# Fries4Fish - Pommes für meinen Fisch?

Virtuelle Fischzucht am Handy – Public Screen als Fischaquarium


Projekt-Team:

Claudia Lanzl

Alena Schneider

Andreas Bösselmann

Dieter Hofbauer

Jewgenij Steinhart

betreut von:

Prof. Dr. Gerhard Meixner & Prof. KP Ludwig John


Programmieren von Mobilgeräten

FH-Augsburg, SS 2006

Inhaltsverzeichnis

## 1.1 Ausgangssituation

Nach einigen grundsätzlichen Überlegungen, welcher Bereich des neuen Gebäudes der FH Augsburg für dieses Projekt genutzt wird, kam es zu einer mehrheitlichen Entscheidung der Projektgruppen zugunsten eines „Public Screens“ im Eingangsbereich der neu entstehenden Hochschul-Mensa. Die Fensterfront zur Friedberger Straße hin mit installationsähnlichen Applikationen zu bespielen (wie z. B. das Projekt „Blinkenlights“ des Chaos Computer Club [BL06]) schien nicht nur vom technischen Aufwand für eine Semesterarbeit zu umfangreich und kostspielig, sondern warf auch die Frage nach der Realisierbarkeit in Hinblick auf die Straßenverkehrsordnung auf. Werden die vorbeifahrenden Verkehrsteilnehmer durch die Installation möglicherweise zu sehr abgelenkt?
Durch die Entscheidung für einen Monitor, oder eine Beamerprojektion im Eingangsbereich des Gebäudes war die Basis für tiefergehende Überlegungen und konkrete Projektideen geschaffen. Im weiteren Projektverlauf bildeten sich vier Gruppen, die sich mit folgenden Themengebieten beschäftigen wollten:

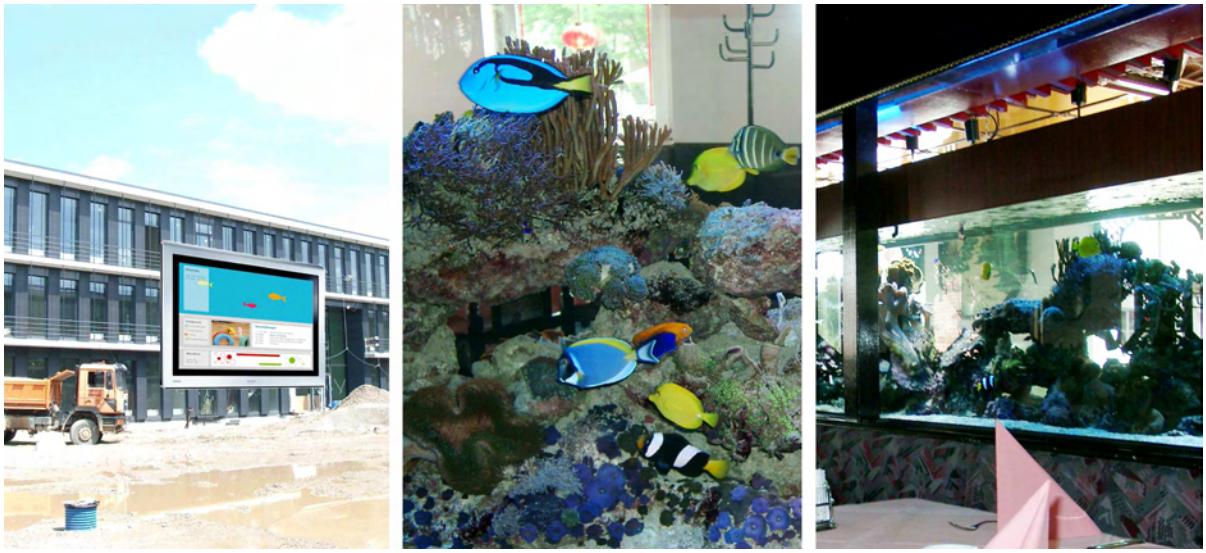
- Gruppe 1: Bereitstellung eines technischen Rahmens
- Gruppe 2: Nutzung der technischen Infrastruktur als Informationsplattform für Studenten
- Gruppe 3: Entwicklung eines Multiplayer-Spiels
- Gruppe 4: Arbeiten mit multimedialen (interaktiven) Inhalten

Diese Dokumentation umfasst die Ideen und die Ausarbeitung der 4. Gruppe.

## 1.2 Entstehung der Idee

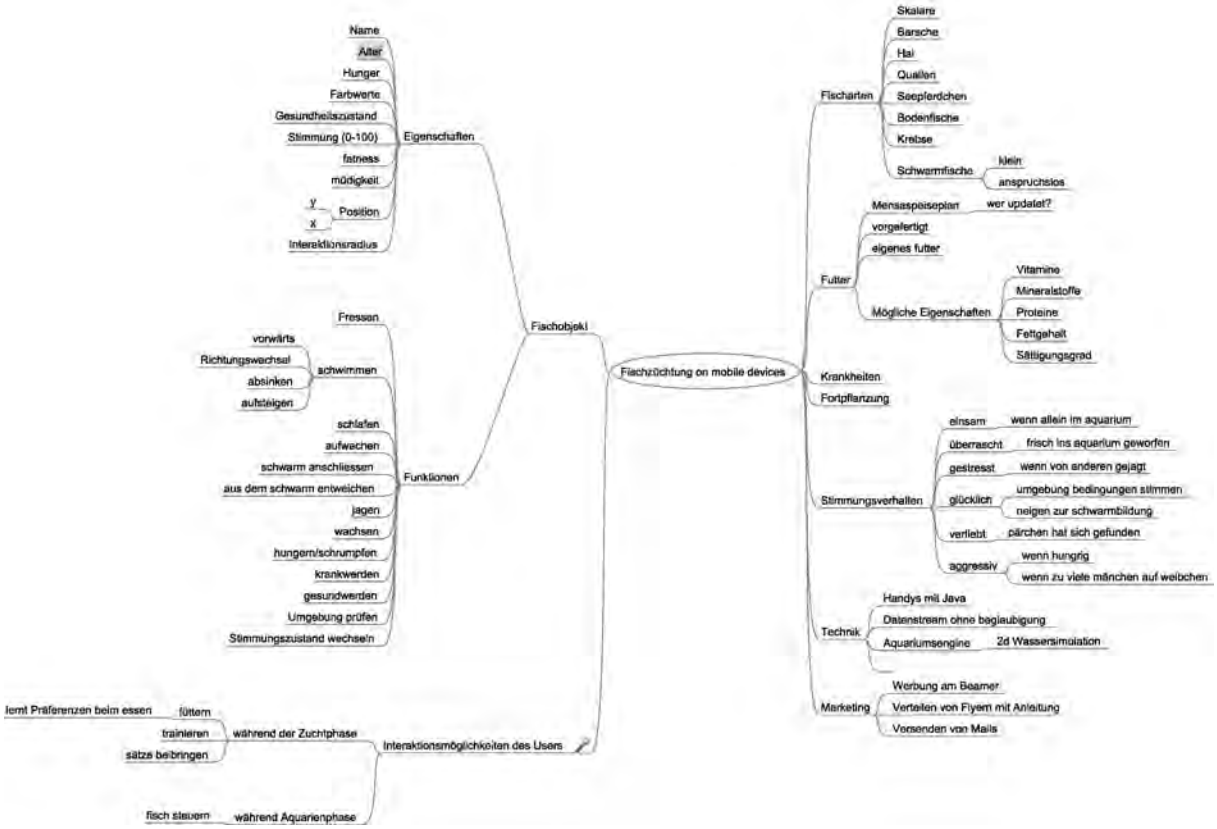
Nach der Aufteilung in oben genannte Projektgruppen fanden sich die einzelnen Mitglieder um die jeweiligen Projektleiter zusammen, um das Team zu bilden.
Durch viele Treffen und Überlegungen wie man den „Public Screen“ als multimediale Plattform nutzen kann, entschied sich die „Multimedia-Gruppe“ dafür, eine unterhaltsame Lösung zu finden.
Wenn man als Student die Mensa betritt möchte man gerne essen, sich von anspruchsvollen Vorlesungen erholen, und mit Kommilitonen austauschen. Knifflige Anleitungen oder aufdringliche Inhalte sollten möglichst vermieden werden, um viele Studenten zu erreichen, und über einen gewissen Zeitraum mit Inhalten zu fesseln. Viele Ideen standen bei dem ersten gemeinsamen Präsentationstermin zur Diskussion, und der Favorit, eine Art virtuelle Fischzucht, fand bei den Professoren und den anderen Gruppen den größten Anklang. Als Hinweis, verworfene Lösungen werden hier nicht mehr diskutiert oder gezeigt, da dies den Rahmen dieser Ausarbeitung sprengen würde. Die Idee, einen überlebensfähigen Fisch am Handy züchten, der sich nach dem Entlassen in ein Server-Aquarium gegen andere Fische

