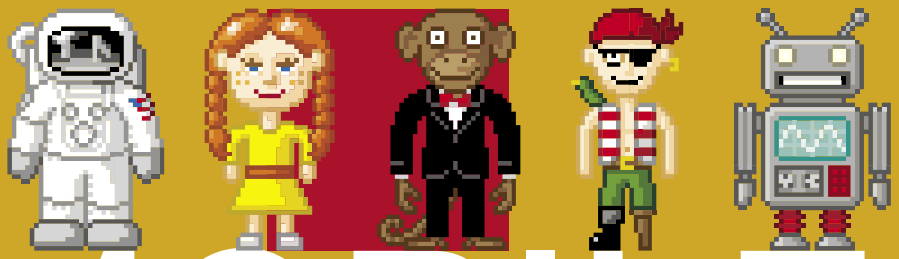
thomas LUCKA



MOBILE GAMES

SPIELEPROGRAMMIERUNG FÜR HANDYS MIT Java ME

HANSER

Inhalt

1 Einleitung – Bevor es losgeht ... 1
1.1 Wozu überhaupt Handyspiele? ... 1
1.2 Voraussetzungen – Für wen ist dieses Buch geschrieben? ... 3
1.3 Hinweise zum Aufbau des Buches ... 4
1.4 Tipps & Tricks: Was Sie beim Lesen beachten sollten ... 5

2 Die Welt der Handyspiele – eine kleine Übersicht ... 7

3 Getting Started ... 13
3.1 Einführung in die Java ME ... 13
3.2 Tools ... 15
3.3 Emulatoren ... 16
3.4 JSRs ... 17
3.5 Hello World ... 20
3.6 Aufbau eines MIDlets ... 25
3.7 Buchbeispiele im Emulator ausführen ... 28
3.8 Installation auf dem Handy ... 29

4 Let The Games Begin – Einführung in die Spieleprogrammierung ... 33
4.1 Was wird benötigt? ... 33
4.2 Das Grundgerüst: MIDlet-Klasse und Lebenszyklus ... 35
4.3 Vorhang auf – die Leinwand (Canvas-Klasse) ... 38
4.4 Fullscreen-Anwendungen ... 41
4.5 Einfache Grafikoperationen (Zeichnen) ... 42
4.6 Bilder laden und anzeigen ... 46
4.7 Arbeiten mit Text (Text anzeigen, Fonts auswählen, Farben) ... 50
4.8 Das Herz des Spiels: Threads/Game Loop ... 54
4.9 Animationen, Bewegungsmuster, Frames ... 60
4.9.1 Rechteckige Bewegung ... 60
4.9.2 Kreisbahn ... 63
4.9.3 Zufallsbewegungen ... 63

4.9.4 Animierte Bewegungen mit Clipping 65
4.10 Input Handling 72
4.11 Kollisionsabfrage 77

5 Ein erstes Spiel – Meteoritenalarm 83
5.1 Das Spielkonzept 83
5.2 Hilfsmittel bündeln – die Toolbox 84
5.3 Meteors-Grundgerüst 88
5.4 Sprites verwalten und einsetzen 91
5.5 Die SpriteTool-Klasse 97
5.6 Eine Klasse für sich – der Spieler 100
5.7 Zentrale Organisation: die Game-Klasse 101
5.8 Von Explosionen und anderen Events 106
5.9 Gegenwehr: Aktionen des Spielers auslösen 113
5.10 Komplexität managen: Flags und State-Machines 116
5.11 Das Spiel abrunden: Scores, Game Over 118
5.12 Backgrounds und weitere Erweiterungen 120

6 Advanced Programming I 123
6.1 Commands vs. Softkey-Codes 123
6.2 GUI-Programmierung 130
6.3 Grafikformate 135
6.4 Sound 140
6.5 Datenspeicherung 158

