

MatheFreak Mobile Dokumentation

Projektdokumentation zum JavaStars Wettbewerb 2006

Martin Fritzsche, Paul Lange, Sören Schmidt

Gruppenname: SöMaPa, Gruppennummer: 239, Schulnummer: 51304

Projektübersicht

Bundesland	Thüringen
Teamnummer	239
Schulnummer	51304
Schulname	Carl-Zeiss-Gymnasium
Schulform	Gymnasium
Name des Teams	SöMaPa
Projektname	MatheFreak Mobile
Projektkurzbeschreibung	Ein Kopfrechentrainer für Java-fähige mobile Geräte. Es gibt einen Trainings- und einen Mehrspielermodus, einen Highscore und ein kleines Bonusspiel, das erst freigeschaltet werden muss.
Unterrichtsfach	Mathematik
Gruppengröße	3 Personen
Alter der Gruppenmitglieder	16 und 17 Jahre
Teilnehmende Mädchen	Nein

Inhaltsverzeichnis

Allgemeine Fragen zum Wettbewerb	4
Wie haben Sie/habt ihr von dem Wettbewerb "JAVA STARS 2006" erfahren?.....	4
Habt Ihr an anderen Wettbewerben teilgenommen?	4
Bei welchen Wettbewerben habt Ihr bisher das Programm schon vorgelegt?	4
Projektbeschreibung.....	5
Projektbeschreibung.....	5
Projektidee.....	5
Thema	5
Unterrichtsfach	5
Nutzen für den Unterricht	5
Zeitaufwand	5
Arbeitsteilung	6
Arbeits- und Testumgebung	6
Installation des Programms	6
Lösungskonzept	7
Aufbau der Lösung.....	7
Eingesetzte Verfahren	7
Programm-Architektur.....	8
Aufgabenerstellung	8
Die Task-Klasse	8
Die Utils-Klasse	8
Aufgabenerstellung	8
Die Low-Level-API.....	8
Die High-Level-API	8
Datenspeicherung.....	10
Wichtige Klassen des RMS.....	10
Die Settings-Klasse.....	10
Laden	10
Speichern.....	11
Die Highscore-Klasse	11
Methodenübersicht.....	11
Einfügen.....	11
Laden	12
Speichern.....	12

Oberflächenerstellung	13
Midlets.....	13
Oberflächenelemente	14
Befehlsbehandlung.....	14
Bluetooth	16
Initialisierung des Stacks	16
Server.....	16
Gerätesuche	17
Client.....	18
Konkrete Anwendung.....	19
BTDiscover	19
Multiplayer-Training.....	20
ScoreSynch	20
Chat	20
Benutzerschnittstelle	21
Benutzeroberfläche	21
Benutzerfreundlichkeit.....	21
Kompatibilität.....	21
Programmablauf	22
Programmstart	22
Das Hauptmenu.....	22
Der Einzelspielermodus.....	22
Der Mehrspielermodus	22
Server.....	23
Client.....	23
Der Highscore	23
Das Einstellungsformular.....	23
Snake – das Bonuspiel.....	24
Referenzen	25
Anlagen	26

Allgemeine Fragen zum Wettbewerb

Wie haben Sie/habt ihr von dem Wettbewerb "JAVA STARS 2006" erfahren?

Wir haben durch ein Poster in unserer Schule, sowie durch unseren Informatiklehrer von JavaStars-Wettbewerb erfahren.

Habt Ihr an anderen Wettbewerben teilgenommen?

Neben dem JavaStars-Wettbewerb haben wir an Mathe- und Physikolympiaden teilgenommen. Außerdem nehmen wir am Bundeswettbewerb Informatik und am Imagine Cup 2007 teil.

Bei welchen Wettbewerben habt Ihr bisher das Programm schon vorgelegt?

Unser Programm MatheFreak Mobile wurde speziell für diesen Wettbewerb geschrieben. Auch eine Version ohne Unterstützung für mobile Geräte existierte vorher noch nicht.

Projektbeschreibung

Projektbeschreibung

Unser Programm soll das Kopfrechnen trainieren. Dazu generiert es zufällig Aufgaben, die jeder mit ein wenig Überlegung lösen kann. Der Mehrspielermodus reizt zudem zu einem kleinen Kräfteessen. Eine Chatfunktion und der Highscoreaustausch regen außerdem zum regen Gebrauch an.

Zur Entspannung (und für langweilige Mathestunden) steht mit Snake außerdem ein kleines Bonusspiel bereit.

Projektidee

Unsere erste Idee war es, ein Programm zur Visualisierung und Erklärung von chemischen Bindungen zu erstellen. Diese Idee wurde aber aufgrund vieler Ausnahmen und Sonderregeln schnell wieder fallen gelassen.

Die zweite Idee ging in Richtung Mathe und Kopfrechnen. Als dann noch das Wort *Handy* fiel war unser Projektthema klar, ein mobiler Kopfrechentainer. Das Thema ist also langsam in der Gruppe gereift.

Thema

Unser Programm dreht sich um Kopfrechnen, welches jedoch auf spielerische Weise trainiert werden soll. Außerdem soll eine Möglichkeit geben auch gegen seine Freund zu spielen.

Unterrichtsfach

Gedacht ist unser Programm hauptsächlich für den Matheunterricht, aber das Training zahlt sich bestimmt auch in Physik oder Chemie aus. So muss man nicht immer zur „elektronischen Schätzkeule“ greifen wenn man ein wenig rechnen muss.

Nutzen für den Unterricht

Unser Programm trainiert neben den Kopfrechenfähigkeiten auch die verschiedenen Rechenoperationen. Da aber bestimmt niemand ein solches Spiel an einem Desktop PC spielen würde, entschieden wir uns für eine mobile Variante. Da man sein Handy immer dabei hat, können auch kleine Freizeiten wie Pausen in der Schule oder die Zeit auf dem Nachhauseweg genutzt werden.

Zeitaufwand

Die genaue Dauer können wir leider nicht angeben, aber wir sind und sicher, dass mehr als 100 Programmier- und Teststunden Arbeit in das Projekt geflossen sind. Programmiert wurde oft im Infonterricht, aber auch viel sehr viel zuhause.

Arbeitsteilung

Martin Fritzsche bearbeitete den Bluetooth Teil und erstellte die Benutzeroberfläche. Paul Lange war für alles verantwortlich, was mit der Datenspeicherung auf dem Handy zu tun hat. Sören Schmidt implementierte die Aufgabengenerierung und passte den Schwierigkeitsgrad an. Trotzdem half jeder jedem, wenn man nicht mehr weiter wusste.

Arbeits- und Testumgebung

MatheFreak Mobile wurde mit Netbeans 5.5 mit dem Mobility-Erweiterungspaket erstellt. Sämtliche Computer, die zur Programmierung genutzt wurden laufen unter Windows.

Das Programm auf dem Netbeans-Emulator und auf verschiedenen Handytypen getestet. Dazu zählen Das BenQ-Siemens C81, das Sony Ericsson 800i und weitere.

Installation des Programmes

Kopieren Sie die Dateien MatheFreak_Mobile.jad und MatheFreak_Mobile.jar auf Ihr Handy. Dies kann zum Beispiel mit Hilfe eines Datenkabels, per Bluetooth oder mittels Infrarot geschehen.

Öffnen Sie auf Ihrem Handy die Dateien MatheFreak_Mobile.jad. Das Programm sollte nun selbstständig installiert werden.

