J2ME & Gaming

By Jason Lam
http://www.jasonlam604.com/
<table>
<thead>
<tr>
<th>Who</th>
<th>Date</th>
<th>Comment</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jason Lam</td>
<td>Sept 1</td>
<td>Initial Release – Original released as separate chapters/files</td>
<td>0.5.0</td>
</tr>
<tr>
<td>Jason Lam</td>
<td>Sept 18</td>
<td>All Chapters are merge into one document</td>
<td>0.5.1</td>
</tr>
</tbody>
</table>
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.

Who Wrote this Book

Author
Jason is a wireless and open source developer enthusiast who enjoys creating synergy and sharing knowledge in the software development world. To learn more about him visit his personal site at http://www.jasonlam604.com

Editor
Frank is part owner and lead developer at Design Interactive Group, http://www.di-group.net, a web development and design firm. Frank is a wireless enthusiast who enjoys developing applications that are useful to everyday consumers. To learn more about him visit his personal website at http://www.frankmanno.com
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:


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](http://wireless.java.sun.com). Other development kits are available at:

<table>
<thead>
<tr>
<th>Who</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nokia</td>
<td><a href="http://forum.nokia.com">http://forum.nokia.com</a></td>
</tr>
<tr>
<td>SonyEricsson</td>
<td><a href="http://www.sonyericsson.com">http://www.sonyericsson.com</a></td>
</tr>
<tr>
<td>Motorola</td>
<td><a href="http://www.motocoder.com">http://www.motocoder.com</a></td>
</tr>
<tr>
<td>Nextel</td>
<td><a href="http://developer.nextel.com">http://developer.nextel.com</a></td>
</tr>
<tr>
<td>Siemens</td>
<td><a href="http://www.siemens-mobile.com">http://www.siemens-mobile.com</a></td>
</tr>
<tr>
<td>Symbian</td>
<td><a href="http://www.symbian.com">http://www.symbian.com</a></td>
</tr>
<tr>
<td>BellMobility</td>
<td><a href="http://www.developer.bellmobility.ca/">http://www.developer.bellmobility.ca/</a></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Who</th>
<th>WebSite</th>
</tr>
</thead>
</table>
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 somewhat 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\textsuperscript{TM} – 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:

- 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.

**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 Wolfenstein, 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:

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

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:

- Infosync – [http://www.infosync.no](http://www.infosync.no)
- WGR (Wireless Gaming Review) - [http://www.wirelessgamingreview.com](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.

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

*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:

<table>
<thead>
<tr>
<th>Company Name</th>
<th>Type of Company</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>AirG</td>
<td>Game Company</td>
<td><a href="http://www.airg.com">http://www.airg.com</a></td>
</tr>
<tr>
<td>AnfyMobile</td>
<td>Game Company</td>
<td><a href="http://www.anfymobile.com">http://www.anfymobile.com</a></td>
</tr>
<tr>
<td>Nokia</td>
<td>Manufacturer</td>
<td><a href="http://forum.nokia.com">http://forum.nokia.com</a></td>
</tr>
<tr>
<td>WirelessDeveloper</td>
<td>Agency</td>
<td><a href="http://www.wirelessdeveloper.com">http://www.wirelessdeveloper.com</a></td>
</tr>
<tr>
<td>TiraWireless</td>
<td>Aggregator</td>
<td><a href="http://developer.tirawireless.com">http://developer.tirawireless.com</a></td>
</tr>
<tr>
<td>Nextel</td>
<td>Carrier</td>
<td><a href="http://developer.nextel">http://developer.nextel</a></td>
</tr>
<tr>
<td>T-Mobile</td>
<td>Carrier</td>
<td><a href="http://www.tmobile.com">http://www.tmobile.com</a></td>
</tr>
<tr>
<td>CellMania</td>
<td>Aggregator</td>
<td><a href="http://www.cellmania.com">http://www.cellmania.com</a></td>
</tr>
<tr>
<td>Cingular</td>
<td>Carrier</td>
<td><a href="http://alliance.cingularinteractive.com">http://alliance.cingularinteractive.com</a></td>
</tr>
<tr>
<td>Telus</td>
<td>Carrier</td>
<td><a href="http://www.telus.net">http://www.telus.net</a></td>
</tr>
<tr>
<td>BellMobility</td>
<td>Carrier</td>
<td><a href="http://www.developer.bellmobility.ca">http://www.developer.bellmobility.ca</a></td>
</tr>
<tr>
<td>4thPass</td>
<td>Aggregator</td>
<td><a href="http://www.4thpass.com">http://www.4thpass.com</a></td>
</tr>
</tbody>
</table>
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 1 - Illustrate Difference in Screen Size Matters](image)