7 Handyspiele mit Top-down-Perspektive 165
7.1 Die Grundidee 165
7.2 Tiles, Tilemaps & Tile-Editoren 168
7.3 Tilemaps erstellen und einbinden 171
7.4 Aufsetzen des Projektes 176
7.5 Scrolling – das Grundprinzip 185
7.6 Tilemap anzeigen 190
7.7 Levelelemente: Items, animierte Tiles, Hintergründe, Mauern 193
7.7.1 Hintergrundbilder einbinden 193
7.7.2 Animierte Tiles 196
7.7.3 Items aufsammeln 198
7.7.4 Kollision mit Hindernissen 199
7.7.5 Zufällige Positionierung von Sprites 201
7.8 Sprites mit KI ausstatten (Grundlagen Künstliche Intelligenz) 202
7.9 Scions of Mars – das fertige Spiel 211

8 Algorithmen für Jump'n'Run-Spiele 213
8.1 Was ist das Besondere an J'n'R-Spielen? 214
8.2 Spielbeschreibung 214
8.2.1 Steuerung 215
8.2.2 Was gelernt werden soll 216

8.2.3 Die benötigten Grafiken für Mine Hunter ... 216
8.3 Das Grundgerüst ... 218
8.4 Wie man den Spieler zum Hüpfen bringt ... 230
8.5 Seitwärtssprünge ... 236
8.6 Positionierung von Sprites ... 237
8.7 Fortbewegung auf Plattformen ... 242
8.8 Das Spiel vervollständigen ... 248
8.9 Parallax-Scrolling ... 252
8.10 Weitere mögliche Erweiterungen ... 256

9 Advanced Programming II ... 259
9.1 ProGuard als Obfuskator einsetzen ... 259
9.2 Preloader ... 261
9.3 Lokalisierung ... 264
9.4 Textfiles einlesen ... 266
9.5 Custom Fonts ... 268
9.6 GameCanvas (MIDP 2.0) ... 272

10 Willkommen in der dritten Dimension: 3D-Spiele mit Java ME ... 275
10.1 Das Grundgerüst: 3D-Sprites und Ego-Perspektive ... 278
10.2 Positionierung und Bewegung von 3D-Objekten ... 282
10.3 Sprite3D-Objekte erzeugen ... 285
10.4 Rendern von 3D-Grafik ... 286
10.5 "Kamera läuft ...!" ... 287
10.6 Animation von 3D-Sprites ... 290
10.7 Kollisionsbestimmung mit der Kamera ... 291
10.8 Echte 3D-Modelle: Meshes erzeugen und einsetzen ... 292
10.9 "Es werde Licht ...!" ... 297

11 Multiplayer-Spiele auf dem Handy? Verbindungen ins Internet ... 301
11.1 Client-Server-Kommunikation ... 302
11.2 Daten austauschen mithilfe von PHP ... 304
11.3 User Permissions ... 308
11.4 Das Request-/Response-Prinzip von HTTP ... 310

12 Ausblick ... 317
12.1 Eclipse mit J2ME-Plugin einsetzen ... 317
12.2 Portierung(en) – wozu denn das? ... 319
12.3 Automatisierung: Ant mit Antenna einsetzen ... 322
12.4 Und was kommt nun? Wie ein Spiel auf den Markt gelangt ... 329

Literatur ... 333

Register ... 335

4

4 Let The Games Begin – Einführung in die Spieleprogrammierung

After working my way through the alphabet soup of J2ME, CLDC, and MIDP,
I've found that writing for the platform is pretty easy.

— John Carmack

4.1 Was wird benötigt?

Wenn Sie damit beginnen, eine neue Programmiersprache zu lernen, steht am Anfang der Umgang mit den benötigten Tools: Sie müssen lernen, wie man den Code aufsetzt, kompiliert und ausführen kann. Diese Schritte haben wir anhand eines einfachen Hello World-Beispieles für die Java ME bereits nachvollzogen.