Lösungskonzept

Aufbau der Lösung

Das Programm unterteilt sich in 3 große Bereiche, die auch in eigene Packages aufgeteilt sind. Diese sind die Aufgabenerstellung, der Multiplayermodus und die Benutzeroberfläche.

Der Bereich der Aufgabenerstellung (MatheFreak.Tasks) kapselt, wie der Name vermuten lässt, die Aufgabenerstellung. Außerdem enthält er die Highscore-Klasse.

Das Package MatheFreak.Bluetooth enthält alle für den Multiplayermodus notwendigen Klassen.

Die Klassen der Benutzeroberfläche sind im Package MatheFreak.UI abgelegt. Sie stellen die Benutzeroberfläche der Anwendung dar.

Eingesetzte Verfahren

Das gesamte Programm ist streng objektorientiert aufgebaut. Zur Entwicklung für mobile Geräte werden CLDC 1.1 sowie MIDP 2.0 genutzt. Darin enthalten ist die Nutzung von Bluetooth-Funktionen, des Record-Management-Systems und der Oberflächenerstellung auf Mobilgeräten.

Um unsere Arbeit untereinander zu Koordinieren nutzten wir außerdem das Versionsverwaltungssystem Subversion.

Programm-Architektur

Aufgabenerstellung

Die Task-Klasse

Task.java ist eine Klasse, welche den komplexen Datentyp *Task* erzeugt. Ein *Task* (zu Deutsch Aufgabe) besteht aus dem Aufgabentext, der Lösung der Aufgabe und dem. Zum gibt es eine Funktion *IsRightAnswer(...)*, welche die Eingabe des Spielers auf Richtigkeit überprüft. Dazu wird die Funktion *equals(...)* genutzt. Dabei werden die Lösung der Aufgabe und die Benutzereingabe verglichen. Bei richtigem Ergebnis wird *true*, bei falschen *false* zurückgeben.

Die Utils-Klasse

Utils.java enthält eine Funktion *Pow(...)*, welche Potenzen berechnen kann. Als Eingabe erhält sie die Basis und den Exponent, zurückgegeben wird der Potenzwert. Die Funktionen *RoundD(...)* und *RoundI(...)* runden den Eingabewert auf einen Double-Wert. Dies ist sehr hilfreich für die Generierung der Aufgaben.

Aufgabenerstellung

Die Low-Level-API

GenericTaskGenerator.java erzeugt die Aufgaben. Zuerst wird zuerst der Zufallsgenerator initialisiert. Dies ist notwendig, da die Aufgaben zufällig erzeugt werden soll, also muss der Zufall der sämtliche Zahlenwerte festlegen. Der *PostTaskString* „ = ?“ wird deklariert um später ans Ende von jedem Aufgabentext gehen zu werden. Es gibt drei Gruppen von Aufgabentypen, Ganzzahlaufgaben, Kommataufgaben und Aufgaben höherer Rechenoperationen. Dieser erzeugt Aufgaben, welche im Zahlenbereich der Ganzen Zahlen liegen, wo es als Operatoren nur +, -, *, / gibt. Die Kommataufgaben werden genauso generiert, nur das die Zahlen im Bereich der reellen Zahlen liegen. Bei sämtlichen Double-Aufgaben werden Zahlen mit genau einer Nachkomastelle erzeugt, sonst wird der Schwierigkeitsgrad zu hoch. Die höheren Grundrechenarten liefert Wurzel-, Potenz- und Logarithmusaufgaben. Alle diese Aufgaben des Höheren Bereiches haben Ergebnisse im Bereich der natürlichen Zahlen, den es soll beim Kopfrechnen bleiben.

Weiterhin ist zu sagen, das bei beiden Subtraktionsaufgaben festgelegt werden kann ob das Ergebnis negativ sein kann oder nicht. Bei den Divisionsaufgaben kann festgelegt werden ob der Quotient natürlich sein soll. Ebenfalls ist bestimmbar in welcher Spanne sich die Nachkommastellen befinden. Also ob nur Ergebnisse mit maximal 0,5 oder 0,25 (0,75) als Nachkommastelle Auftauchen dürfen.

Die High-Level-API

TaskGenerator.java besteht aus einer großen Funktion *nextTask(...)*, welche als Eingabewert nur die Klassenstufe erhält. Die Funktion unterteilt sich dann in 3 Teile, für Klassenstufe 5 und 6, 7 und 8 und 9 und 10. Es wird für die ausgewählte Klassenstufe ein Zufallswert erzeugt. Es gibt so viele Fälle, wie es verschiedene Zufallswerte geben kann. Für jeden Fall wird ein bestimmter Aufgabentyp erzeugt, manche Aufgabentypen werden auch in 2 oder 3 Fällen erzeugt, so dass diese Aufgaben mit größerer Wahrscheinlichkeit im Spiel vorkommen. In den fünften und sechsten Klassen ist es wahrscheinlicher,

dass Ganzzahladditionen und Ganzzahlsubtraktionen auftreten. Double- Subtraktion, Multiplikation und Divisionsaufgaben müssen bei diesem Schwierigkeitsgrad noch nicht bearbeitet werden.

Bei Klasse 7 und 8, und ebenso bei 9 und 10, liegt der Häufungspunkt bei Double- Subtraktion, Multiplikation und Division. Bei 9 und 10 sind es zusätzlich noch die Potenz- und Logarithmusaufgaben.

Hier nun eine Tabelle mit allen möglichen Aufgabentypen:

Name	Funktion
IntAdditionTask(...)	Erzeugt eine Additionsaufgabe im Zahlenbereich der natürlichen Zahlen
IntSubstractionTask(...)	Erzeugt eine Subtraktionsaufgabe im Bereich der natürlichen Zahlen
IntMultiplicationTask(...)	Erzeugt eine Multiplikationsaufgabe im Bereich der natürlichen Zahlen
IntDivisionTask(...)	Erzeugt eine Divisionsaufgabe im Bereich der natürlichen Zahlen
DoubleAdditionTask(...)	Erzeugt eine Additionsaufgabe im Bereich der Reellen Zahlen
DoubleSubstractionTask(...)	Erzeugt eine Subtraktionsaufgabe im Bereich der reellen Zahlen
DoubleMultiplicationTask(...)	Erzeugt eine Multiplikationsaufgabe im Bereich der reellen Zahlen
DoubleDivisionTask(...)	Erzeugt eine Divisionsaufgabe im Bereich der reellen Zahlen
SquareTask(...)	Erzeugt eine Potenzaufgabe im Bereich der natürlichen Zahlen
RootTask(...)	Erzeugt eine Wurzelaufgabe im Bereich der natürlichen Zahlen
LogartihmTask(...)	Erzeugt eine Logarithmusaufgabe im Bereich der natürlichen Zahlen

Datenspeicherung

Daten dauerhaft zu speichern ist auf dem Handy (unter J2ME) bedeutend komplizierter als auf einem Desktop PC (J2SE). Das Handy besitzt zwar ein Dateisystem, auf dieses hat man über die Java-Laufzeitumgebung jedoch keinen Zugriff. Zur dauerhaften Datenspeicherung muss deshalb auf das Record-Management-System (im folgenden RMS) zurückgegriffen werden.

Die Klassen des RMS liegen im Namespace `javax.microedition.rms`. Daten die auf diese Weise gespeichert werden bleiben auch über Ein- und Ausschalten des Handys und sogar dem Herausnehmen des Akkus erhalten.