behaupt, entstand beim Essen in einem chinesischen Restaurant. Die Analogie hat sich folgendermaßen ergeben: beim Chinesen beobachtet man gerne die schönen bunten Fisch, die im Aquarium umher schwimmen, und eine beruhigende Wirkung auf viele Gäste haben, während des Essens. Genau diese Stimmung könnte man auch auf einen - möglicherweise gestressten und ausgelaugten - Studenten übertragen. Ihm ein bisschen das Gefühl von Entspannung und Erholung durch Beobachten des Fischaquariums geben. Zudem soll die Möglichkeit angeboten werden, das selbst verzehrte Essen auch an seinen Fisch zu verfüttern, durch Aufnehmen des wöchentlichen Mensa-Essensplans in die Futterliste des Fisches. Das Aquarium an sich ist hier nicht das klassische bekannte Wasserbassin mit realen Pflanzen und Tieren, sondern eine Übertragung dessen ins Virtuelle. Dadurch kommt ein weiterer sehr wichtiger Faktor in die Entwicklung der Idee. Die aktive Beteiligung des Studenten am Aussehen des Aquariums, sowie die Möglichkeit einer Interaktion mit eigenen Fischen.
Die nachfolgende Fotomontage zeigt links den Screen am Fh-Gebäude, rechts das Aquarium im chinesischen Restaurant, und in der Mitte das Herzstück um das sich alles dreht, das Fischaquarium.



Fotomontage: Claudia Lanzl (cc)

Der Gedanke, einen Fisch am eigenen Handy zu erstellen, zu pflegen und nach einer gewissen Zeit in das „Public Screen-Aquarium“ zu entlassen wurde vom Projektteam weiter verfolgt, wobei im ersten Ansatz folgende Mindmap entwickelt wurde.



Mindmap „Fischzucht on mobile devices“ zur Ideenfindung

Nachstehend wird der Ideenstrang der sich herauskristallisiert hat konkretisiert, und das Team konnte mit der Einarbeitung in J2ME, Eclipse, Subversion bzw. Flash und Action Script beginnen.

## 1.3 Spielidee konkreter

Bei „Fries4Fish - Pommes für meinen Fisch?“ wählt sich jeder Teilnehmer / Spieler aus verschiedenen Fischarten seinen "Lieblingsfisch" aus, tauft das Fischbaby (gibt ihm einen Namen), und wählt eine passende Farbe aus. Der Fisch wird dann gespeichert, bekommt ein Alter, seine Lebensenergie sowie einen finanziellen Startbetrag zum Überleben. Nun kann er mit (Mensa-) Essen gefüttert oder auch trainiert werden, wobei beides eine Auswirkung auf den Fisch hat. Er wächst, wird dick oder magert ab, oder Ähnliches. Da der Fisch wie jedes real existierende Lebewesen eine begrenzte Lebenserwartung hat, und nicht nur am Handy existieren soll, muss man sich entscheiden, wann er reif für die große weite Welt - das Aquarium - ist. Ist der eigene Fisch erst mal dort, kann er nicht mehr auf das Mobilfunkgerät zurück, und lebt dort je nach Fischtyp als Einzelgänger, in einem Schwarm, als Raubfisch, usw. Um die soziale Bindung nicht zu verlieren kann der jeweilige Züchter mit seinem Bluetooth-Handy am Aquarium vorbeikommen, um seinen Fisch zu besuchen. So besteht die Möglichkeit der Kontaktaufnahme, und der Fisch kann seinem „Herrchen“ früher (am Handy) erlernte Nachrichten überbringen.

Aus der Vielzahl der Möglichkeiten wurden de nachstehenden Elemente implementiert.

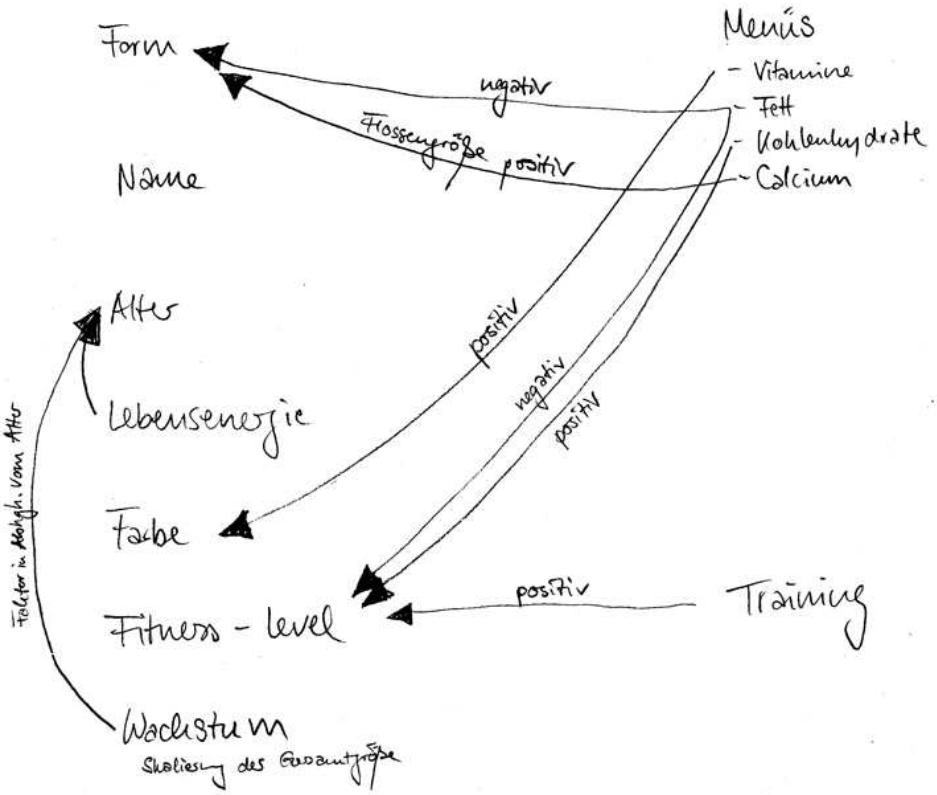
**Features Handy:**

- Fisch auswählen und konfigurieren
- Fisch trainieren (Mini-Spiel)
- Sätze beibringen mit denen der Züchter vom Fisch aus dem Aquarium begrüßt wird
- Fisch mit unterschiedlichen Speisen füttern, die jeweils verschiedene Fett- und Energiegehalte haben

**Features Aquarium:**

- Im Aquarium befinden sich alle Fische, sichtbar ist immer nur eine bestimmte Anzahl, die zufällig wechselt
- Der Züchter kann in Bluetooth-Reichweite mit seinem Fisch kommunizieren, dieser begrüßt in mit gelernten Sätzen
- Fische, die am Handy besser trainiert und ausgewogener gefüttert wurden, können im Aquarium länger überleben

Die möglichen internen Zusammenhänge um die verschiedenen Fischzustände (aktuell implementiert sind diese drei: dünn, normal, dick) zu realisieren sind in der folgenden Skizze aufgeführt.