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 than 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 2- Illustrate Difference in Interface Matters](image-url)
**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](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
- 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.

- 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.
- Accessing class variables directly is faster then using setter and getter methods
- Using local variables are more efficient then instance/class variables
- Using variables are more efficient then arrays.
- Avoid synchronization in loops because there is an extra over head to lock and unlock each time the loop occurs
- Counting down in loops is faster then counting up
- Use compound operators like x += 1 instead o f x = x + 1 because fewer byte codes is generated
- Remove constant calculations in loops
- Reuse objects
- Assign null to unused objects, especially unused threads
- 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.

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

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 in 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 overall effectiveness of game play. If the player finds the game too unreasonable from reality he/she will become discouraged 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 3 - 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 an 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 Process 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 of 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 breakdown. This would be your development layout by default this is somewhat 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 formulating 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 than one developer working on one project. This ensures all code is checked in and checked out without loss or overwritten code. Source control gives you further management with respect to different stages of testing from development, test, quality acceptance and ultimately production.

Even as a 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 an 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.

```java
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:

```java
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 4 - Simple GameCanvas Example Source Code
Figure 5 - 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 6 - Example of a Sprite](image)

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 7 - A Sprite Set](image)

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

- ```java
   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 8- Sprites Set with Grid Overlay](image)
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  
\[2^{24} = 16.7\,\text{million}\] colors  
\[2^{32} = 16.7\,\text{million} + \text{an 8-bit alpha channel}\]

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.

![Collision of Sprites](image)

*Figure 9- Sprite Collision*

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

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

or

```java
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 10 - 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:

<table>
<thead>
<tr>
<th>Static Fields</th>
<th>Integer Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANS_NONE</td>
<td>0</td>
</tr>
<tr>
<td>TRANS_MIRROR_ROT180</td>
<td>1</td>
</tr>
<tr>
<td>TRANS_MIRROR</td>
<td>2</td>
</tr>
<tr>
<td>TRANS_ROT180</td>
<td>3</td>
</tr>
<tr>
<td>TRANS_MIRROR_ROT270</td>
<td>4</td>
</tr>
<tr>
<td>TRANS_ROT90</td>
<td>5</td>
</tr>
<tr>
<td>TRANS_ROT270</td>
<td>6</td>
</tr>
<tr>
<td>TRANS_MIRROR_ROT90</td>
<td>7</td>
</tr>
</tbody>
</table>

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:

```java
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);
        }
    }
Figure 11 - Basic Sprite Example Source Code
Figure 12 - Simple Sprite Example Emulator Screen Shot
Extending the Sprite Class

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

1. Add additional Sprite functionality
2. Use the Sprite class as your base Sprite class for all game sprites
3. Encapsulate 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 sprite’s 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 your 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 and complete it to your customized requirements.

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

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 = positionY;
    }
    // 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.

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.

Figure 13- Z Axis Illustration
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 14 - setViewWindowIllustration
```

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:**

```java
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);
    }
```
public void start() {
    isPlay = true;
    Thread t = new Thread(this);
    t.start();
}

public void stop() { isPlay = false; }

public void run() {
    Graphics g = getGraphics();
    while (isPlay) {
        input();
        drawScreen(g);
        try { Thread.sleep(delay); }
        catch (InterruptedException ie) {} 
    }
}

