Lethal Metal
The Manual

1. System, Hardware and Software Requirements

Running the jad/jar:
- a device (mobile phone) compatible with the release; the jad/jar of a release contain the device name (eg LethalMetal_Nokia3510i_EN_001.jad), so it is easy to detect what kind of device you need; basically, each phone should have a special-tailored version to cope with specifics like bugs, screen size, heap size, jar size
- a method to transfer the jad/jar to the device; there are several methods to do this, depending, of course, on the device (bluetooth, infrared, usb/serial data cable, wap); for some of the Nokia phones, Oxygen PhoneManager II could be used, it's a very cool program

Developing:
- Nokia 3510i SDK from http://www.forum.nokia.com/main/0,,034-10,00.html (24Mb) (you will need to register on Nokia Forum site to obtain the serial number for the SDK)
- Sun WTK 2.1 from http://java.sun.com/products/j2mewtoolkit/download-2_1.html (16Mb)
- JDK from http://java.sun.com/j2se/1.5.0/download.jsp (55Mb)
- Nokia 3510i SDK needs Windows XP
- You will also need an IDE; actually, you dont need one, you could simply use notepad and the command line ant build system if you feel hardcore, but I do recommend the excelent free Eclipse IDE
- There are no special requirements for the system, except for memory, as usual with java applications

This project has a set of dependencies:
- Apache Ant
- Antenna
- Proguard

It is advisable that you update them to the latest version available.

2. Tools

2.1 Eddie

2.1.1The Tileset

The tileset that will be used by the editor must be called tileset.bmp and must be placed in a directory "data" in the same directory as the eddie.exe.

To edit the tileset, go to Tools|Tile Editor. A modal window will appear.

A tileset is formed of tiles (of course :). Use "add tile", "delete tile", "rename tile" buttons and the combobox to edit the tiles.

A tile has the following properties: a rectangle in the tileset image, collision information (left, right, top, bottom) and ladder behavior (normal ladder or head of a ladder). There is no restriction regarding the collision and ladder flags, but take care that it may result in strange behaviors (like ladders that actually are walls).

For the Lethal Metal java game engine, the tiles must be 16x16 pixels, although you may edit tiles of any size. To easily create the 16x16 tiles, press the "1x1" button.

You may export the tileset by using File|"Export 16bit tileset". Take care that the engine needs the tileset as a file called tid4.dat16

2.1.2 The Tool Windows

Paint Tool is used to select the type of editing (from left to right): placing/writing, fill, erase, select. Only one may be active. These options apply to tiles and items.

Drop Tool is used to select the alignment (horizontal and vertical) of an item to be placed on the map relative to the clicked position.

Log is used to display the results of a map verification.

Tool parameters is used to set the parameters of an item to be placed. The values entered here are stored in data/itemrepository.txt file.

Tool is used to select the item to be placed: tile or item.

2.1.3 Editing a map

You may create a new map by using File|New. Take care that the device heap size will influence the map size. For the Nokia 3510i the maximum map size is recommended to 25x25.

After you create a new map, the map cells will be grayed. This means that the cell has not been edited and has no value: if you select the Tools|Verify in this state, you will notice in the log window several "Brush not set" errors. Take care that the java game engine must be provided a map with all cells (brushes) in edited state, so make sure you check the map before exporting (otherwise, the java game engine may crash).

To place a tile on the map, select the placing style in the Paint Tool (first icon on the left) then select the "Tiles" tab of the Tools window. Select the tile you want from the list and click in the map.

Right clicking on the map acts as quick selection of the tile.

To delete a map cell, select the Paint Tool with the 3rd button on the left, select the "Tiles" tab in the Tool window, then click on the map. You will notice that the map cell has turned gray. Dont forget to edit and put a tile.

You may fill a region with a certain tile by selecting the 2nd button on the Paint window, selecting the Tiles tab of the Tools window and making a selection rectangle on the map. The selection you just made will be filled with the tile.

You may copy and paste pieces of a map. To do this, select the 4th button of the Paint window. Make a selection rectangle on the map. The selection has been placed on the clipboard. Now move the mouse somewhere on the map and press CTRL+V to paste the clipboard at the mouse cursor position. You may save the clipboard selection to a file by using Edit|"Save architecture from clipboard" and load in clipboard from a file with Edit|"Load Architecture to clipboard". Note: this selection will contain both tiles and items.

You may undo and redo any changes you made by using Edit|Undo and Edit|Redo.

In order to place an item on the map, select the desired item in Tools window. Then edit its parameters in the Tool Parameters window (selecting an item in the Tools window will focus on the corresponding tab in the Tool Parameters window). Select the placing type in the Drop window. Make sure the first button is selected in the Paint window. Click on the map :)

In order to remove an object from the map, select the desired type of item in the Tools window. Select the 3rd button (erase) in the Paint window. Left click on the map in the object area to remove it. Take care that trying to delete an object with another type (other tab in Tools window) will fail.