Wie geht es nun weiter? Da sich dieses Buch an Einsteiger richtet, die zwar über grundlegende Java-Kenntnisse verfügen, ansonsten aber noch keine Erfahrungen in Bezug auf die Java ME und die Spieleprogrammierung mitbringen, werden wir in den folgenden Abschnitten mit möglichst einfach gehaltenen und in sich abgeschlossenen Beispielen beginnen. Schwerpunktmäßig steht dabei natürlich die Spieleentwicklung im Vordergrund. Da Spiele zu den komplexeren Anwendungen zählen, die u.a. Kenntnisse der Grafikverarbeitung, Datenspeicherung, Datenvernetzung, Benutzerschnittstellen und Soundverarbeitung beinhalten, decken diese Themen auch viele wichtige Aspekte anderer bzw. angrenzender Bereiche der mobilen Anwendungsentwicklung ab. Sie können die Kenntnisse, die in den nachfolgenden Kapiteln erworben werden sollen, daher nicht nur – aber auch – für Spiele einsetzen.

Um sich in neues Themengebiet einzuarbeiten, ist es hilfreich, sich erst einmal über die Fragen klar zu werden, die man an das jeweilige Thema hat. Unabhängig von der Wahl der Programmiersprache brennen Ihnen als angehender Spieleprogrammierer zu Beginn sehr wahrscheinlich die folgenden Fragen auf den Nägeln:

- Wie können Bilder geladen werden? Welche Grafikformate sind möglich?
- Wie können die Bilder auf dem Screen angezeigt werden?
- Wie kann Text auf dem Screen angezeigt und positioniert werden?
- Wie kann der Screen in regelmäßigen Abständen erneuert werden? Wie lässt sich zu diesem Zweck ein einfacher Thread realisieren?
- Welche Eingabemöglichkeiten gibt es, und wie können diese abgefragt werden?

Wenn Sie diese grundlegenden Fragen beantworten können, sind Sie schon ein ganzes Stück weiter und können im Prinzip mit der Verwirklichung der ersten eigenen Spiele anfangen. Die Antworten auf die Fragen hängen jedoch von der jeweiligen Programmiersprache ab. Wenn Sie bereits wissen, wie sich die angesprochenen Aufgabenstellungen mithilfe der Java SE realisieren lassen, so wird Ihnen der Einstieg in die Java ME nicht besonders schwerfallen. Viele Methoden und Konzepte wurden in leicht abgewandelter Form übernommen, da aber auch Neues hinzugekommen ist und bei der Programmierung mit der Java ME einige Besonderheiten zu beachten sind, werden wir ausführlich auf diese eingehen. Auch als Neuling, der bisher nur wenig mit der Java SE zu tun hatte, sollte Ihnen daher der Einstieg nicht allzu schwerfallen.

Um aufwendigere Spiele zu programmieren, benötigen Sie weitere Techniken und/oder Hilfsmittel. In der Regel handelt es sich dabei um Datenstrukturen und Algorithmen, die Ihnen helfen, bestimmte in der Spieleprogrammierung häufig wiederkehrende Aufgaben zu erledigen. Die Antworten auf die folgenden Fragen hängen weniger von einer bestimmten Programmiersprache ab, sondern lassen sich auch auf andere Sprachen, beispielsweise C/C++ oder Flash/ActionScript, übertragen:

- Wie kann festgestellt werden, ob zwei sich bewegende Körper miteinander kollidieren?
- Wie können Animationen und Bewegungsabläufe dargestellt werden?
- Auf welche Art und Weise lassen sich effizient häufig wiederkehrende Ereignisse auslösen, wie z.B. Explosionen oder Game Over-Sequenzen?
- Wie können Levelmaps platzsparend erzeugt werden?
- Welche Algorithmen kann man einsetzen, um diese Levelmaps mit wenig Aufwand zu scrollen?
- Wie lassen sich einfache GUIs erzeugen?
- Wie lassen sich Texte mit eigenen Schriftarten darstellen?