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 15 - Simple LayerManager 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).

```java
// Method to Display Graphics
private void drawScreen(Graphics g) {
    g.setColor(0xffffff);
    g.fillRect(0, 0, getWidth(), getHeight());
    g.setColor(0x0000ff);
    playerSprite.setPosition(currentX,currentY);
    // display all layers
    layerManager.setViewWindow(55,20,140,140);
    layerManager.paint(g,20,20);
    flushGraphics();
}
```

Figure 16 - LayerManager using setViewWindow method Screen Shot
One more note you will notice just using the setViewWindow method the user viewing area will appear off 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 feedback to the user. For example, you can now display the high score, current user score, level and number of lives.
Figure 17 - Display Extra Display Features Demo Screen Shot
LayerManager and Scrolling Background

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

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.*;

Figure 18- Scrolling View
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 19 - 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:

![TiledLayer Example Screen Shot](image)

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.
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.

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

\[
\text{TiledLayer(int columns, int rows, Image image, int tileSizeWidth, int tileSizeHeight)}
\]

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 at 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:
```java
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
    };
    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 an 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, 4, 1, 1, 1, 1, 1, 6,
        5, 1, 3, 1, 1, 1, 1, 1, 6,
        5, 1, 2, 1, 2, 1, 1, 1, 6,
        5, 1, 2, 3, 1, 2, 1, 1, 6,
        5, 1, 4, 2, 1, 2, 1, 1, 6,
        5, 1, 1, 4, 1, 2, 1, 1, 6,
        5, 1, 1, 1, 1, 4, 1, 1, 6,
        5, 1, 1, 1, 1, 1, 1, 1, 6,
        5, 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);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
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

Coming Soon!!!
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)

*INCOMPLETE – To be finished when book is complete*
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 overall 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;
    }

    public void display() {
        // splash screen code
    }
}

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:

![Image of the Splash Screen]

*Figure 23 - "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.

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

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visually more appealing</td>
<td>More design and implementation</td>
</tr>
<tr>
<td>Re-enforces company branding</td>
<td>More resources, ie: may require my graphics to be done</td>
</tr>
<tr>
<td>Re-enforces client branding</td>
<td>Possibly more code, thus large overall jar size</td>
</tr>
<tr>
<td>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</td>
<td>If over customized to fit a certain look and feel the menu may not be re-useable in another game</td>
</tr>
<tr>
<td></td>
<td>May require more resources to when changes are made to the menu, ie new icons are needed for new menu options.</td>
</tr>
</tbody>
</table>
**Basic Main Menu**
The following diagram best illustrates a very basic menu flow.

![Basic Menu Flow Diagram](image-url)

*Figure 24 - 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 overwhelm 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:**

```java
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;
            }
        } catch (Exception e) {
            alert = new Alert(midlet.display, Alert.ERROR, e.getMessage(), Alert.CLOSE);
            alert.show();
        }
    }

    private void scnNewGame() {
        // Code to start a new game
    }

    private void scnSettings() {
        // Code to show settings
    }

    public void displayAlert(Alert alert) {
        Alert.show(midlet.display, alert);  // Show the alert
    }
}
```

http://www.jasonlam604.com/
```java
    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"

Main Midlet Source Code:

```java
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:

![Basic Menu Screen Shot](image)

*Figure 25 - 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 26 - 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:**

```java
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);
        }
    }
}

public class Eliminator implements GameLife {
    public Eliminator() {
        // Initialization code
    }

    public void startGame() {
        // Start game logic
    }

    public boolean isGameOver() {
        // Game over logic
        return false;
    }

    public int getScore() {
        // Score logic
        return 0;
    }
}

public class GameScreen extends List implements CommandListener {
    private Eliminator midlet;
    private Command continueCommand = new Command("Continue", Command.CONTINUE,1);
    private Command quitCommand = new Command("Quit", Command.EXIT,1);

    public GameScreen(Eliminator midlet) {
        super("Game Screen",Choice.IMPLICIT);
        this.midlet = midlet;
        append("Continue",null);
        append("Quit",null);
        addCommand(continueCommand);
        addCommand(quitCommand);
        setCommandListener(this);
    }

    public void commandAction(Command c, Displayable d) {
        if (c == continueCommand) {
            midlet.quitGame();
        } else if (c == quitCommand) {
            midlet.quitGame();
        }
    }
}