# 2. Konzept

## 2.1 Anforderungen

### 2.1.1 Zielgruppe

Zielgruppe für „Fries4Fish - Pommes für meinen Fisch?“ sind ortsbedingt in erster Linie Professoren, Fachhochschul-Mitarbeiter und Studenten. Da nahezu 100 Prozent dieses Personenkreises Handybesitzer sind, wird erstmal niemand ausgeschlossen. Jede Altersstufe, die mit der Benutzung ihres Handys in den Grundzügen vertraut ist kann teilnehmen. Technisch bedingt können aber nur Nutzer mit bluetoothfähigen Geräten mitmachen, da die Applikation aus zwei Teilen besteht. Die Fischzucht am mobilen Gerät wäre prinzipiell möglich, nicht jedoch das Senden des Fisches ins Aquarium, und sein weiteres Fortbestehen dort.

### 2.1.2 Verwendungszweck

Das vorrangige Ziel von Fries4Fish ist, Professoren, Mitarbeiter und Studenten eine kleine unterhaltsame Abwechslung zum Studienalltag zu bieten. Im Gegensatz zu einem Tamagotchi, das man andauernd regelmäßig wie ein richtiges Lebewesen füttern, pflegen, unterhalten, usw. muss, und deshalb immer bei sich tragen sollte, zielt Fries4Fish darauf ab, den Benutzern einen kleinen Zeitvertreib zu bieten. Das Mobiltelefon ist meist sowieso ständiger Begleiter, wobei das ist für die Applikation kein wichtiger Faktor ist. Der User hat drei Events pro Tag, die er nutzen kann, aber nicht muss. Diese drei Aktionen wird er in der Regel einmal am Tag auskosten. Doch auch wenn er mehrere Tage nicht an seinen Fisch denkt, stirbt dieser nicht sofort, wenn er auch abgemagert sein wird. Diese kleine Unterhaltung für den Benutzer der Applikation wird noch verstärkt, durch die Möglichkeit, den eigenen Fisch ins Aquarium zu schicken, bzw. dort nach ihm zu sehen und sein Verhalten zu beobachten. Ein weiterer Aspekt könnte die Aufklärung der User über die Inhaltsstoffe des Essens und deren Auswirkungen auf den Körper sein. Dieser Aspekt ist in der realisierten Form jedoch nicht ausschlaggebend, und wird sehr abstrahiert umgesetzt.

### 2.1.3 Anwendungssituation

Konkret ist die Anwendung für den Eingangsbereich zur Mensa des neu entstehenden Gebäudekomplexes der Fachhochschule Augsburg an der Friedberger Straße realisiert worden. Es sind jedoch auch andere Einsatzorte denkbar, wobei sich der Einsatz in Cafés,

Restaurants oder Kantinen am besten eignet, da die gleiche Personengruppe regelmäßig dort verkehrt, und die Idee, die durch das chinesische Restaurants entstanden ist, bestehen bleibt.

## 3. Das Aussehen – Designaspekte

### 3.1 Allgemeines zur Erstellung

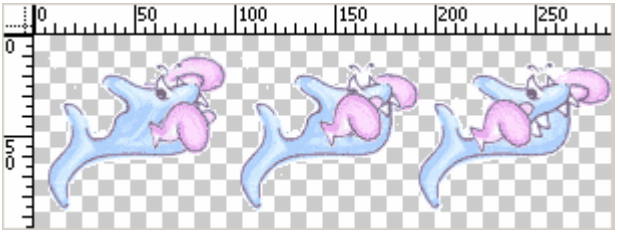
Grundsätzlich sollte die Gestaltung möglichst einfach gehalten werden, aber trotzdem eine eher von Hand gezeichnete Anmutung haben, nicht wie eine Pixelgrafik wirken, und auch als Vektorgrafik im Aquarium darstellbar und sofort wieder erkennbar sein.
Nach einigen Versuchen am Rechner, mit vektorisierten Zeichnungen zu arbeiten, wurde dieser Weg nicht weiterverfolgt, sondern eine für Handygrafiken eher ungewöhnliche Richtung eingeschlagen. Alle Objekte wurden von Hand mit Layoutstiften auf Papier erstellt, gescannt, und nach der anschließenden Bearbeitung mit dem Programm *Gimp* freigestellt, um im Icon-Programm *Axialis IconCreator* weiterbearbeitet zu werden. Hier wurden die Bilder dann Pixel für Pixel für verschiedene Größen optimiert, und anschließend wurde mit *Adobe Photoshop* die Dateigröße ohne Qualitätsverlust minimiert.
Aus dieser Arbeit resultieren nun diverse Grafiken, alle in den Formaten 32x32, 64x64 sowie 96x96 Pixel, jeweils mit weißem und als png-Dateien mit transparentem Hintergrund.
Für die Implementierung eignete sich die Benutzung von `Sprites`, was für die Erstellung der Bilder folgendes bedeutet. Um einen animationsartigen Ablauf von Einzelbildern zu erhalten, die später per Code einfach nacheinender „durchlaufen“ werden,

Bild 0

Bild 1

Bild 2

müssen die Grafiken (hier Bild 0 bis Bild 2) exakt aneinandergereiht werden. In diesem Beispiel hier ist jedes Einzelbild genau 96 Pixel breit. Ob die Anordnung horizontal oder vertikal erfolgt, spielt für die Implementierung keine Rolle.




fertiges Sprite

## 3.2 Fischarten und Grundformen

Aus einer Vielzahl von möglichen Auswahlformen für die Grundform des Fisches kristallisierten sich drei Formen heraus, für die ein Aquariumsverhalten implementiert wurde. Konkret sind das ein Kugelfisch, ein Knochenfisch und ein Raubfisch, wie im folgenden Bild zu sehen ist.

Kugelfisch Knochenfisch Raubfisch

Die Grafiken wurden speziell dafür entwickelt, und sollen durch ihre Verschiedenheit möglichst jedem Benutzer eine ansprechende Grundform für den Fischkörper bieten. Um die Ausdrucksstärke der jeweiligen Form am deutlichsten herauszustellen, werden alle Grafiken in ihrem Profil, also in einer Seitenansicht gezeigt.
Hier kann man am besten die verschiedenen „Dickenstufen“ erkennen, die der Fisch annimmt, wenn er beispielsweise zu wenig, oder aber im Gegenteil zu fettes Essen bekommt.

Raubfisch: normal dick dünn Kugelfisch: normal dick dünn

Weitere Ideen für Fischtypen werden in der nächsten Grafik aufgelistet.