Das RMS arbeitet nach dem Vorbild einer satzorientierten Datenbank. Die Datenbank, die als Record-Store bezeichnet wird, enthält Datensätze, die durch einen eindeutigen Identifikator gekennzeichnet sind.

Im MatheFreak Mobile Midlet werden zwei Dinge dauerhaft gespeichert. Das sind zum Einen die Einstellungen des Benutzers, die von der Settings-Klasse verwaltet werden, und der Highscore, für den die Klasse Highscore erstellt wurde. Beide sollen im Folgenden näher vorgestellt werden.

Wichtige Klassen des RMS

Der Zugriff auf die verschiedenen Record-Stores findet über die Klassen des Namespaces `javax.microedition.rms` statt. Am wichtigsten ist die Klasse `RecordStore`. Sie stellt neben Methoden zum Öffnen von Record-Stores auch Methoden zum Speichern von Datensätzen oder zum Löschen von Record-Stores bereit.

Weiterhin ist die Klasse `RecordEnumeration` zu erwähnen. Mit ihr ist es deutlich einfacher durch alle Datensätze des Record-Stores zu iterieren. Diese Funktionalität ist vor allem in der Klasse Highscore nötig.

Die Settings-Klasse

Die Klasse Settings speichert die Einstellungen des Benutzers sowie einige programmspezifische Optionen. Die gespeicherten Daten sind in der folgenden Tabelle aufgelistet:

Name	Beschreibung
FirstRun	Zur Feststellung ob erster Programmstart erfolgt.
SnakeSize	Größe der Snake Felder
SnakeControl	Steuerung des Schlange im Snakespiel
StandardClass	Standardmäßig ausgewählte Klassenstufe
Version	Version des Midlets. Für eventuelle Updates
PlayerName	Name des Spielers. Für Highscore und Multiplayermodus
EnableSounds	Sounds an/aus

Da im Programm häufig auf die Einstellungen zugegriffen werden muss, wird die Settings-Klasse als öffentliches Feld in der Klasse `MatheFreakMidlet` abgelegt. Da das Midlet an jedes Formular übergeben wird, hat jedes Formular Zugriff auf die Einstellungen.

Laden

Die `Load`-Methode ist statisch und gibt ein Settings-Objekt zurück. Sie lädt zuerst den Record-Store mit dem Namen *MatheFreak Settings* und erstellt diesen notfalls.

Danach wird überprüft, ob bereits ein Datensatz im Record-Store vorhanden ist. Ist dies nicht der Fall werden die Standardeinstellungen geladen. Andernfalls wird der Datensatz gelesen und der Inhalt durch die Methode `FromByteArray` in die Settings-Klasse eingefügt.

Speichern

Beim Speichern wird wie beim Laden zuerst der Settings-Record-Store geladen. Wenn in diesem bereits ein Datensatz enthalten ist, wird er mit den aktuellen Einstellungen überschrieben, sonst wird ein neuer Datensatz angelegt.

Die Highscore-Klasse

Der Highscore speichert die zehn besten Ergebnisse aus dem Training. Aufgrund dessen muss im Highscore auch eine Liste von Einträgen verwaltet werden, weshalb die Highscore-Klasse deutlich umfangreicher ist als die Settings-Klasse.

Die Einträge im Highscore sind Objekte vom Typ `HighscoreItem`. Dies ist eine selbst erstellte Klasse, die Informationen über Name, Punktzahl und die berechnete Note enthält. Außerdem besitzt sie die Methoden `FromByteArray`, die ein `HighscoreItem`-Objekt aus einem byte-Array erstellt, sowie die Methode `ToByteArray`, die den `HighscoreItem` in einen byte-Array konvertiert. Beide sind für die spätere Speicherung in RMS notwendig.

In der Highscore-Klasse werden die Elemente mit Hilfe eines Vector verwaltet. Damit ist es leichter Elemente an bestimmten Stellen einzufügen oder zu löschen.

Methodenübersicht

Die öffentlichen Methoden der Highscore-Klasse sind im Folgenden aufgelistet. Die wichtigsten werden außerdem erläutert.

Name	Beschreibung
<code>IsEmpty()</code>	Sind Einträge vorhanden?
<code>ItemCount()</code>	Anzahl der Einträge
<code>GetHighscoreItem(...)</code>	Element an bestimmter Stelle auswählen
<code>GetHighscore()</code>	Alle Elemente auswählen
<code>IsInHighscore(...)</code>	Wäre die angegebene Punktzahl im Highscore
<code>Insert(...)</code>	Fügt einen Eintrag in den Highscore ein
<code>Load()</code>	Lädt den Highscore
<code>Save()</code>	Speichert den Highscore
<code>Delete()</code>	Löscht den Record-Store des Highscores

Einfügen

Die `Insert`-Methode bekommt als Parameter ein `HighscoreItem`-Objekt übergeben, welches dann in den Highscore eingefügt wird.

Um dabei die richtige Stelle zum Einfügen zu finden, werden alle Elemente des Highscore, vom besten bis zum schlechtesten, durchlaufen. Dabei wird in jedem Durchlauf das entsprechende Element mit der `GetHighscoreItem(...)`-Methode geladen. Ist dieses Element *null*, was aufgrund des Schleifenaufbaus auf jeden Fall am Ende eintritt, wird das Element an der aktuellen Position eingefügt. Ist das Element hingegen nicht *null*, so wird geprüft, ob die Punktzahl des einzufügenden

Elements größer als die des aktuellen Elements ist. Ist das der Fall, wird das einzufügende Element an der aktuellen Stelle eingefügt, ansonsten in den nächsten Schleifendurchlauf gesprungen.

Nach dem Einfügen eines Elements wird außerdem die nicht öffentliche Methode `Resize()` aufgerufen. Durch sie wird die Größe des Highscore auf zehn Elemente begrenzt.

Laden

Beim Laden des Highscore wird zuerst der Record-Store *MatheFreak Highscore* geladen. Außerdem wird zusätzlich zum RecordStore-Objekt ein RecordEnumeration-Objekt erstellt. Mit diesen wird durch alle vorhandenen Datensätze im Record-Store iteriert und jeder Eintrag durch die `Insert(...)`-Methode in den Highscore eingefügt.

Speichern

Beim Speichern wird ebenfalls der Record-Store *MatheFreak Highscore* geladen und die entsprechende RecordEnumeration erstellt.

Danach werden alle Elemente des Highscore durchlaufen, die vorher mit der Methode `GetHighscore()` geladen wurden. Für jedes einzelne wird geprüft, ob ein weiterer Datensatz im Record-Store vorhanden ist. Ist dies der Fall, wird der Eintrag überschrieben. Sonst wird ein neuer Eintrag angelegt. Zum Umwandeln der HighscoreItems wird die Methode `ToByteArray()` der HighscoreItem-Klasse genutzt.