Da diese Fragen sehr häufig gestellt werden, finden Sie in den meisten Programmbibliotheken moderner Programmiersprachen viele vorgefertigte Methoden, welche die angesprochenen Probleme lösen können. Vorgefertigte Bibliotheken bieten oft den Vorteil, dass Teilprobleme mittels Hardwareunterstützung gelöst werden können. Dies ist bei einer rein softwareseitigen Lösung über geeignete Algorithmen nicht möglich. Seit MIDP 2.0 stellt die Java ME mit der integrierten Game API über das User Interface Package `javax.microedition.lcdui.game` Methoden u.a. zur Kollisionsbestimmung und zur Erstellung und Verwaltung von Levelmaps zur Verfügung. Da die API auf Hardwarebe-

schleunigung verzichtet und ihrerseits nur auf die Standardklassen von MIDP 1.0 zurückgreift, können Sie durch die Verwendung der API bestenfalls die JAR-Size verringern, nicht aber die Performance verbessern. Durch die Verwendung der Game API sind Sie zudem an das MIDP 2.0-Profil gebunden und können viele, vor allem ältere, MIDP 1.0-Handsets nicht mehr unterstützen. In der Praxis wird die Game API daher nur in Ausnahmefällen eingesetzt. Gerade bei der Programmierung für ressourcenschwache Geräte wie Mobiltelefone ist es außerdem hilfreich, sich genauer mit den Techniken auseinanderzusetzen, die hinter der Game API stehen. Neben der Beschreibung der wichtigsten Neuerungen, die das MIDP 2.0-Profil bietet, wird versucht, im weiteren Verlauf des Buches die oben aufgeworfenen Grundlagenprobleme möglichst flexibel und allgemeingültig zu lösen.

Abschließend gibt es noch eine dritte Gruppe von Fragen, die Aufgabenstellungen ansprechen, deren Lösungen Zugriffe auf die Hardware des Zielgerätes erfordern:

- **Sound:** Welche Soundformate können wie abgespielt werden?
- **Datenspeicherung:** Wie können Daten dauerhaft gespeichert werden?
- **Connectivity:** Welche Möglichkeiten der Vernetzung bestehen, und wie können diese genutzt werden?

Diese letzte Gruppe von Fragen ist mehr als ein Add-on zu verstehen. Sie können schließlich auch Spiele ohne Soundeffekte und Musik entwickeln, auch das Speichern von Spielständen ist nicht zwingend notwendig (vergeben Sie z.B. pro Level einen Code, den der User sich merken muss). Dennoch können Sie durch diese Add-ons den Spieleindruck des Users aufwerten, daher werden wir uns nachfolgend zu den Themen Sound und Datenspeicherung mehrere Beispiele in Form kurzer und wiederverwendbarer Bausteine ansehen. Dem Thema Connectivity werden wir im Hinblick auf Multiplayer-Spiele ein eigenes Kapitel widmen.

4.2 Das Grundgerüst: MIDlet-Klasse und Lebenszyklus

Also dann, gehen wir frisch ans Werk. Nach diesen Vorüberlegungen soll es endlich richtig losgehen.[1] Zunächst greifen wir noch einmal unser Hello World-Beispiel aus dem vorigen Abschnitt auf. Den Quelltext zu diesem Beispiel finden Sie im SRC-Verzeichnis als separates WTK-Projekt mit dem Ordnernamen "MobileGames_01_HelloWorld". Sofern noch nicht geschehen, kopieren Sie dieses bitte in das apps-Verzeichnis des WTK, und starten Sie das Projekt über File -> Open Project.

Der Aufbau eines MIDlets ist nicht schwer zu verstehen: Das Hello World-Beispiel ist in zwei Klassen unterteilt, die erste Klasse, `MyMIDlet`, stellt die zentrale MIDlet-Klasse dar, die für jede Java ME-Anwendung gebraucht wird. Die zweite Klasse, `ActionCanvas`,

[1] Die Beispiele bzw. Bausteine dieses Kapitels richten sich primär an absolute Neueinsteiger und können daher von fortgeschrittenen Lesern übersprungen werden. Viele der gezeigten Bausteine werden in späteren Kapiteln aufgegriffen und erweitert.

stellt unsere Leinwand dar, sozusagen die Zeichenfläche, auf der wir unser Spiel zeichnen können. Neben der Leinwand (engl.: Canvas) gibt es noch eine weitere Möglichkeit, mittels vordefinierter Screen-Elemente Zugriff auf das Handydisplay zu bekommen. Wir werden diese sogenannten "High-Level" User-Interface-Klassen erst in einem späteren Kapitel kennenlernen, da diese für die Spieleentwicklung eine eher untergeordnete Rolle spielen.[2]