Um eine größere Artenvielfalt zu garantieren, kann bei zukünftigen Implementierungen über diese oder ähnliche Fischtypen nachgedacht werden.

### 3.3 Farben

Für eine größere Unterscheidung und Individualisierung des vom Benutzer gewählten Fischtypus in Abgrenzung zu den anderen Fischen wählt der User eine Farbe für seinen Fisch. Die folgenden Farben stehen ihm dafür zur Auswahl bereit:

Gelb | Grün | Magenta | Rot | Türkis

Diese sollen möglichst gesättigt, und auf allen Endgeräten gleich darstellbar sein. Ein reiner dunkler Blauton ist aufgrund der Farbigkeit des Aquariums nicht realisierbar. Die Fische sollen sich möglichst gut vom Untergrund abheben, und vom Benutzer wieder erkannt werden, statt sich wie im realen Leben oft üblich, zu tarnen.
Durch die Kombination der einzelnen Fischformen mit den Farben und Dickenstufen ergibt sich eine große Vielfalt an unterschiedlich aussehenden individuellen Fischobjekten.

### 3.4 weitere Grafiken

Speziell für das Essen und die Trainingseinheiten wurden aus einer Vielzahl von Entwürfen folgende Grafiken für die Handy-Applikation realisiert.

Apfel | Bier | Hamburger | Karotte | Currywurst mit Pommes

Auch für die Trainingseinheiten Grafiken erstellt. Wenn man als Fischtyp den Raubfisch gewählt hat, bekommt man die erste unten abgebildete Grafik bei der Auswahl des Menüpunkts „Training“ zu sehen. Je schneller man das Training ausführt, umso schneller läuft die Animation ab.

Hier boxt der Fisch, je nach gedrückter Geschwindigkeit des Spielers. Als Farbe kann ein Blauton eingesetzt werden, da diese Animation nur am Handy gezeigt wird.

Für alle weiteren Fischtypen wird momentan eine Animation gezeigt, in der ein kleiner Fisch aus einer Welle springt, um anschließend wieder in sie einzutauchen.

## 4. Softwarearchitektur

### 4.1 Projektteam und Zuständigkeitsbereiche

An der Umsetzung der Projektidee waren folgende Personen mit nachstehenden Tätigkeitsschwerpunkten beteiligt:

- Claudia Lanzl
  Projektleitung, Implementierung der Menüstruktur der Handy-Applikation, Fischgenerierung am Handy, Erstellung der Grafiken, J2ME Polish, Dokumentation der Arbeit
- Alena Schneider
  Realisierung des Aquariums, also der gesamten Anwendung auf dem Server mit Flash / Action Script
- Andreas Bösselmann
  handyseitige Umsetzung des Trainings und der Fütterung des Fisches
- Dieter Hofbauer
  Bereitstellen der Schnittstelle zwischen der Handy-Applikation und dem Server, Designüberarbeitung des Menüs (CustomCanvas-Paket)
- Jewgenij Steinhart
  Implementierung des Speicherns und des Status des Fisches

Die einzelnen Punkte dieser Dokumentation wurden von den jeweils dafür zuständigen Mitgliedern verfasst. Dies ist durch eine Angabe hinter den Überschriften der entsprechenden Artikel gekennzeichnet. Alle ungekennzeichneten Passagen stammen von der Autorin dieser Dokumentation.

## 4.2 Zielsetzung

Ziel war es, in erster Linie ein möglichst einfaches und flexibles System zu entwickeln, das leicht erweitert werden kann. Es soll auf möglichst vielen Endgeräten genutzt werden können und dabei immer ein einheitliches Aussehen haben, das nach belieben modifiziert werden kann. Optimal wäre eine strikte Trennung von Funktionalität und Design.

## 4.3 Umsetzung

### 4.3.1 Ansatz mit J2ME Polish

Diese Anforderungen erfüllte ein Werkzeug, mit dem die Struktur der Applikation mit der Java 2 Micro Edition, kurz J2ME, realisiert werden kann, und das Design über verschiedene Cascading Style Sheets (CSS) festgelegt wird. „J2ME Polish ist eine hochintegrierte Kollektion von Werkzeugen für die Entwicklung von J2ME Anwendungen“. [J2P06] Die Einarbeitung in J2ME Polish nimmt etwas Zeit in Anspruch, aber dieser Zeitaufwand macht sich dafür anschließend bezahlt. Durch J2ME Polish kann eine Anwendung für beliebig viele Zielgeräte und Sprachen automatisch erstellt werden. Die Geräte-Datenbank umfasst mit über 300 Modellen alle gängigen Mobiltelefon-Typen mit ihren jeweiligen Fähigkeiten wie der Canvas-Grösse, APIs, unterstützten Formaten, etc.

Siemens

| Vendor | Device | GUI | Platform | APIs | ScreenSize | BitsPerPixel | Features |
|---|---|---|---|---|---|---|---|
| Siemens | 2128 | - | MIDP/1.0 CLDC/1.0 | siemens-game-api | 101x64 | 2 | midp, midp1, cldc, cldc1.0 |
| Siemens | 3138 | - | MIDP/1.0 CLDC/1.0 | siemens-game-api | 101x64 | 2 | midp, midp1, cldc, cldc1.0 |
| Siemens | A56i | - | MIDP/1.0 CLDC/1.0 | siemens-game-api | 101x64 | 2 | midp, midp1, cldc, cldc1.0 |
| Siemens | C55 | - | MIDP/1.0 CLDC/1.0 | siemens-game-api | 101x64 | 2 | midp, midp1, cldc, cldc1.0 |
| Siemens | C56 | - | MIDP/1.0 CLDC/1.0 | siemens-game-api | 101x64 | 2 | midp, midp1, cldc, cldc1.0 |
| Siemens | C60 | ✔ | MIDP/1.0 CLDC/1.0 | siemens-game-api, siemens-color-game-api, siemens-extension-api | 101x80 | 12 | midp, midp1, cldc, cldc1.0, supportsWMAPIWrapper |
| Siemens | C61 | ✔ | MIDP/1.0 CLDC/1.0 | siemens-game-api, siemens-color-game-api, siemens-extension-api | 101x80 | 12 | midp, midp1, cldc, cldc1.0, supportsWMAPIWrapper |
| Siemens | C65 | ✔ | MIDP/2.0 CLDC/1.1 | wmapi, location-api, jtwi | 130x130 | 16 | hasCommandKeyEvents, midp, midp2, cldc, cldc1.1, hasFloatingPoint |
| Siemens | CF62 | ✔ | MIDP/1.0 CLDC/1.0 | siemens-game-api, siemens-color-game-api, cf62-led-api, siemens-extension-api | 130x130 | 16 | midp, midp1, cldc, cldc1.0, supportsWMAPIWrapper |
| Siemens | CT56 | - | MIDP/1.0 CLDC/1.0 | siemens-game-api | 101x64 | 2 | midp, midp1, cldc, cldc1.0 |

Auszug aus der Gerätedatenbank

Konkret gibt man die gewünschten Endgeräte in der *build.xml*-Datei im Knoten `<deviceRequirements>` an. Die Art und Weise wie dies geschieht zeigt folgender Codeabschnitt:

```
<deviceRequirements>
    <requirement name = "Identifier"
                 value = "Nokia/6230, Siemens/S65" />
</deviceRequirements>
```