Oberflächenerstellung

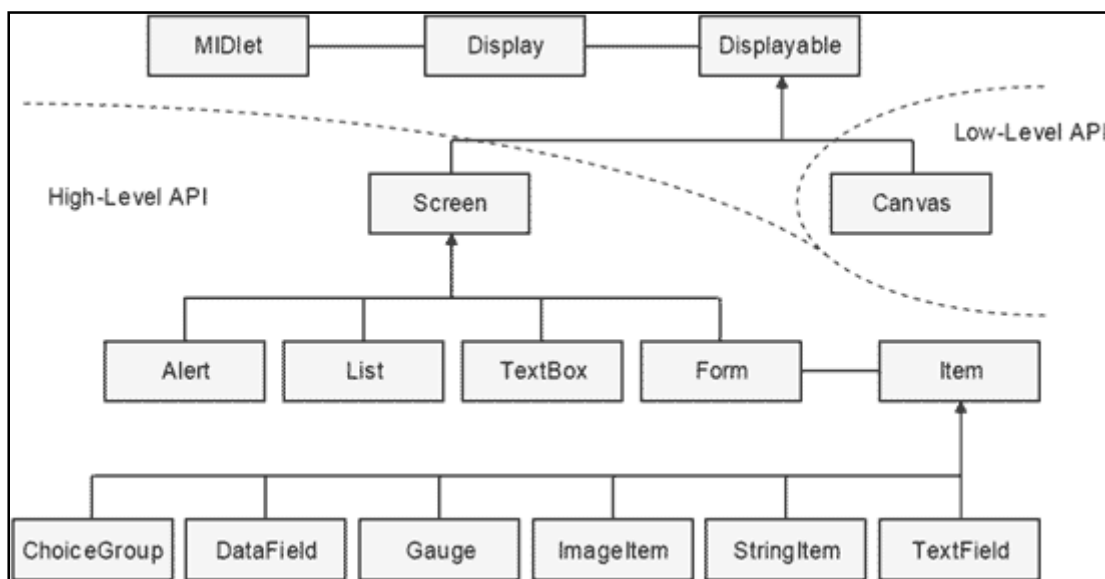
Praktisch alle der heutzutage neu verkauften Handys sind in der Lage, Java Programme („Midlets“) auszuführen. Dies ist der J2ME zu verdanken, der Java 2 Micro Edition. Sie ist eine abgespeckte bzw. speziell an mobile Endgeräte angepasste Version der Programmiersprache Java von Sun Microsystems. Die Grundlage bilden dabei die Konfigurationen und Profile. Die Konfigurationen stellen verschiedene Bibliotheken und eine Virtuelle Maschine zur Verfügung. Die einzige wirklich verbreitete Konfiguration nennt sich CLDC, Connected Limited Device Configuration, und ist auf jedem Java-fähigen Mobiltelefon implementiert. Profile sind die APIs, die es zu einer Konfiguration gibt. So existiert vor allem für Mobiltelefone das Mobile Information Device Profile (MIDP). Es erweitert die CLDC um Klassen für Netzwerkfunktionen, Benutzungsoberflächen und Klassen, die ein lokales Speichern von Daten, die nach Beenden der Applikation erhalten bleiben, ermöglichen.

Midlets

In einem Midlet kann man via folgender Zeile etwas auf dem Handydisplay anzeigen:

```
Display.getDisplay(MIDlet).setCurrent(Displayable);
```

Midlet bedeutet dabei einen Verweis auf die Hauptklasse des Projekts, welche auch ausgeführt wird sobald man das Spiel startet. Das Midlet muss also immer mit übergeben werden, damit man aus einem beliebigen Dialog auch etwas anderes Anzeigen kann.



Die Displayables sind Objekte, die auf dem Handydisplay dargestellt werden können. Man kann sie in eine Low- und eine High-Level API unterteilen. Die **Low-Level API** erlaubt es, direkt einzelne Pixel des Bildschirms anzusprechen. Der Programmierer hat hier volle Gestaltungsfreiheit. Dazu gehören die Klassen Canvas und seit MIDP 2.0 auch GameCanvas, welches in unserem Projekt Anwendung gefunden hat. So sind der Startbildschirm und das Snake-Spiel von GameCanvas abgeleitet. Man kann bei diesen Canvas-Objekten wie von J2SE gewohnt ein Graphics-Objekt erstellen und mit diesem dann beliebige Zeichenfunktionen durchführen:

```
//this -> von GameCanvas abgeleitete Klasse
```

```
Graphics g = this.getGraphics();
```

```
g.drawRect(0,0,10,10);

//Backbuffer wird zu Frontbuffer

this.flushGraphics();
```

Da GameCanvas aber speziell für Spiele ausgelegt ist, arbeitet es mit einem BackBuffer. Das heißt, die Zeichenoperationen werden nicht sofort sichtbar, sondern man muss erst, nachdem man mit Zeichnen fertig ist, Front- und BackBuffer austauschen, wie oben in der 5. Zeile zu sehen ist.

Oberflächenelemente

Bei der **High-Level API** dagegen kann man das Design nicht beeinflussen. Dies wird hierbei völlig der Implementation des Handyherstellers überlassen, um so eine gute Darstellung auf allen Geräten zu gewährleisten. Dies hat natürlich auch den Vorteil, dass man sich nicht um verschiedene Displaygrößen o.ä. kümmern muss, wie es beim GameCanvas der Fall ist. In obiger Grafik sieht man die 4 Objekte der High-Level API, welche auch alle bis auf die TextBox in MatheFreak Mobile genutzt werden. Diese Objekte können direkt via *display.setCurrent()* angezeigt werden.

Ein **Alert** ist dabei eine Art Meldefenster. In diesem kann ein Text und ein Bild eingebunden bzw. ein Typ der Meldung spezifiziert werden. Der User kann dabei nur „OK“ drücken.

List stellt eine Liste dar. Man kann beliebig viele Einträge hinzufügen und selber auf Eingaben des Users reagieren. Zu jedem Eintrag kann auch noch ein Bild zugewiesen werden. Eine Liste unterstützt 3 verschiedene Arten. *Choice.EXCLUSIVE* stellt die Listeneinträge als Radiobuttons dar, also als eine Auswahl, bei der nur ein Eintrag aktiviert sein kann. *Choice.MULTIPLE* dagegen zeigt die Listeneinträge als Checkboxes an, es ist also eine Mehrfachauswahl möglich. *Choice.IMPLICIT* schließlich zeigt überhaupt nichts an außer dem Eintragtext. Bei dieser Liste ist nur der Eintrag selektiert, der gerade markiert ist. Dies bietet sich für Menüs jeglicher Art an und wurde auch im Haupt- und Mehrspielermenü eingesetzt.

Am häufigsten benötigt wurde jedoch das **Form**. Auf dieses können per *Form.append(Item)* diverse Objekte wie Texte, Eingabefelder, Ladebalken und Bilder hinzugefügt und damit auch angezeigt werden. Der Einstellungs-Dialog (*SettingsForm.java*) ist z. B: ein recht komplexes Form. Auf ihm befinden sich mehrere Eingabefelder (TextField) und eine Auswahlmöglichkeit (ChoiceGroup). Dabei bestimmen wir als Programmierer lediglich die Reihenfolge, in der wir die Elemente zum Form hinzufügen. Die Anordnung übernimmt das Mobiltelefon selbst.