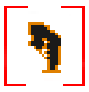
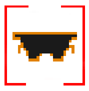
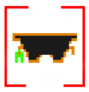
In order to edit the parameters of a object on the map, use CTRL + Left Click over the object area on the map. A window will appear and you may edit the object parameters.

Note: the player object must exist at all times on a map, so it cannot be removed.

Here's a list of the objects (items) in the game and their description:

Item	***Icon***	***Info***
Slasher	↓	A small robot that patrols on horizontal level. It will blow up if collided with the player, dealing certain points of damage.
Shooter	▯	A robot that will patrol an area and will shoot the player on sight. Depending on its agressivity, it will behave differently.

Item	***Icon***	***Info***
Robot Shooter	🤖	Not used, ignored by the engine (although the code exists).
Health	✚	A box that will restore point of healt to the player.
Weapon upgrade	🔫	Not used. Supposed to increase the damage dealt by the player weapon.
Mine	✳ MINE	Explodes on contact with the player, dealing a certian ammount of damage.
Props	🌳	A non-collidable animated object, use to embelish the levels. There are two such props, called ani1 and ani2. You will have to specify the index (1 or 2) and the game engine will chose the sprite.
Ammo	▥	A box that will restore an amount of ammo to the player.
Player	🧍	The player. It has start parameters.
End Level Trigger	END LEVEL	It has a single parameter, the number of the next level. The game engine will look for a map called "map N.map" in jar.
Save Trigger	SAVE GAME	One-time trigger, will save the game if the player touches it. Invisible.
Text Trigger	TEXT	One-time trigger, will display a text. Invisible.

Item	***Icon***	***Info***
Button		Used to call a Normal Elevator, by using the id of the elevator.
Normal Elevator		Normal elevator, can be called by a button object. It will wait for the player to touch it after arriving at destination, before moving to the initial position.
Automatic Elevator		Will move between two positions, waiting a certain time in each position.

2.1.4 Exporting a map

Simply save the map and copy it to the res/device/jar folder in order to be put in the jar and used by the game engine.

The first map must be always called map1.map. The other maps numbering may depend on the End Level Triggers.

2.2 Xsprite

Use this tool to build engine sprites from .bmp files and associate them an action sheet.

Defining the sprite is easy. First, open a bmp file. Opening will cause xsprite to look in the same directory for a file with the same name but with .dat extension (used for cell and points) and for a file with the same name and with extension .actions.

Both .dat and .actions are text files and may be edited with a text editor.

After opening, you may create a cell by pressing "new" button and you may edit its name by pressing the "edit" button.

A cell has two properties: a bounding box and a point (damage point or hot spot). After selecting a cell in the combo box, click the radio buttons to chose bounding obx or damage point and do the action in the sprite sheets. It is easier to do this operation using a zoom.

The actions are used for a more data driven game engine. They represent the logical usage of the cells you defined. Not all sprites have associated actions (only the player, the shooter and the slasher).

In the "Action editor" you may right click to get the editing options.

An action has the following parameters:

- **ID** – it has only information purposes as it is not exported. Basically, each action is indexed as it is stored in the java game engine into an array, the index to acces it in java should be the same as the id in the editor
- **UPDATE** – in frames, represent the time to elapse before changing to the next cell
- **START FRAME/END FRAME** – the indices for the starting and ending cell of this action. Look in the .dat file with a text editor to get this information – it is the row of the cell – 1
- **XALIGN/YALIGN** – may have LEFT/CENTER/RIGHT for xalign and TOP/CENTER/BOTTOM for yalign. Represent the alignment flags that will be used to draw each cell of this action.
- **EFFECT** – NORMAL or FLIPPED, flag to draw the cell
- **INACTIVE** – YES or NO, used for object to object collision detection (during some actions like recovery the player is invulnerable)
- **REPEAT** – how many times to repeat each frame before changing to the next frame
- **XSPEED/YSPEED** – there must be a speed for each cell in this action (2 cells = 2 xspeeds and 2 yspeeds). Represent the offset on the screen regarding the previous position

There is a "Correct" option in the action popup menu. Use it to verify the values before exporting (it will change mispelled words/values or speeds that exceed the number of cells for this action to the default value).

The actions should be exported separately, depending on the needs. Right-click in the action sheet to bring up the pop-up menu and chose "Export".

The sprite must be exported using "Export as 4 bit indexed sprite..." for now.

2.3 DataGrowth

This is the tool to create paks/data/collection files out of existing files (sprites, actions and so on). This is useful because storing each file separately costs 90bytes of jar size.