Die Anpassung an die Zielgeräte ohne die Portabilität der Anwendungen zu verlieren wird durch Preprocessing gewährleistet. Da die Build-Werkzeuge auf Ant basieren, kann J2ME Polish flexibel konfiguriert und in jede IDE – in unserem Fall Eclipse - integriert werden. Was die GUI betrifft kann man Anwendungen mit einfachen CSS Textdateien gestalten, z. B. durch Setzen von Attribute in der CSS-Datei wie:

```
background-color: rgb(123, 54, 233);

title{
       padding: 15;
       margin-top: 5;
       margin-bottom: 7;
       margin-left: 5;
       font-color: green;
       font-style: bold;
       layout: hotzontal-center | horizontal-expand;
       background: none;
       border: none;
}
```

Es können zudem Bilder, Bitmap-Fonts, unterschiedliche Hintergründe, Animationen und so weiter genutzt werden. Für detaillierte Informationen zur Installation sowie einer Übersicht welche Attribute in welchen Dateien untergebracht sind, wird an dieser Stelle auf das in diesem Zusammenhang von der Projektgruppe erstellte *How-to* verwiesen. Zu finden ist es auf beiliegender CD, sowie im Internet [CL06].
Wichtig ist, nicht wie gewohnt zu versuchen ein Midlet zu starten, sondern einen Ant-Build durchzuführen. Ebenso beachtet man möglicherweise anfangs die Präprozessor-Direktiven (beispielsweise: `//#style mainCommand`) in der Java-Datei nicht, da sie beim ersten Blick wie Kommentare anmuten. Dies sind wichtige Hinweise zur Arbeit mit diesem Werkzeug, eine tiefer gehende Beschreibung würde den Rahmen dieser Dokumentation sprengen. Bei weiterem Interesse wird an dieser Stelle auf die weiterführenden Links unter dem Menüpunkt *Quellen und weiterführende Links* am Ende dieser Ausarbeitung verwiesen.

### 4.3.2 Umsetzung ohne J2ME Polish

Leider konnte J2ME Polish in diesem Projekt dann letztendlich doch nicht zum Einsatz kommen, da das Ausführen eines Ant-Builds vom technischen Rahmen nicht unterstützt werden sollte. Die Applikation wurde folglich komplett mit J2ME – ohne Polish – realisiert.

## 4.4 Menüstruktur

### 4.4.1 Userinterface / Navigation

Um die Navigation so einfach und intuitiv wie möglich zu realisieren, wurden Interaktionsmuster aus diversen Bereichen recherchiert. Die typischen Navigations-Strukturen von interaktiven Handyapplikationen galt es zunächst herauszufinden.
Übliche Tastenbelegungen bei verschiedenen Handymodellen wurden erforscht und dienten als Grundlage für die hier entstandene Struktur.
Die Fries4Fish-Applikation umfasst grundlegend drei Funktionsebenen wie sie in der folgenden Grafik gezeigt werden.


Die Erstellung des Fisches, falls man die Applikation zum ersten Mal startet, das Züchten des Tieres, indem man ihm Sätze trainiert, ihn füttert oder Sport machen lässt, sowie die Möglichkeit den Fisch in das virtuelle Aquarium zu entlassen.

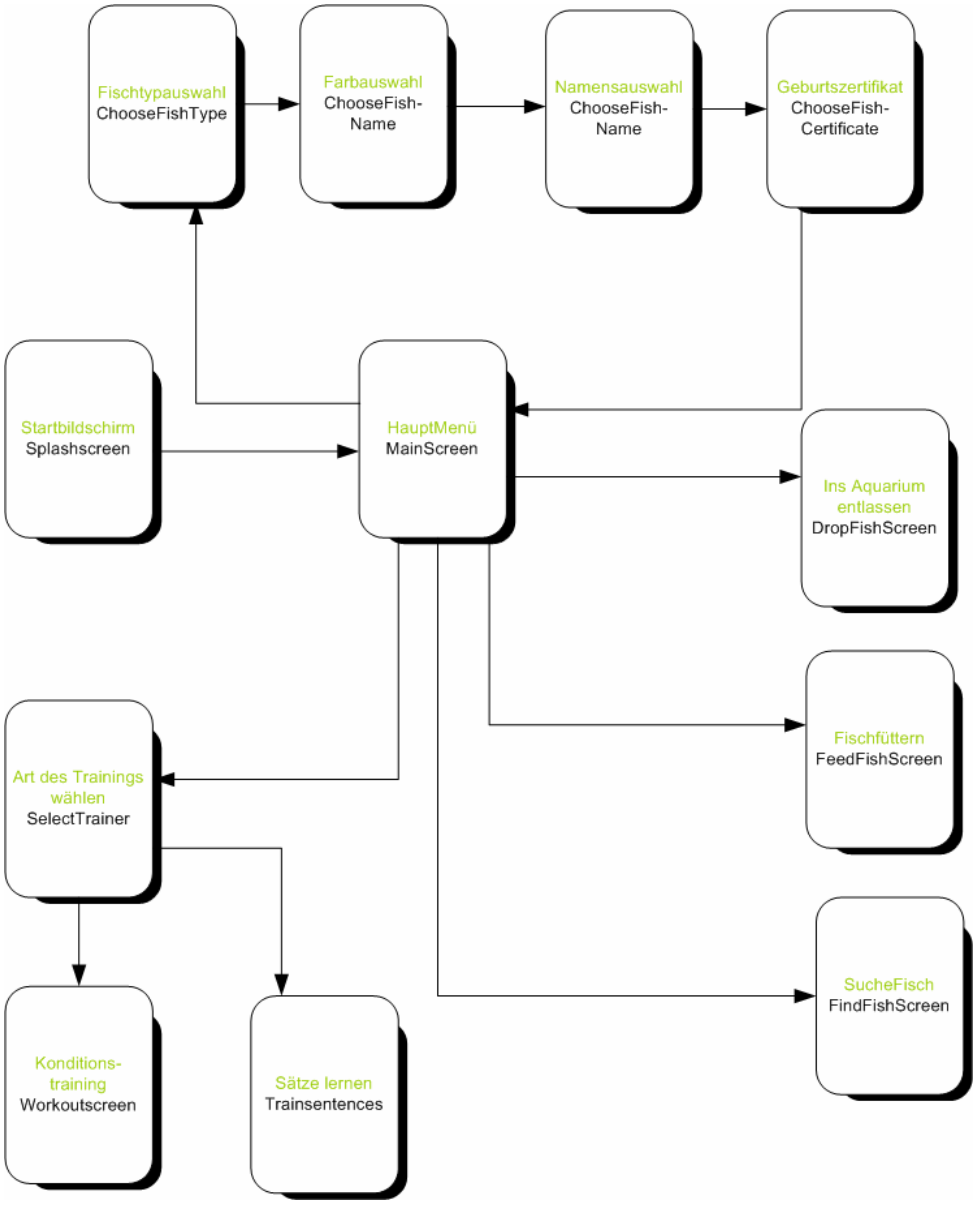
Diese drei Ebenen lassen sich detaillierter unterteilen, und stehen in folgendem Zusammenhang:
Das Herz der Applikation bildet das Hauptmenü, das nach dem Startbildschirm gezeigt wird. Hier hat der Nutzer die Auswahl zwischen nachstehenden Varianten:

- Erstellen eines Fisches, falls noch kein Fischobjekt am Handy existiert; dieser Menüpunkt gliedert sich auf in Fischtypauswahl, Farbauswahl, die Namensgebung und das anschließende Übersichtszertifikat.
- Trainieren des Fisches;

hier besteht die Möglichkeit dem Fisch entweder Sätze zu lernen, oder seine Kondition zu trainieren.