Befehlsbehandlung

Abschließend soll noch erwähnt werden, wie man auf Eingaben des Users reagieren kann. Dazu weist man dem Form oder der Liste einen Command-Handler zu. Dieser muss jedoch erst implementiert werden:

```
public class Test extends Form implements CommandListener {..}
```

Zunächst muss man, wie in dieser Zeile ersichtlich in den Kopf der Klasse *implements CommandListener* hinzufügen. Dadurch erscheint dann auch automatisch in der Klasse eine neue Funktion (zumindest in Netbeans):

```
public void commandAction(Command com, Displayable disp) {...}
```

Nun muss dem Form noch mitgeteilt werden, wo der CommandListener zu finden ist:

```
this.setCommandListener(this);
```

Jetzt können Befehle hinzugefügt werden. Diese werden dann vom Mobiltelefon auf die beiden Softkeys verteilt, bzw. in einem Menü angeordnet wenn es mehr als 2 Befehle werden:

```
Command cmdGO = new Command(„Starten“, Command.OK, 1);
```

```
this.addCommand(cmdGO);
```

Der erste Parameter des Konstruktors enthält den Namen des Befehls, der dann (auf gewöhnlichen Handys) links oder rechts unten angezeigt wird. Der zweite Parameter spezifiziert die Art des Befehls, so dass das Handy diesem Befehl selbständig eine bestimmte Taste zuweisen kann. So wird z. B. Command.BACK meistens auf die Roter-Hörer-Taste gelegt. Der dritte und letzte Parameter gibt die Priorität an. Je höher, desto weiter oben steht der Befehl in der Liste aller Befehle, falls es mehr Befehle als Softkeys gibt (und ein Menü angelegt wird).

Alle Details zu den jeweils verwendeten Displayables, Items, ... lassen sich auch aus den Kommentaren entnehmen. Dies sollte nur einen groben Überblick über die Funktionsweise der Java Platform for Micro Devices geben.

Bluetooth

Gleich zu Beginn unserer Arbeit kam uns die Idee, einen Mehrspielermodus in das Spiel zu integrieren. Ein Großteil der in den letzten ein bis zwei Jahren verkauften Mobiltelefone unterstützen einen Funkstandard namens Bluetooth. Dieser ist durch einen geringen Stromverbrauch, geringe Reichweite und leider ebenfalls recht kleine Durchsatzraten perfekt auf mobile Einsatzzwecke abgestimmt. Neben Bluetooth ist auf praktisch allen modernen Handys auch eine Java Variante (J2ME) installiert. Einige Handyhersteller haben daraufhin eigene Bluetooth-API's entwickelt. Damit man aber plattformübergreifend auf die Bluetooth-Schnittstelle des Handys zugreifen kann, setzten sich im Oktober 2000 Motorola, Erisson, IBM, Sun und noch viele andere zusammen und schufen die JSR-82 (Java Specification Request), eine standardisierte Bluetooth-API.

Initialisierung des Stacks

Jedes JSR-82 unterstützende Mobiltelefon besitzt einen Bluetooth Stack. Dies ist die Software/Firmwarekomponente, die direkten Zugriff auf die Hardware hat. Bevor wir überhaupt über Server und Clients nachdenken, muss zunächst der Bluetooth Stack initialisiert werden, d.h. Bluetooth wird aktiviert und das Gerät wird „sichtbar“ gemacht:

```
localDevice = LocalDevice.getLocalDevice();  
  
localDevice.setDiscoverable(DiscoveryAgent.GIAC);  
  
discoveryAgent = localDevice.getDiscoveryAgent();
```

In der 3. Zeile wird ein „Entdeckungs-Agent“ Objekt erstellt. Wie der Name bereits sagt, kann nun mittels diesem Objekt ein anderes Handy gesucht (und hoffentlich entdeckt) werden. Es ist allerdings nicht für die Initialisierung notwendig. Die soeben genannte Gerätesuche wird gleich näher erläutert, doch bevor man einen Dienst suchen kann, muss natürlich ein Server existieren.

Server

Um einen Server zu starten, benötigt man (genau wie für das Verbinden mit einem Server) nur eine Verbindungs-URL. Diese könnte folgendermaßen aussehen:

```
String url = "bt12cap://localhost:" + uuid + ";name=MatheFreak  
Server;master=false;authenticate=false;authorize=false;encrypt=false  
;ReceiveMTU=48;TransmitMTU=48";
```

bt12cap ist dabei das Protokoll, über das die Verbindung abläuft (Logical Link Control and Adaption Layer Protocol). Wir haben L2CAP gewählt, da bei diesem die Paketgröße direkt festgelegt werden kann und so ein extra Parser für empfangene Pakete nicht benötigt wird. Mit localhost wird definiert, dass diese Verbindungs-URL einen Server starten soll. Die ‚uuid‘ ist eine eindeutige, einzigartige, 32 Bit lange Kennnummer (UUID), anhand welcher der Dienst identifiziert werden kann. Nach diesen notwendigen Angaben können noch beliebig viele Parameter angehängt werden. So wird hier z. B. der Name spezifiziert, weiterhin wird angegeben, dass es dem Server egal ist, ob er eine Master oder Slave-Rolle spielt und das weder eine Authentifizierung, Autorisierung oder Verschlüsselung vorgenommen werden soll. Die letzten beiden Parameter geben die MTU für aus- und eingehende Pakete an. Die **Maximum Transmission Unit** beschreibt die maximale Paketgröße, die über ein Netzwerk übertragen werden kann. Dies bedeutet hier speziell, dass in jedem Paket maximal 48 Byte

übertragen werden können, womit sich aber alle für MatheFreak benötigten Übertragungen gut durchführen lassen.

Gestartet wird der Server nun durch folgenden Code:

```
L2CAPConnectionNotifier server =  
(L2CAPConnectionNotifier)Connector.open(url);
```

Durch diese Zeile wird unser Server in die Service Discovery Database (SDDB), also die Liste der vom Handy bereitgestellten Dienste, eingetragen. Er kann nun von anderen Geräten entdeckt werden.

Nun müssen allerdings noch die eingehenden Verbindungen gehandhabt werden. Durch die folgende Zeile wird der aktuelle Thread so lange blockiert, bis sich ein Client verbindet. Die Verbindung zu diesem Client ist dann in der Variable connection gespeichert:

```
L2CAPConnection connection = server.acceptAndOpen();
```

Nun besteht eine Verbindung zwischen unserem Server und einem Client. Es können jetzt Daten gesendet und empfangen werden:

```
//Bytes senden  
connection.send(Byte[]);  
  
//Warten bis Client etwas sendet  
while (!connection.ready())  
{try {Thread.sleep(50);} catch (InterruptedException ex) {}}  
  
//Paket vom Client empfangen  
  
//(m Bytes werden im Array data gespeichert)  
int m = connection.receive( data);
```

Jetzt, wo unser Server läuft, bleibt noch offen wie man sich mit ihm verbinden kann. Dazu muss zunächst eine Gerätesuche durchgeführt werden.

Gerätesuche

Die Suche wird gestartet mit folgender Zeile:

```
discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this);
```

Der erste Parameter bedeutet dabei, dass nach allen derzeit erreichbaren Geräten gesucht werden soll und der zweite Parameter gibt an, dass der discoveryListener in dieser gleichen Klasse zu suchen ist. Dieser muss allerdings zunächst implementiert werden. Dazu schreibt man in den Kopf der Klasse *implements DiscoveryListener* so wie in diesem Beispiel:

```
public class BtClient implements DiscoveryListener
```

Weiterhin müssen dann noch 4 Funktionen in dieser Klasse deklariert werden.

```
deviceDiscovered()
```

```
inquiryCompleted()
```

```
servicesDiscovered()
```

```
serviceSearchCompleted()
```

Im Moment interessieren uns jedoch nur die ersten beiden. Nachdem die Suche nun mit *startInquiry()* gestartet wurde, wird im Hintergrund nach anderen Geräten gesucht. Sobald etwas gefunden wird, wird die Funktion *deviceDiscovered()* aufgerufen. In dieser kann nun das gefundene Gerät beispielsweise in einem Vektor gespeichert oder für den User ausgegeben werden. Sobald die Gerätesuche abgeschlossen ist, wird *inquiryCompleted()* aufgerufen und man kann so nun z. B. eine Geräteauswahl darstellen.