In der `MyMIDlet`-Klasse werden über die Importanweisungen die benötigten Klassen bekannt gemacht:

```
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.Display;
```

Falls Sie mehrere Klassen eines Packages benötigen, können Sie auch die Wildcard "`.*`" einsetzen, also z.B. "`import javax.microedition.lcdui.*;`". Auf die Performance oder die Speicherbelegung hat dies keinen Einfluss. Um auf den ersten Blick deutlich zu machen, welche Klassen benötigt werden, ist es sinnvoll, die Importanweisungen auszuschreiben. So ist erkennbar, dass die `MyMIDlet`-Klasse zwei weitere Klassen, `MIDlet` und `Display`, benötigt. Doch welchen Zweck erfüllen die beiden Klassen aus den Packages `midlet` und `lcdui`?

Zunächst einmal: Über die `Display`-Klasse des UI Packages können wir Zugriff auf den Handyscreen erhalten. Die genaue Bedeutung erschließt sich erst im Zusammenhang mit der `ActionCanvas`-Klasse, daher werden wir weiter unten ausführlicher auf die `Display`-Klasse zurückkommen. Die `MIDlet`-Klasse hingegen bildet sozusagen des Zentrum der Entwicklung mit der Java ME: Der Name "`MIDlet`" deutet es bereits an, MIDlets ähneln in ihrer Struktur den aus dem Internet bekannten Applets der Java-Sprache. Unser MIDlet trägt den Namen "`MyMIDlet`" und ist von der Elternklasse `MIDlet` abgeleitet, die drei abstrakte Methoden enthält – dies bedeutet, dass jede unserer eigenen `MIDlet`-Klassen ebenfalls über diese Methoden verfügen muss. Der Grund besteht darin, dass die zentrale Organisation des Lebenszyklus eines MIDlets von den Schnittstellenmethoden `startApp()`, `pauseApp()` und `destroyApp()` abhängt.

Die *externe* Steuerung des Lebenszyklus erfolgt dabei über das sogenannte Application Management System (AMS, dt.: Anwendungsmanagementsoftware). Dies ist ein umschreibender Name für den Teil des jeweiligen Betriebssystems des Mobiltelefons, der den Zugriff auf Java ME-Anwendungen regelt. Ein MIDlet kann sich in drei unterschiedlichen Zuständen (engl.: State) befinden: `active`, `paused`, `destroyed`. Das AMS steuert diese States (und die Übergänge von einem State zu einem anderen) folgendermaßen:

- **`startApp()`:** Diese Methode wird vom AMS aufgerufen, wenn die Anwendung durch den User auf seinem Handy gestartet wird. Damit stellt diese Methode den zentralen

[2] Wenn Sie neu in der Programmiersprache Java sind, empfiehlt es sich, sich zunächst ein wenig mit den Grundlagen dieser Sprache vertraut zu machen. Einen leicht verständlichen Überblick finden Sie z.B. bei [Niemann], ausführlicher: [Louis]. Eine detailliertere Beschreibung der Bibliotheken der Java ME können Sie u.a. in den Büchern von [Breymann] oder [Schmatz] nachlesen.

Einstiegspunkt dar, vergleichbar zur `main()`-Methode in C/C++. Nach dem Aufruf dieser Methode befindet sich unser MIDlet im aktiven Zustand (engl.: active).