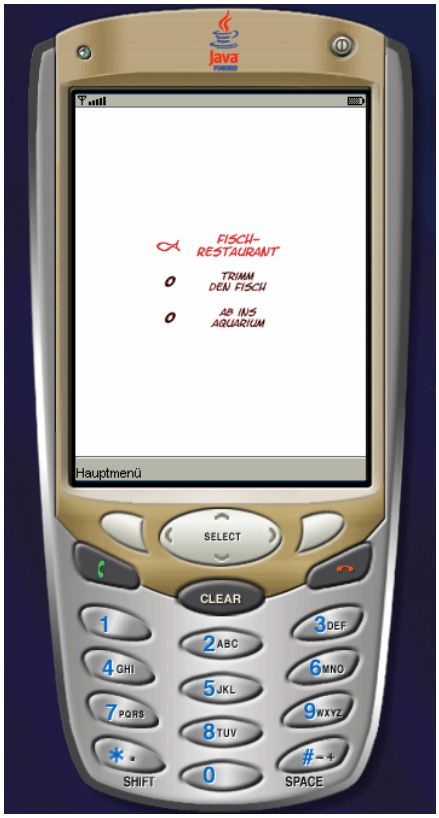
- Fütterung des Fisches;
- den Fisch ins Aquarium zu schicken;
- Suchen des Fisches;

  dieser Faktor kommt erst zu r Anwendung, wenn man seinen Fisch schon im Aquarium hat, ihn besucht, und nicht im sichtbaren Bereich schwimmen sieht. Nach Auswahl dieser Option schwimmt der ehemals gezüchtete Fisch dann in den Bereich, wo ihn der Nutzer sehen kann.



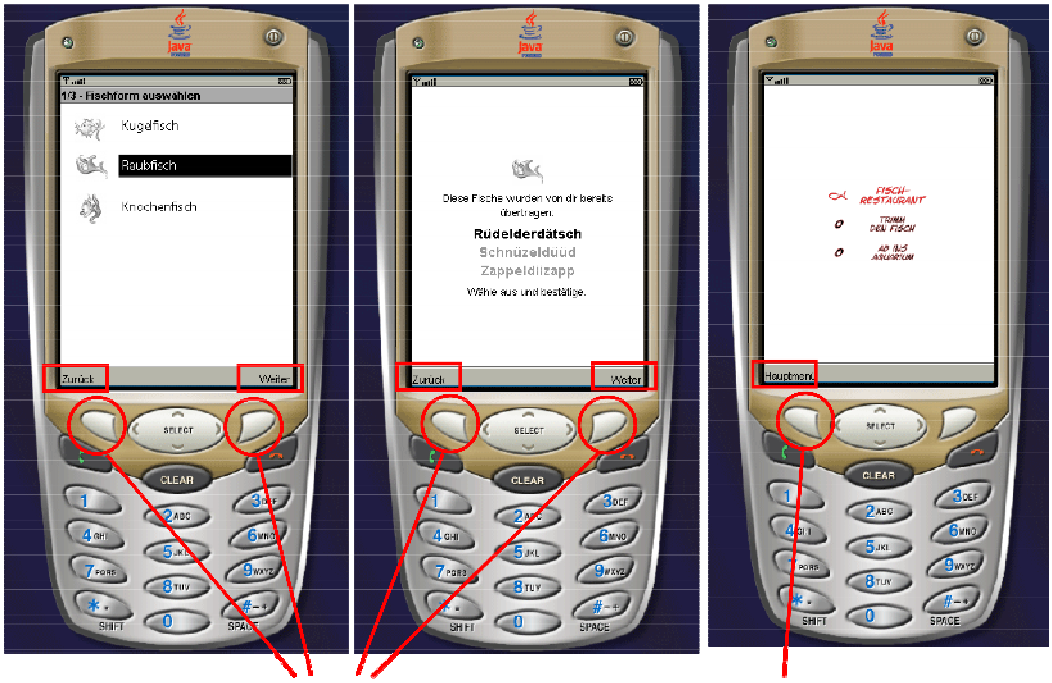

Darstellung der möglichen Navigationsebenen

In der aktuellsten Version der Applikation gibt es ein intelligentes Hauptmenü. Das heißt es wird nicht immer das gleiche Menü angezeigt, sondern unterschieden, ob der Besitzer aktuell einen Fisch am Handy hat, oder ob dies nicht der Fall ist.
Die beiden möglichen Menüvarianten werden hier kurz gezeigt.


Auf dem linken Bild ist das Hauptmenü zu sehen, das beim Start der Applikation geladen wird, falls aktuell kein Fisch am Handy existiert. Der Anwender hat dann die Möglichkeit einen neuen Fisch zu erstellen, oder falls er einen früher erstellten und schon ins Aquarium entlassenen Fisch besuchen möchte, diesen zu sehen.
Auf dem rechten Screenshot kann ein existierender Fisch gefüttert, trainiert oder ins Aquarium geschickt werden.

Die Navigationsstruktur und Tastenbelegung in der Menü-Ebene, sowie einer Hierarchie tiefer, wird durch die drei folgenden Grafiken veranschaulicht.