Im Normalfall bietet jedes Handy zahlreiche Dienste (Services) an, wie z.B. Datei-Transfer, Serielle Verbindung, Headset usw. . Um aus diesen vielen Diensten eines Geräts den richtigen auszuwählen, muss eine weitere Suche gestartet werden. Diese funktioniert ähnlich der Gerätesuche. Gestartet wird sie mit:

```
discoveryAgent.searchServices(attrSet, uuidSet, remoteDevice, this);
```

Über den ersten Parameter kann man bestimmen, welche Attribute der gefundenen Dienste übertragen werden sollen. Üblich wäre hier z.B. 0x0100, wodurch der Name des betreffenden Dienstes mit übertragen wird. Der 2. Parameter bestimmt, wie genau die Suche ist. Wie bereits erwähnt, besitzt jeder Dienst eine eindeutige, einzigartige UUID. Weiterhin gibt es grobe uuid's, welche z.B. alle Dienste beeinhaltet, die auf einem bestimmten Protokoll aufbauen. Über diese uuid kann also die Suche eingeschränkt werden. Als 3. Parameter muss ein im ersten Schritt gefundenes Gerät übergeben werden, auf welchem dann nach den Diensten gesucht werden soll. Als letztes wird wieder angegeben, wo der DiscoveryListener zu finden ist. Erinnern Sie sich noch an die 2 bisher nicht benötigten Funktionen? *ServicesDiscovered()* und *ServiceSearchCompleted()* funktionieren analog zu *DeviceDiscovered()* etc. Sobald ein Dienst gefunden wird, erfolgt ein Aufruf der Funktion *servicesDiscovered()*, wo dann die gefundenen Dienste in einem Array oder Vektor gespeichert werden müssen. Wenn die Suche abgeschlossen ist wird *ServiceSearchCompleted()* aufgerufen. Nun kann aus den erforschten Diensten der betreffende ausgewählt und die Verbindungs-URL extrahiert werden:

```
String URL =
ServiceRecord.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT,
false);
```

Diese URL ist alles was man benötigt, um sich mit dem Server/Dienst zu verbinden. Der 1. Parameter gibt dabei die benötigte Sicherheit der Verbindung an, der zweite ob es dem Gerät egal ist ob es Master oder Slave „spielt“.

Client

Nach Kenntnis der Verbindungs-URL, wollen wir natürlich auch eine Verbindung zu dem Server aufbauen. Dies funktioniert ähnlich wie das Starten eines Servers.

```
L2CAPConnection connection =(L2CAPConnection)Connector.open(URL);
```

Nun können, genau wie bei dem Server, Bytes mit `connection.send()` gesendet und mit `connection.receive()` empfangen werden. Um sich wieder vom Server zu trennen, muss folgender Befehl aufgerufen werden:

```
connection.close();
```

Gleiches gilt für den Server, falls er die Verbindung zu einem Client trennen will. Soll der Server komplett beendet und der Eintrag des Dienstes aus der SDDB entfernt werden, so muss zusätzlich noch folgendes ausgeführt werden:

```
server.close();
```

Konkrete Anwendung

Mithilfe der soeben erläuterten Bluetooth-API haben wir 3 verschiedene Funktionen in MatheFreak Mobile implementiert.

TrainingClient/Server enthalten eine Multiplayer-Variante des Kopfrechentrainings. Der Server generiert 10 Aufgaben, schickt diese an den Client und beide rechnen die Aufgaben aus. Sobald einer fertig ist, wird das Spiel beendet und der Sieger angezeigt.

Die *ScoreSynch* Klassen enthalten eine Highscore-Synchronization. Zwei MatheFreak Mobile Spieler mit Bluetooth-Handys können über diesen Modus ihre Highscores Synchronisieren.

Als kleinen Bonus haben wir auch noch einen simplen *Chat* programmiert. Allerdings funktioniert dieser nur zwischen 2 Personen.

BTDiscover.java enthält die komplette Gerätesuche.

Im Nachfolgenden soll nun kurz erklärt werden, wie diese einzelnen Klassen funktionieren.

BTDiscover

Die Gerätesuche verläuft praktisch exakt wie oben beschrieben. Zunächst wird Bluetooth initialisiert. Danach wird eine Suche nach Geräten gestartet. Zunächst werden nur die Hardware-Adressen der Geräte angezeigt. Erst wenn die Suche abgeschlossen ist, werden die Namen aufgelöst. Der User bekommt nun eine Liste, aus welcher er das Gerät, mit dem er sich verbinden möchte, auswählen muss. Nun wird eine Serversuche (Dienstsuche) auf dem ausgewählten Gerät gestartet. Normalerweise könnte man nun alle Dienste ausgeben und den Benutzer den richtigen Dienst selektieren lassen. Wir haben aber diesen Schritt noch weiter vereinfacht. Gesucht wird nur nach einer ganz speziellen uuid. Diese ist bei allen 3 Servern (Training, ScoreSynch, Chat) gleich. Die Server unterscheiden sich nur in ihrem Namen. Es wird also nach dieser uuid gesucht, und nachdem die Suche abgeschlossen ist, wird der erste gefundene Dienst genommen und Verbindungs-URL sowie Name des Dienstes werden gespeichert. Dies kann so gemacht werden, weil bei der Suche nach der speziellen ID sowieso nur MatheFreak Server aufgelistet werden und auf einem Gerät nicht 2 MatheFreak Server gleichzeitig laufen sollten (auf eigentlich allen Mobiltelefonen auch gar nicht möglich). Nun wird je nach Name des Dienstes (z. B. „ScoreSynch Server“) die entsprechende Client-Klasse aufgerufen (*ScoreSynchClient.java*). Diese bekommt im Konstruktor auch gleich noch die Verbindungs-URL übergeben.

Die Klassen bauen dann jeweils wie oben unter Client beschrieben eine Verbindung mit dem Server auf. Ab diesem Punkt unterscheiden sie sich allerdings.

Multiplayer-Training

Das Mehrspieler-Training ähnelt stark dem Einzelspielermodus. Allerdings wird nicht nach jeder Frage eine Neue zufällig generiert. Sobald der Server gestartet wird, generiert er gleich 10 Aufgaben und speichert jeweils Aufgabentext, Punkte und Lösung in einem Array. Jetzt wartet er, bis sich ein Client verbindet. Sobald das passiert, werden alle 10 Aufgaben an den Client gesendet. Dieser empfängt sie und speichert sie genau wie der Server in 3 Arrays. Von diesem Punkt an läuft es sowohl im Server als auch im Client genauso wie im Einzelspielermodus ab. Die beiden Spieler versuchen jeweils einzeln, die gleichen Aufgaben so schnell wie möglich zu lösen. Damit man dabei nicht immer erst den „Eingeben“ Button drücken muss, existiert noch ein weiterer Thread im Hintergrund der jederzeit das Eingabefeld für die Lösung überwacht und zur nächsten Aufgabe springt, wenn die richtige Lösung eingegeben wurde. Ein weiterer Unterschied zum Einzelspielermodus ist, dass nach jeder Frage die nächste Aufgabe eben nicht neu generiert, sondern aus den vorbereiteten Arrays eingelesen wird. Sobald einer der beiden Spieler fertig ist, sendet er seine Punktzahl an den anderen. Dieser bricht nun das Spiel ab und sendet die bisher erreichten Punkte zurück. Auf beiden Handys wird nun der Sieger ermittelt und eine dementsprechende Ausgabe erstellt.