- **`pauseApp():`** Wird das AMS aufgrund von externen Events gezwungen, die Anwendung vorübergehend zu pausieren, z.B. durch einen eingehenden Telefonanruf, dann wird vor der "Zwangspause" die `pauseApp()`-Methode aufgerufen. Sinnvollerweise können Sie in dieser Methode also z.B. den aktuellen Spielstand speichern oder auch das Spiel komplett anhalten. Ist das externe Event beendet, ruft das AMS erneut die `startApp()`-Methode auf.
- **`destroyApp(boolean unconditional):`** Falls die Anwendung beendet wird, wird zuvor die Methode `destroyApp()` aufgerufen. Eine sinnvolle Verwendung besteht z.B. darin, in dieser Methode Ressourcen freizugeben, laufende Sounds zu stoppen oder Spielstände zu speichern. Übergeben wird der Methode die boolesche Variable `unconditional`, die uns mitteilt, ob dem Aufruf der Methode unbedingt (engl.: unconditional) Folge zu leisten ist. Wenn `unconditional` den Wert `true` besitzt, muss unsere Applikation also in den Destroyed-State wechseln, ob wir wollen oder nicht. Besitzt `unconditional` den Wert `false`, dann könnten wir eine `MIDletStateChangeException` auswerfen, um zu signalisieren, dass unser MIDlet noch nicht zerstört werden soll. Dies wird aber in der Praxis relativ selten vorkommen, sodass wir uns nicht weiter darum kümmern wollen. Stattdessen lassen wir zu, dass die Anwendung "brav" beendet wird.

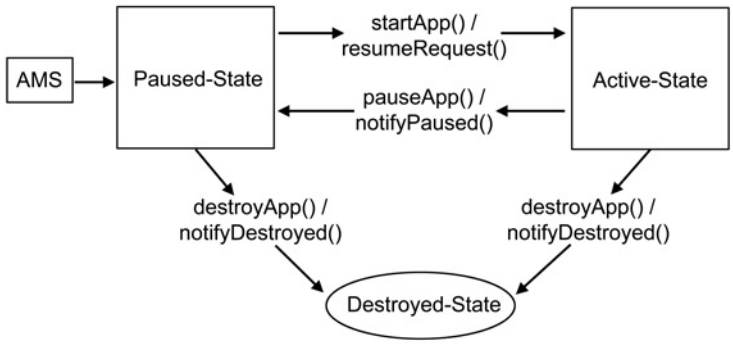


Abbildung 4.1 MIDlet-Lebenszyklus

Kann man auch selbst den Wechsel eines States bewirken? Mittels `resumeRequest()`, `notifyPaused()` und `notifyDestroyed()` können Sie dem AMS mitteilen, in welchen Zustand Sie gerne wechseln möchten. Die Entscheidung, ob dies dann tatsächlich erfolgt, liegt immer in letzter Instanz beim AMS.

- **`resumeRequest():`** Rufen Sie diese Methode auf, wenn sich Ihr MIDlet im Paused-State befindet, und signalisieren Sie darüber dem AMS, dass Ihr MIDlet gerne in den aktiven Zustand (Active) wechseln möchte. Dies ist in der Praxis nur sinnvoll, wenn

Register

3

3D-Spiele 275

A

Alpha Blending 46
AMS 14, 27, 36, 37
Android 4
Animation 56, 60, 197, 290
Ant 319, 322
Antenna 322
APN 303
Applet 36
ARGB 46, 137, 140

B

Beschleunigung 232
Bilder 46
Billboards 278, 292
Billing-Schnittstelle 330
Bit-Shifting 63, 140
Bluetooth 14, 19
Blur-Effekt 320
Bounding-Box 78, 166, 198, 199
BREW 4
Build-Prozess 319

C

Chunk 66
CLDC 14, 18, 21
Clipping 65, 189
Commands 41, 124
Connected Limited Device Configuration
siehe CLDC
Custom Fonts 50, 268

D

Datenspeicherung 35, 158
Dekompilierung 259
Display 36, 38, 43, 165, 186, 319
Double Buffering 185
Drumcomputer 144

E

Eclipse 15, 317
EclipseME 317
Einheitsmatrix 282
Events 106

F

Flags 116
Flash Lite 4

Font 51
FPS 58
Frame 56, 65, 66, 68, 237
Fullscreen 41, 126

G

Game Actions 76
Game API 34, 35, 273
Game Loop 54
Gameboy Advance 7, 12, 332
Gateway 303
Gelegenheitsspieler 9
Generic Connection Framework 302
Geschwindigkeit 57
GET 307
GIF 135
GPRS 301
Grafikformate 135
Grafikkontext 55, 67
GUI 124, 130