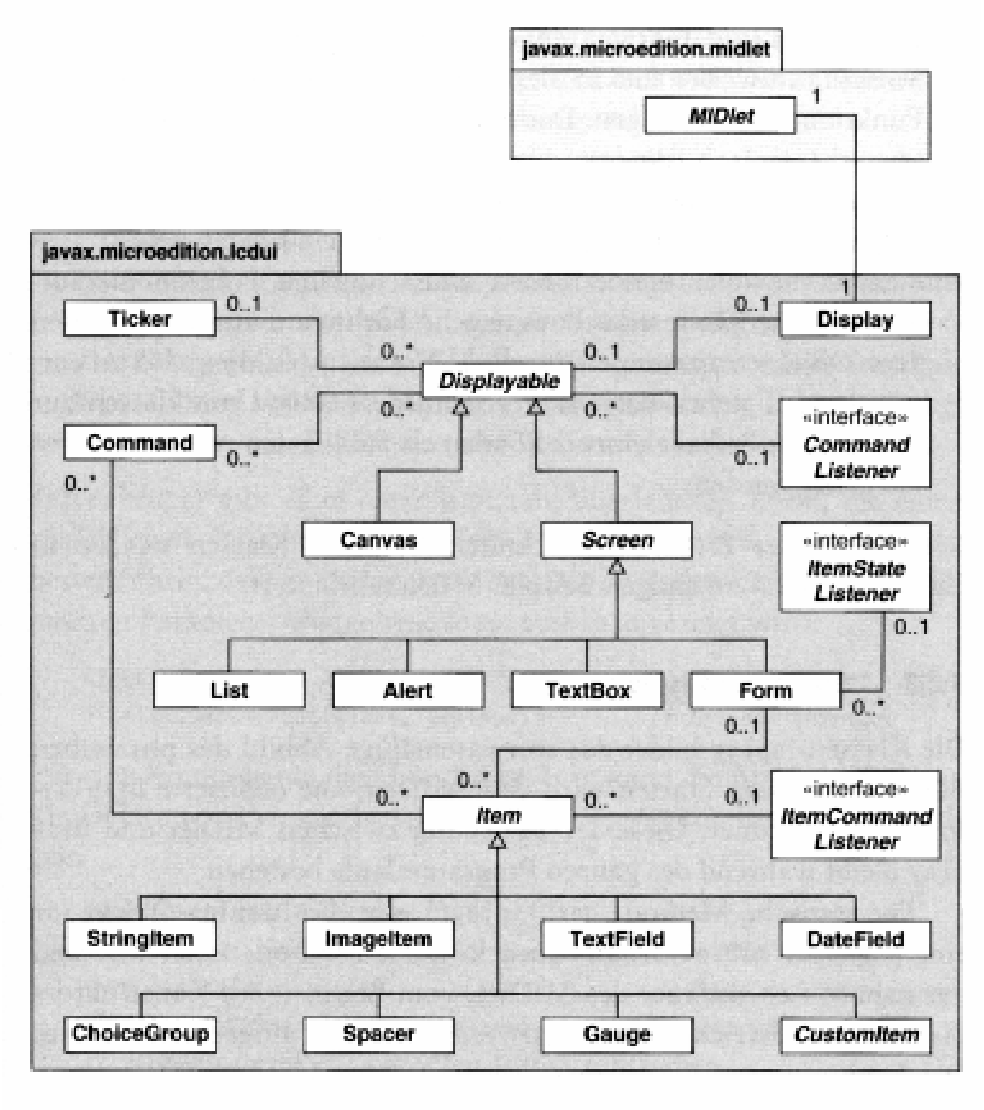



Softlinks zum Vor- und Zurückspringen

Beenden auf Hauptmenü-Ebene

### 4.4.2 Realisierung

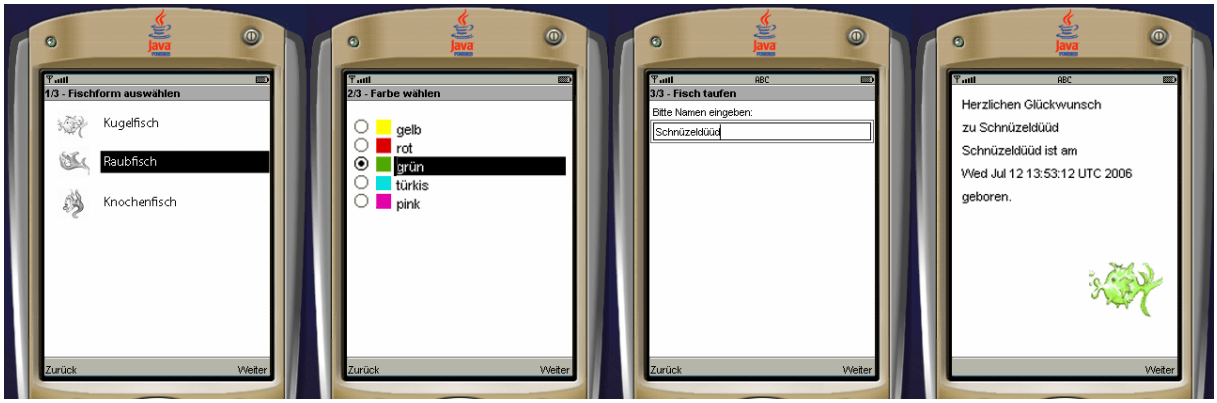
Für die Umsetzung der Menüstruktur (package `publicScreenClientFish` → package `screen`) wurde im ersten Schritt auf das High-Level-API zurückgegriffen, um eine möglichst gute Portabilität zu gewährleisten. Erweiterungen fanden anschließend unter Einsatz des Low-Level-APIs statt, da hiermit das *Look and Feel* der Applikation wesentlich stärker beeinflusst werden kann, und dieser Ansatz am weitesten die Idee, die ursprüngliche mit J2ME Polish verfolgt wurde, erfüllt. Die Komplexität nimmt dadurch unverhältnismäßig zu, deshalb soll in diesem Teil der Ausarbeitung in erster Linie auf die von der Klasse `Screen` abgeleiteten essentiellen Klassen zur Menüerstellung eingegangen werden. Anschließend wird dann auf die `Canvas`-Klassen eingegangen (siehe Menüpunkt 4.8).
Das Klassendiagramm des LCDUI verschafft einen wichtigen Überblick über die einzelnen Klassen und deren Vererbungshierarchie.




Quelle: [KDS04]

Bei der Programmierung wird man sich als Einsteiger möglicherweise anfangs wundern, warum gewisse Dinge nicht funktionieren oder Methoden nicht implementiert sind, z. B. würde man eventuell erwarten, dass man in eine `List` eine `ChoiseGroup` einfügen kann. Hier kann als Einstieg die Übersicht der LCDUI-Klassen helfen.

Nun werden stellvertretend einige der implementierten Menü-Screens, die zur Erstellung des Fisches dienen, mit Handy-Screenshots und Erklärungen vorgestellt.




1-ChooseFishType 2-ChooseFishColor 3- ChooseFishName 4-ChooseFishCertificateCanvas

Die Auswahl der Fischform (class `ChooseFishType`) ist als `List` implementiert, an die die einzelnen Menüpunkte einfach angehängt werden können. Hier kann neben dem Text auch eine Bilddatei angegeben werden, wie hier die Grafiken für die verschiedenen Fischformen. Der jeweils ausgewählte Punkt, beispielsweise „Raubfisch“ wird beim Klick auf den Weiter-Button gespeichert, und man gelangt zum zweiten der drei Erstellungs-Screens. Hier in der Klasse `ChooseFishColor` nimmt eine `Form` eine `ChoiseGroup` auf, an die die verschiedenen Items (hier die Farben) angehängt werden. Im letzten Erstellungsschritt des Fisches, implementiert in der Klasse `ChooseFishName` nimmt wiederum eine Form ein TextField auf, in das der Benutzer den gewünschten Namen eingeben kann. Durch Drücken auf den Weiter-Button wird der fertige Fisch gespeichert, und der Ersteller bekommt den Folge-Bildschirm zu sehen, eine Zusammenfassung des Fisches.
Hier (`ChooseFishCertificateCanvas`) wird nicht mit der LCDUI-Klasse `Screen` gearbeitet, sondern mit der Klasse `Canvas`, um Texte und das Bild des erstellten Fisches frei auf der zur Verfügung stehenden Display-Fläche anordnen zu können.

### 4.4.3 Design-Veränderung des Menüs (Claudia Lanzl, Dieter Hofbauer)

Durch das Bestreben, auch nach dem Wegfall der J2ME Polish-Lösung ein grafisch anspruchsvolles animiertes Menü zu generieren, wurden einige Menüpunkte mit einer über die Klasse `Canvas` realisierten Optik umgeschrieben.
Bei der grafischen Umsetzung kamen spezielle Klassen aus dem Paket `CustomCanvas` zum Einsatz. Hierbei werden Instanzen der Klassen `AnimatedMenuItem`, einer so genannten `MenuItemGroup` zugeordnet, mit deren Hilfe die Funktionalität des Menüs hergestellt wird. Jeder `AnimatedMenuItem` funktioniert dabei als eigenständiger Thread, der mit einer beliebigen Anzahl von animierbaren Sprites versehen, seine Animationsrichtung danach ausrichtet, wie sein jeweiliger aktueller Status in der `MenuItemGroup` ist.

Das Paket `CustomCanvas` wird an einer späteren Stelle dieser Ausarbeitung näher diskutiert.