Theoretisch sollte dies perfekt funktionieren, solange nicht beide Spieler innerhalb von 50-100ms (praktisch gleichzeitig) fertig werden. Praktisch (zwischen zwei unserer Benq/Siemens C81 Testgeräten) gab es allerdings schon Probleme, wenn beide Spieler innerhalb einer halben bis einer Sekunde fertig werden. Dagegen kann man aber so gut wie nichts machen, da die Mobiltelefone die Bluetooth-Pakete einfach nicht schneller verarbeiten können.

ScoreSynch

Die Highscore-Synchronisierung funktioniert relativ simpel. Sobald eine Verbindung zwischen zwei Geräten aufgebaut ist, überträgt der Client zunächst die Anzahl seiner Highscore-Einträge und danach die Einträge selber. Der Server empfängt die Einträge und führt zunächst eine zusätzliche Überprüfung durch, ob exakt der gleiche Eintrag schon in der Highscore-Liste steht(Wenn das der Fall ist, soll dieser natürlich am Ende nicht doppelt eingetragen werden). Wenn der Eintrag also noch nicht im Highscore steht, dann wird die *Highscore.Insert()* Funktion aufgerufen. Nachdem der Server nun seinen Highscore und den des Clients zusammengefügt hat, schickt er die Anzahl der Highscore-Einträge und die Einträge an sich an den Client. Dieser löscht seinen alten Highscore und trägt den kompletten Highscore den er zugeschickt bekommt ein. So haben nach der Synchronisation beide Spieler die gleiche Highscore-Liste auf ihren Mobiltelefonen, in der beide vorkommen können.

Chat

Da die Programmierung der Bluetooth-Verbindung ziemlich schwierig war, mussten wir auch sehr viele Tests durchführen, vor allem mit echter Hardware und nicht nur im Emulator. Dazu haben wir zuerst einen sehr einfachen und provisorischen Chat geschrieben, womit wir dann selber die anderen Server testen konnten, indem wir beliebige Strings an sie sendeten. Nachdem wir nun einmal einen solchen „Chat“ programmiert hatten, dachten wir uns, dass wir ihn auch gleich noch als Bonus mit in den MatheFreak packen konnten. Er funktioniert genauso wie die anderen Verbindungen, nur warten Server und Client hier in einem separaten Thread in einer Endlosschleife auf ankommende Pakete, die sie dann direkt ausgeben. Nebenbei kann der User in ein Textfeld selber einen Text eingeben und ihn durch einen Senden Button absenden.

Benutzerschnittstelle

Benutzeroberfläche

Die Benutzeroberfläche ist der Teil der Anwendung, mit der der Benutzer in Berührung kommt. Sie sollte deshalb einfach zu bedienen sein und weitestgehend selbsterklärend aufgebaut. Außerdem soll sie auf vielen Handys dargestellt werden können und auch gleiche Funktionalität unabhängig vom Handy bieten.

Daraus ergeben sich die folgenden Hauptdesignziele für die Benutzeroberfläche:

- Benutzerfreundlichkeit
- Kompatibilität mit viele Handymodellen

Benutzerfreundlichkeit

Um eine hohe Benutzerfreundlichkeit zu bieten nutzen wir verschiedene Ansätze.

Zum Einen sind alle Formulare selbsterklärend, sodass es keiner weiteren Hilfe bedarf. Wenn nötig wurden kleine Hilfetexte direkt integriert. Die Kommandos wurden so gewählt und angeordnet, dass die häufigsten Aufgaben mit wenigen Klicks erreichbar sind.

Weiterhin werden, wenn möglich, kleine Icons genutzt um zum Einen den Wiedererkennungswert der Menüpunkte zu erhöhen und zum Anderen die Funktion des Menueintrags weiter zu unterlegen.

In Länger andauernden Prozessen wird der Benutzer außerdem durch Statusmeldungen auf dem Laufenden gehalten.

Kompatibilität

Um eine weitreichende Kompatibilität der Anwendung zu gewährleisten wird versucht hauptsächlich die Steuerelemente, die direkt in die Java Mobile Edition integriert sind zu nutzen. Da jeder Handyhersteller Java auf das jeweilige Handy anpasst kann so sichergestellt werden, dass die Steuerelemente richtig angezeigt werden.

Programmablauf

Im Folgenden soll der prinzipielle Programmablauf dargelegt werden. Dabei wollen wir auch die Möglichkeiten des Benutzers verdeutlichen und alles mit Screenshots hinterlegen.

Programmstart

Beim Programmstart wird zuerst ein SplashScreen angezeigt. Dieser zeigt auf künstlerische Art und Weise den Inhalt unseres Programms.

Durch das Drücken einer beliebigen Taste wird daraufhin das Hauptmenu geöffnet. Von hier aus kann der Benutzer auf alle Funktionen unserer Anwendung zugreifen.

Das Hauptmenu

Das Hauptmenu ist das am häufigsten verwendete Formular der Anwendung. Von hier aus hat der Benutzer Zugriff auf alle Funktionen. So kann er das Training oder den Multiplayermodus starten, sich den Highscore ansehen. Weiterhin kann er seine Einstellungen anschauen und gegebenenfalls verändern, sich die Hilfe anzeigen lassen oder MatheFreak Mobile beenden.

Der Einzelspielermodus

Im Einzelspielermodus kann das Kopfrechnen trainiert werden. Dazu wählt man zuerst seine Klassenstufe, und damit den Schwierigkeitsgrad, aus. Standardmäßig ist hier die Klassenstufe ausgewählt die in den Einstellungen ausgewählt wurde.

Nachdem die Klassenstufe gewählt wurde beginnt das eigentliche Training. Hier werden dem Benutzer zehn Aufgaben gestellt, die er nacheinander beantworten muss. Nach der

Bestätigung einer Lösung wird dem Benutzer dabei eine Rückmeldung gegeben, ob seine Lösung richtig oder falsch war.

Nachdem alle zehn Aufgaben beantwortet wurden wird geprüft, ob die Punktzahl ausreicht, um in den Highscore zu gelangen. Erreicht man den ersten Platz im Highscore wird hier ein Pokal angezeigt. Schafft man mit der erreichten Punktzahl keinen Eintrag in den Highscore, wird ein trauriger Smiley angezeigt.

Danach wird entweder in den Highscore oder zurück ins Hauptmenu gesprungen.