J2ME & Gaming
By Jason Lam
http://www.jasonlam604.com/
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("Difficulty 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
JL      50
JL      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:

```java
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 remaining 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:

```java
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
        Version 1.0.0
        By 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 27- Main Menu with 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 forget 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 Eliminator.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 your 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 Eliminator, 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 28 - 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty</td>
<td>integer</td>
<td>1,2 or 3</td>
<td>1 = easy, 2 = medium, 3 = hard</td>
</tr>
<tr>
<td>Auto Fire</td>
<td>integer</td>
<td>1 or 0</td>
<td>1 = on, 0 = off</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>High Score</td>
<td>String</td>
<td>3 Char</td>
<td></td>
</tr>
<tr>
<td>Initials/Name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>High Score Value</td>
<td>integer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Settings and High Score Design

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 necessary 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--) {

This is the end of the code snippet. The code appears to be a class method for handling scores, including loading, getting names, and updating scores. It involves initializing values and names, opening a record store, and processing scores based on their values.
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.

```java
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 {
            this.open();
            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 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 30 - Settings Screen Shot
High Score Screen Shot

Figure 31 – 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, Eliminator.java the GameScreen object is added

```java
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().

```java
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.

```java
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:

```java
// 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:

```java
// 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:

```java
// 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.

```java
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_HEIGHT);

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

```java
// 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 32 – Terrain in defined pieces

Figure 33 – 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) {
                    // Add code here to handle synchronization...
                }
            }
        }
    } catch (InterruptedException e) {
        // Handle InterruptedException...
    }
}
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_HEIGHT);

    // Define Terrain Map
    int[][] map = {
        {0,0,0,0,0},
    }
{3,0,0,0,0,0},
{6,0,0,0,0,0},
{6,0,0,1,2},
{6,0,0,4,5},
{6,0,0,7,8},
{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},
{6,0,0,1,2},
{6,0,0,4,5},
{6,0,0,7,8},
{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},
{6,0,0,1,2},
{6,0,0,4,5},
{6,0,0,7,8},
{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 34 - Multiple Terrains](image)

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 reusable 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.

```java
gameMap = new GameMap(scnViewHeight);
terrain = gameMap.getTerrain();
```
Scroll Terrain
To scroll the terrain the method `scrollTerrain` on the `GameMap` class is called.

```java
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.

```java
terrain = gameMap.getTerrain();
```
Source Code

GameMap.java:

```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},
        {3,0,0,0,0},
        {6,0,0,0,0},
        {6,0,0,1,2},
        {6,0,0,4,5},
        {6,0,0,7,8},
        {6,0,0,0,0},
        {9,0,1,2,3},
        {0,0,4,5,6},
        {0,0,7,8,9},
        {0,0,0,0,0},
        {0,0,0,0,0},
        {0,0,0,0,0},
        {3,0,0,0,0},
        {6,0,0,0,0},
        {6,0,0,1,2},
        {6,0,0,4,5},
        {6,0,0,7,8},
        {6,0,0,0,0},
        {9,0,1,2,3},
        {0,0,4,5,6},
        {0,0,7,8,9},
        {0,0,0,0,0},
        {0,0,0,0,0},
        {0,0,0,0,0},
        {3,0,0,0,0},
        {6,0,0,0,0},
        {6,0,0,1,2},
        {6,0,0,4,5},
        {6,0,0,7,8},
        {6,0,0,0,0},
        {9,0,0,0,0},
    }
```
private static final int[][] map2 = {
    {0,0,0,0,0,0},
    {3,0,0,0,0,1},
    {0,0,0,0,0,0},
    {0,0,0,0,0,0},
    {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},
    {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},
    {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},
    {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 needed 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_HEIGHT);

    // 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;
    }
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();
                }
            }
            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();
    }
More Chapters to come

Soon!!!!