These files (in Nokia3510i version) may be found in res/Nokia3510i_opt(or ref)/jar as 01.data, 02.data, 03.data and 04.data.

They consist on a list of entries. Each entry is identified by a four characters code. This code is taken from the first four letters of the file that is added or replacing and an existing entry.

These data files are "mounted" by the game engine (CdataFile) and may be further used to obtain data access by other game classes (eg CXSprite) using the 4 character identifier.

3. Developing for Lethal Metal Project

3.1 Directory Structure

The following dir structure/project template allows to easily manage multiple projects (one for each device).

Data is used for storage of data for each device/set of devices and for tools; for the devices, it contains sprites, actions, sounds, maps; for the tools, various graphical resources like icons and so on

Dev contains project setups for different IDEs, both for the java projects and c++ projects; under each IDE, there are folders for each project.

Dist is used to hold the distribution (ready for release) executables, jarjads and data.

Doc is used to store documentation (specifications) of the project.

Lib is used to hold libraries and executables on which this project depends for building. It holds ant, antenna, proguard, kzip and utils.

Res is used to store data that are used to build the jad/jar distribution: data (maps, data files created by the datagrowth tool), basic jad and manifest files. The data is stored on a per-device basis, given the specifics of each device. The data for each project is exported here after a build data target.

Src is used to store sources for both c++ and java. The Java sources are organized on a per-device basis, given the specifics of each device.

Tmp is used to store temporary data from compiling and building processes (.class, . obj and so on). Usually, the ant build system will create the directory needed here.

3.2 The Build System

The build system is based on ant and antenna. For each device, an ant build file exists. This build file is called build.xml and may be found in dev/DEVICE.

It is advisable to read the ant and antenna manuals if you plan to make modifications on the build.xml file/create build.xml files for other devices. Otherwise, simply reading the usage guide is enough.

Using the build system is easy:

build.bat targetname device [params]

Example:

build.bat run Nokia3510i_opt

The whole ant build system revolves around the concepts of tasks and targets. Here is the list of the important targets for this project and their description:

Target	***Info***	***Usage***
run	Runs the jad/jar using the device emulator. It does not obfuscate before running.	build run device
obfuscate	Creates an obfuscated distribution, ready for delivery and testing on the phone	build obfuscate device
dist	Creates a normal (not obfuscated) distribution. It is used by the "run" target.	build dist device
clean	Cleans the tmp/device folder.	build clean device
properties	Sets the project properties: public name and language.	build properties device PUBLICNAME LANGCODE

A version ready to be deployed on the phone has the format (for both jad and jar):

LethalMetal_DevicePublicName_Language_version

Where:
- DevicePublicName might differ than the device name, used for directories in src and dev; as an example, for the Nokia 3510i, the Nokia3510i_opt is the deployable project, that has the public name Nokia3510i
- Language is a two letter code; usually, you will use EN for english, FR for french, DE for german, IT for italian, SP for spanish
- version must be in format majorminormicro, eg 110

Summarizing, we may have for a Nokia3510i_opt device the following name:

LethalMetal_Nokia3510i_EN_100(.jar or .jad)

You may manually change the language and the public name. Open the dev/ant/device/main.properties and make the changes.

The version is not automagically increased. You must do it manually, by opening the dev/ant/device/version.properties and making the necessary changes.

You may need to make changes in the build.xml file, to reflect the paths on your system. Check the following comment in the build.xml file:

```
<!-- This is the place to include SDK specific zips or jars.  -->
<!-- Also, put here the paths that are specific to your dev   -->
<!-- environment.                                             -->
```

You may also need to modify the build.bat file (only once) – to modify the location of the JAVA_HOME.

3.3 Starting development for a new device

Let's assume you want to make a version of Lethal Metal for Motorola triplets (V300/400/500). The public name of this project will be "*MotorolaV300400500*" while the device name will be "*triplets*".

Start by creating a dir called triplets in dev/ant. Copy here build.xml, build-data.xml, main.properties, version.properties from another project.

You will have to alter the build.xml for: paths to the motorola sdk (possibly add to project.classpath a specific motorola class jar) and alter the run target to use the motorola emulator. The emulators do not have standard command line, thus the need to change.

Create a data/triplets folder. Copy here the data to be used by the Motorola version of the game (possibly with larger sprites and so on).

Create a res/triplets folder and copy data from a previous res/device. Modify the res/triplets/jad/LethalMetal.jad (which is the templated jad) for properties specific to the Motorola phone (or device, if appropriate). Also, modify the res/triplets/jad/Manifest.mf for device specific properties.

Create a src/java/triplets folder and copy sources from the master version.

You are now ready to do the Motorola port.

3.4 Using an IDE

Any java ide may be used. For the sake of clarity, it is advisable to create the project in dev/my_preffered_ide/device and to set the output path to tmp/my_preffered_ide/device.