Der Mehrspielermodus

Alle Funktionen des Mehrspielermodus sind über das

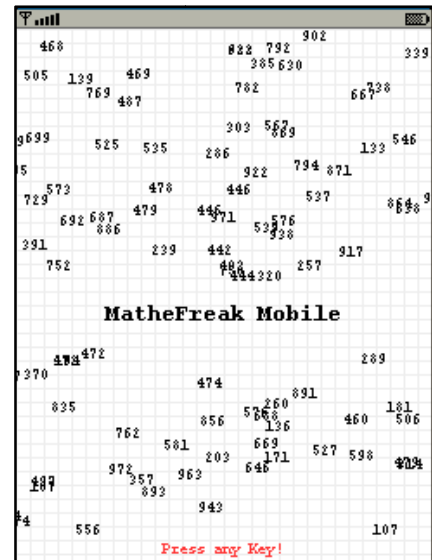


Abbildung 1: Der SplashScreen

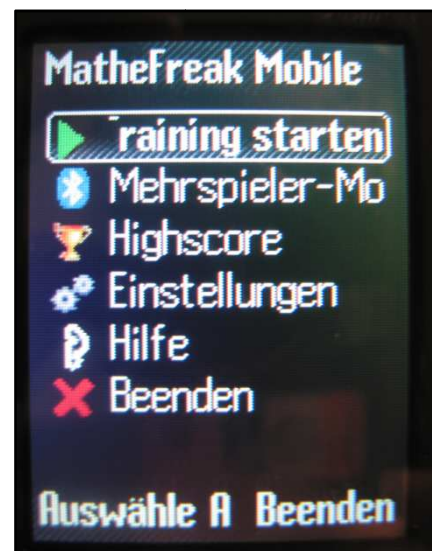


Abbildung 2: Das Hauptmenu

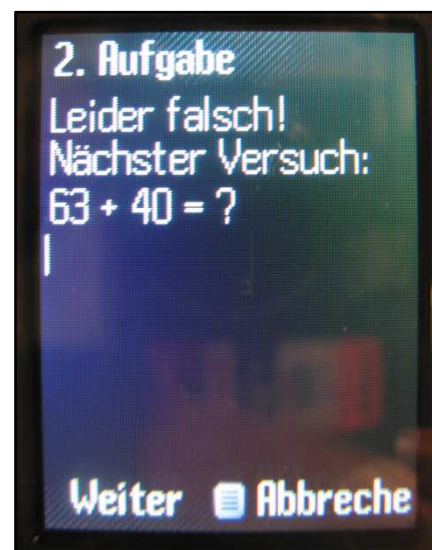


Abbildung 3: Der Einzelspielermodus

Mehrspielermenu erreichbar.

Zur Kommunikation zwischen zwei Handys ist es nötig, dass ein Handy als Server agiert, das andere als Client. Deshalb gibt es im Multiplayermenu Die Möglichkeit einen Server zu starten oder als Client zu agieren und selbstständig einen Server zu finden.

Server

Wenn man sein Handy als Server agieren lassen möchte muss man nur im Multiplayermenu den Eintrag des gewünschten Servers auswählen. Zur Verfügung stehen dabei:

Name	Beschreibung
Trainings Server	Gegen einen Freund spielen.
ScoreSynch Server	Den Highscore mit einem anderen Handy austauschen.
Chat Server	Mit Freund chatten.

Der Server wird durch einfaches Auswählen gestartet und wartet dann auf den entsprechenden Client.

Client

Um als Client zu agieren muss man zuerst den Eintrag *Mit Freund Verbinden* auswählen. Danach sucht das Handy nach anderen erreichbaren Handys. Hierbei wird zuerst eine eindeutige ID angezeigt, nach einer Weile wird aber der richtige Bluetooth-Name des Handys angezeigt. Dann wählt man das Handy aus, auf dem der Server läuft. Der entsprechende Client wird dann automatisch gestartet.

Der Highscore

Im Highscore kann man seine besten Ergebnisse aus dem Training anschauen. Außerdem kann der Highscore im Mehrspielermodus mit Freunden ausgetauscht werden.

Die besten Ergebnisse werden im Highscore durch Pokale gekennzeichnet.

Außerdem gibt es eine Funktion zum Löschen des Highscore. Dadurch werden alle Einträge unwiderruflich gelöscht. Deshalb ist eine doppelte Bestätigung notwendig.

Das Einstellungsformular

Im Einstellungsformular kann der Benutzer seine Einstellungen ändern. Dazu zählen sein Name und seine Klassenstufe, die dann im Training automatisch ausgewählt wird. Weiterhin können die Soundeffekte aktiviert und deaktiviert werden.

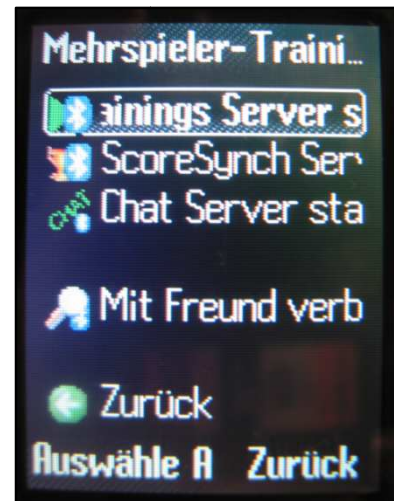


Abbildung 4: Das Multiplayermenu



Abbildung 5: Der Highscore

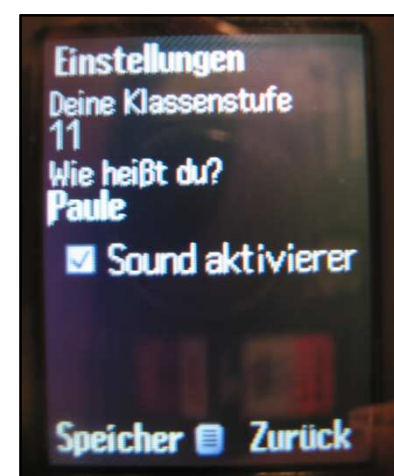


Abbildung 6: Das Einstellungsformular

Wurde das Bonusspiel freigeschaltet gibt es außerdem noch Einstellungen zur Blockgröße von Snake sowie der Art der Steuerung.

Snake – das Bonusspiel

Um die Motivation zu verbessern und Abwechslung zu bieten ist ein kleines Spiel in die Anwendung integriert worden. Dieses muss jedoch erst freigeschaltet werden. Dazu ist es nötig im Training mindestens so viele Punkte zu bekommen wie das doppelte der gewählten Klassenstufe.

Daraufhin erscheint im Hauptmenu ein neuer Eintrag mit dem Namen Snake. Über diesen kann das Spiel gestartet werden.

Ziel ist es die grün-roten Äpfel mit der Schlange zu verzehren. Dadurch steigt zum einen Ihre Punktzahl an, die Schlange wird aber ebenfalls mit jedem gegessenem Apfel länger und ihre Geschwindigkeit steigt.

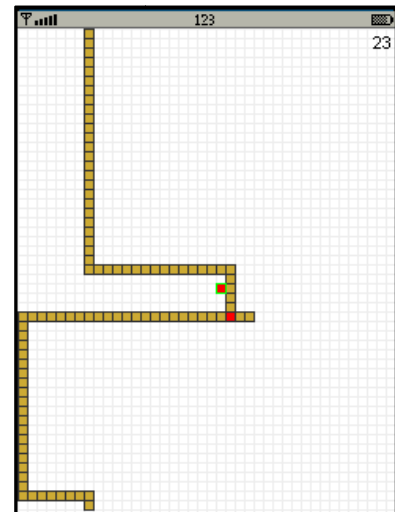


Abbildung 7: Das Bonusspiel Snake

Mit der Freischaltung des Bonusspiels erscheinen auch auf der Einstellungs-Seite neue Optionen. So können Sie die Art der Steuerung und die Größe der Felder im Spiel einstellen.

Referenzen

Sun Microsystems. *developer.sun.com*. Abgerufen am 10. 12 2006 von <http://developer.sun.com>

Schmatz, K.-D. (2007). *Java Micro Edition*. Heidelberg: dpunkt.verlag.

Ullenboom, C. (2005). *Java ist auch eine Insel*. Bonn: Galileo Press.

Anlagen

Quelltexte

Der vollständige Quelltext befindet sich auf der beigelegten CD. Aufgrund der enormen Ausmaße drucken wir nur wesentliche Teile des Quelltextes aus.

Einverständniserklärungen

Die Einverständniserklärungen sind unserer Dokumentation beigelegt.