return false;
            }
        }
        return true;
    }
}
```

Chapter 19 - Custom Interface

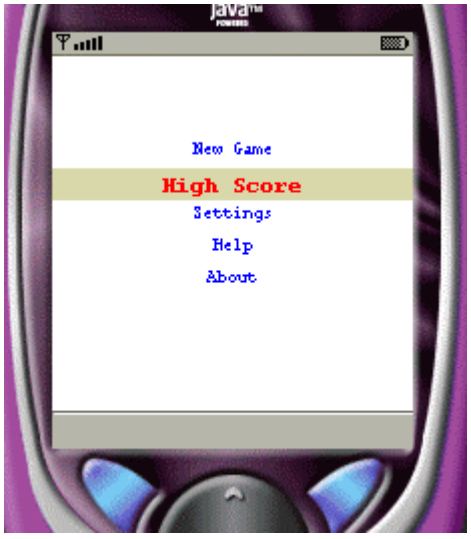
This chapter goes through the basics of how to create your own custom interface using the GameCanvas or Canvas, we won't go through the advantages and disadvantes of why one would create a custom interface, this is already covered in chapter 8.

Some the major considerations when creating a custom interface is the method of input. This is fairly obvisous for menu selection, but becomes a little more tricky for such things as settings. It may be difficult to start creating your own checkboxes, radio buttons and/or drop downs. What about character input for high score input, credential input for games that require login and/or secret codes to skp levels? We go through this in the coming section.

Main Menu

Simple Highlight Menu

The main menu now that we have total control lets us design something more creative and something that looks more like a game. Let us start with something easy, a centered menu with five menu items (New Game, High Score, Settings, Help, About). The current selected menu item will standout by increasing the font size, different font color and it will have a highlight bar behind it. The following is what it may possibly look like:



Now lets look at the source code

.

Main Midlet – SimleCustomMenu.java:

Fairly Straight-forward, like any other midlet driver.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.*;

public class SimpleCustomMenu extends MIDlet implements CommandListener {

  Display display;
  Display pauseDisplay;
  boolean isSplash = true;
  MenuScreen menuScreen;

  public SimpleCustomMenu() {
    MenuScreen menuScreen = new MenuScreen();
    display = Display.getDisplay(this);
    display.setCurrent(menuScreen);
  }

  protected void startApp() throws MIDletStateChangeException   {
  }

  protected void pauseApp() {  }
  protected void destroyApp (boolean flag) throws MIDletStateChangeException {}

  public void commandAction (Command cmd, Displayable dis) {

  }
}
```

MenuScreen.java

There really isn't much code, what we need to do here is center the menu items based on the number items and the size of font. Because the highlight menu font is enlarged we base space between menu times on the larger font. See the code comments for more information. Its farily self-explantory

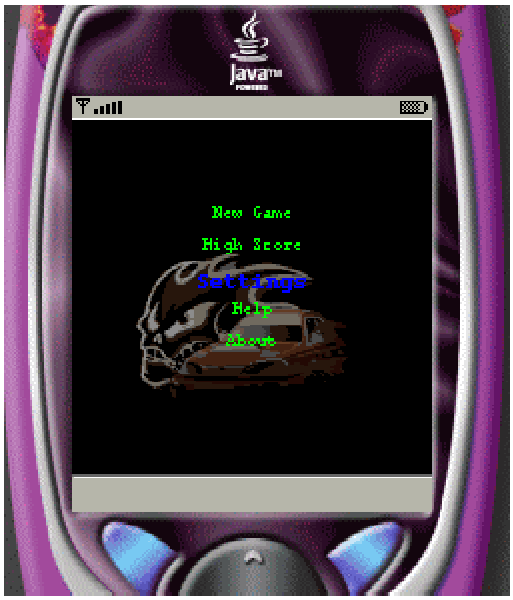
```
import javax.microedition.lcdui.*;

public class MenuScreen extends Canvas implements Runnable {

  // Set Fonts
  static final Font lowFont  = Font.getFont (Font.FACE_MONOSPACE,
Font.STYLE_PLAIN, Font.SIZE_SMALL);
  static final Font highFont = Font.getFont (Font.FACE_MONOSPACE,
Font.STYLE_BOLD, Font.SIZE_MEDIUM);

  // Set Color
  static final int lowColor  = 0x000000FF;    // Not Highlighted
  static final int highColor = 0x00FF0000;    // Highlighted
  static final int highBGColor = 0x00CCCCCC;  // Highlighted Background

  static int width;   // screen width
  static int height;  // screen height

  static int startHeight;  // height where the menu starts

  static final int spacing = highFont.getHeight()/2;  // spacing between menu items

  // Menu Item Labels
  static final String[] mainMenu = {"New Game","High
Score","Settings","Help","About"};

  // To hold the current highlighted menu option
  static int menuIdx;

  // Menu Thread
  Thread menuThread;

  // Constructor
  public MenuScreen() {

    // Get Width and Height of Canvas
    width = getWidth();
    height = getHeight();
```

```
    // Calculate Start Height of Menu
    startHeight = (highFont.getHeight() * mainMenu.length) + ((mainMenu.length-1) *
spacing);
    startHeight = (height - startHeight) / 2;

    // Set Selected Menu Item to the first menu item
    menuIdx = 0;

    // Create Thread and Start
    menuThread = new Thread(this);
    menuThread.start();
  }

  // Simple Run -- Should be modified for better performance/efficiency
  public void run() {
    while(true) {
      repaint();
    }
  }

  // Paint Main Menu
  public void paint(Graphics g) {

    g.setColor(0x00FFFFFF);
    g.fillRect(0,0,width,height);


    for (int i=0; i<mainMenu.length; i++) {

      if (i==menuIdx) {
          g.setColor(highBGColor);
          g.fillRect(0,startHeight + (i*highFont.getHeight()) +
spacing,width,highFont.getHeight());
        g.setFont(highFont);
        g.setColor(highColor);
        g.drawString(mainMenu[i],
                  (width - highFont.stringWidth(mainMenu[i])) / 2,
                   startHeight + (i*highFont.getHeight()) + spacing,
                   20
                  );
      } else {
        g.setFont(lowFont);
        g.setColor(lowColor);
        g.drawString(mainMenu[i],
                  (width - lowFont.stringWidth(mainMenu[i])   ) / 2,
```

```
                    startHeight + (i*highFont.getHeight()) + spacing,
                    20
                   );
      }
    }
  }

  // Capture Keypresses for Menu Selection
  protected void keyPressed (int code) {
    if (getGameAction(code) == Canvas.UP && menuIdx - 1 >= 0) {
      menuIdx--;
    } else if (getGameAction(code) == Canvas.DOWN && menuIdx + 1 <
mainMenu.length) {
      menuIdx++;
    }
  }
}
```

Improving the Simple Highlight Menu

Well I won't go through all the different ways of improving the menu simply because there is probably at least a dozen basic alternations you can do simply by incorporating images. For example, one very simple improvement would be to add a background image.



Widget Inputs

Character Input

Chapter Unknown

Grid Computer
Rendezvous
P2P

Competition
Java Vs MS Vs Symbian Vs Palm Vs Brew

Reminder to self -→ Possible other chapters

- 2D Transformations
- Isometric Tiles
- AI

Multiplayer

Appendix A – Running the Book Examples

Appendix B – Automate Build Process

Using Ant

Using Ant & Antenna

Appendix C – OTA