H

Halbkreis 44
Head-up Display siehe HUD
Heap 17, 19, 49, 54, 97, 137, 266, 320
Hello World 20, 22, 31
High-Level-API 124
High-Level-GUI 134
Hintergrund 193, 278
HTML 29, 30, 312
HTTP 302, 305, 311
HUD 50, 120, 189

I

Icon 26

J

JAD 24, 26, 159
Jamba 8
JAR 24, 137, 259, 320
Java Application Descriptor siehe JAD
Java Archive siehe JAR
Java Community Process siehe JCP
Java Specification Request siehe JSR
Javadoc 17
JCP 17
JDK 15
J'n'R 213
JPEG 135
JSP 304
JSR 17, 18, 305, 320
Jump'n'Run siehe J'n'R

K

KI 202
Klingeltöne 140
Kollisionsabfrage 77, 205, 291
Kommunikation 301
Koordinatensystem 39, 186, 277
Kreis 44
Künstliche Intelligenz siehe KI
KVM 14, 24

L

Lebenszyklus 27, 36
Leinwand 36, 38, 41, 123, 274
Levelmap siehe Tilemap
Lokalisierung 264

M

M3G 15, 276, 295
Manifest 27
Mesh 292, 293
MIDI 142, 143, 146
MIDlet 21, 25, 35
MIDP 14, 21
MIME 29, 147, 153
MMAPI 140, 141, 144
MMORPG 302
Mobile Information Device Profile
siehe MIDP
Mobile Service Architecture 18
Mobile Service Architecture siehe MSA
Mobilfunk 301
Modulo-Operator 64, 203
MP3 141, 143

MSA 18, 19
Multiplayer-Spiele 301
Multipurpose Internet Mail Extensions
siehe MIME
MySQL 304, 305

N

NetBeans 15, 317
Network-Monitor 315
Netzbetreiber 8, 11, 303, 330
Nintendo DS 7, 72, 275, 332

O

Obfuskator 259, 328
OTA 29
Outlines 54
Over the Air siehe OTA

P

Parallax-Scrolling 252
Performance 54, 319
Persistance 158
PHP 304, 307
PNG 46, 66, 135
Portierung 319
Positionierung 201, 237, 282
POST 307
Präprozessor 319, 322, 329
Preloader 48, 87, 90, 261
Publisher 10, 330

R

Raycasting 276
Record Management System siehe RMS
Record Store 159
Remote Server 302
Request 302, 310
Response 302, 310
Revenue Share 330
RGB 43, 136, 285
Ringtones 141
RMS 158

S

Sampler 144
Screenauflösungen 42
Scrolling 165, 168, 185, 214
Sequencer 144
Servlet 304
Sidescroller 165
Signierung 162
Sinuskurve 63
SNAP 304
Social Gaming 302
Softkeys 41, 50, 74, 76, 123, 127
Softwaresynthese 145
Sony PSP 7, 72, 275, 332
Sound 35, 140
Sprite 86, 91, 278, 290
State-Machine 117
Sternenfeld 120
Steuerkreuz 72
Symbian 4, 275
Synthesizer 144
Systemschriftart 50

T

Tastaturabfrage 72, 73
Telekommunikation 7
Third-Person-Perspektive 275
Thread 48, 55, 58, 73, 261
Tile 168, 191, 196, 217
Tilemap 122, 170, 171, 190, 193, 198
Tileset 168
Toolbox 84, 87
Top-down-Perspektive 166, 216
Transformationsmatrize 282
Translation 189, 282
Transparenz 46

Ü

Übersicht Softkey-Codes 127

U

UMTS 301
Unicode 264

Unterhaltungsindustrie 7
URL-Kodierung 312
User Permission 163, 309

V

Vermarktung 329, 330
Vertriebsweg 8, 11
Viewport 165, 186, 253, 291
Virtual Machine 13
Vogelperspektive 214

W

WAP 29, 30, 330, 331
WAV 142
Wertschöpfungskette 10
Wireless Toolkit siehe WTK
WTK 15, 16, 20, 28, 40

X

XML 30, 322

Z

Zeichenoperationen 43
Zufallszahlen 63, 64, 85