## 4.5 Weitere Menüpunkte (an der Schnittstelle)

### 4.5.1 „Fisch senden“ (Dieter Hofbauer)

Die grundlegende Bildschirmstruktur wurde unter Verwendung der nachstehend erwähnten Klassen aus dem CustomCanvas-Paket erzeugt. Unabhängig von der verwendeten Bildschirmgröße ist damit ein einheitliches Layout auf dem jeweiligen Ausgangshandy möglich. Realisiert ist bislang ein zentriertes Layoutsystem. Eine Erweiterung an z.B. ein linksbündiges Layout ist aber ohne großen Aufwand möglich.
Für die Funktionalität der Fischübertragung wurden gemäß den Vorgaben der Technikgruppe eine spezielle Deserializer-Klasse und eine eigens für diese Aufgabe konzipierte Message generiert. (FishUploadmessage). Des weiteren wurden verschieden ActionTypes die den Kommunikationsfluss nach Abschluss der Fischübertragung regeln, entworfen.
Der Ablauf vollzieht der Fish-Senden-Funktion funktioniert im wesentlichen in folgenden Schritten: zuerst wird eine neue FishUploadMessage aus den aktuellen State-Attributen gebildet. Danach wird die Message serialisiert, per Bluetooth übertragen und nach dem Empfang wieder deserialisiert. Anschliessend werden die Parameter für die Fisch-Generierung ausgelesen und der zugehörige Flash-Funktionsaufruf generiert und an den Flash-Client übertragen. Dort wird die jeweilige den Fish erzeugende Funktion aufgerufen. Über Ausnahmefälle, die bei der Fischübertragung auftreten können, so z.B. dass aktuell keine Bluetooth-Verbindung besteht, dass noch kein Fisch erzeugt ist oder dass bei der Übertragung ein Fehler passiert ist, wird mittels eigener Screens informiert.
Nach erfolgreicher Übertragung wird der Fisch in einer internen Liste auf dem Recordstore mittels einer eindeutigen ID aus Bluetooth-ID und aktueller Zeit, die auch zur Identifizierung im Aquarium verwendet wird, abgespeichert.

### 4.5.2 „Fisch finden“ (Dieter Hofbauer)

Auch diesem Menüpunkt liegen die im Folgenden erklärten Klassen aus dem CustomCanvas-Paket zugrunde. Vom Ablauf her wird zuerst überprüft, ob bereits Fische an das Aquarium übertragen wurden und ob aktuell eine Bluetooth-Verbindung besteht. Ist beides der Fall wird auf den Screen verzweigt, der die Fischsuche im Aquarium veranlasst. Anhand einer Auswahlliste kann der Benutzer aus den bisher gewählten Fischen auswählen und genau diesen im Aquarium wieder finden lassen.

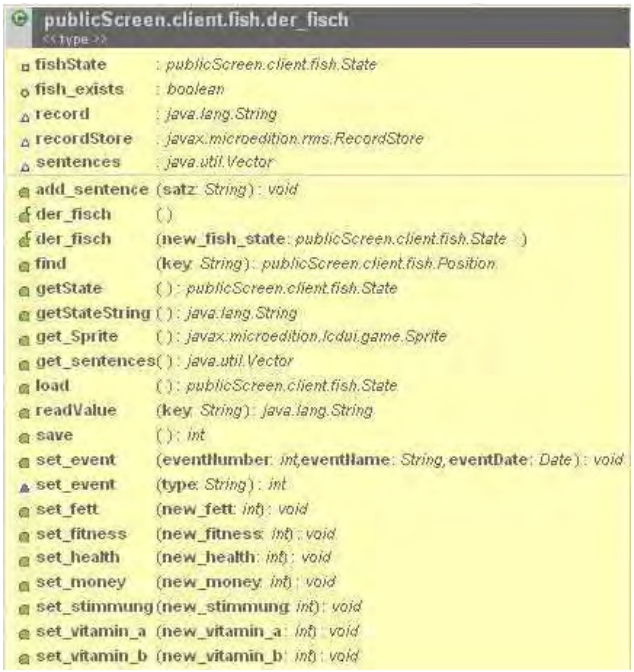
Für die Funktionalität des Fischfindens wurden gemäß den Vorgaben der Technikgruppe auch hier eine spezielle Deserializer-Klasse und eine eigens für diese Aufgabe konzipierte Message (FishFindMessage) generiert. Des weiteren wurden verschieden ActionTypes die den Kommunikationsfluss nach Abschluss der Fischübertragung regeln, entworfen.
Vom internen Ablauf her wird nach Auswahl des Fisches dessen ID in der internen Liste auf dem Recordstore nachgeschlagen und diese in einer FishFindMessage per Bluetooth an den Sender übertragen. Das ServerPlugin sendet daraufhin einen Funktionsaufruf an den Flash-Client, der mithilfe der übertragenen ID die Suche auf der Aquariumsseite einleitet.

### 4.6 Fischrepräsentation und Speicherung im RecordStore (Jewgenij Steinhart)

In folgenden Unterabschnitten werden Klassen und Methoden beschrieben, die die Speicherung des Fisches im RecordStore regeln und den Zugriff auf die Parameter des Fisches ermöglichen. Bei veralteten Methoden, oder Methoden, die aus-schließlich für die interne Benutzung gedacht sind, auch wenn nicht als privat deklariert, wird von einer detaillierten Beschreibung abgesehen.


Die Klasse `State` enthält die meisten Parameter des Fisches und wurde als eine

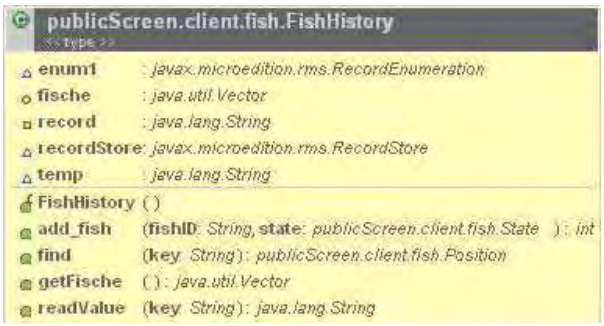
„Transportklasse“ konzipiert, um die Fischdaten zwischen Methoden und Klassen auszutauschen. Beim Speichern eines Fischzustandes im RecordStore werden die Attribute der `State`-Klasse in einen `String` zusammengefasst und mit XML-ähnlichen Tags versehen, damit die Daten möglichst einfach aus dem `String` extrahiert und die ursprüngliche Datenstruktur wiederhergestellt werden kann.




Die Klasse `der_Fisch` (der Name der Klasse sollte in späteren Version geändert werden um den gängigen Konventionen zu entsprächen) ist die eigentliche Repräsentation des Fisches im Programm. Die Klasse wurde als Singleton konzipiert, sprich nur eine Objektinstanz der Klasse sollte erstellt werden, der Zugriff sollte über entsprechende Methoden erfolgen. In der aktuellen Version des Programms, wurde jedoch eine etwas abweichende Implementierung realisiert: ein neues Objekt der Klasse kann an jeder beliebigen Stelle des Programms intanziiert werden, die Fischdaten werden vom Klassenkonstruktor aus dem RecordStore geholt und man erhält die Möglichkeit auf die aktuellen Daten über diese Instanz zuzugreifen. Dies ist eine Kompromisslösung und sollte in einer späteren Version wieder geändert werden.
`der_Fisch` enthält eine Instanz von `State` und bietet Methoden, mit denen der Zustand des Fisches geändert werden kann.
Es gibt zwei Konstruktoren der Klasse, der parameterlose holt die Daten des Fisches, falls welche vorhanden, aus dem RecordStore, sonst werden die Defaults und ein Flag, dass kein Fisch existiert, gesetzt. Der zweite Konstruktor bekommt ein Objekt der Klasse `State` als Parameter und generiert daraus einen neuen Fisch mit vorgegebenen Eigenschaften. Der Konstruktor wird z. B. beim Neuanlegen eines Fisches verwendet. Die Methode

`add_sentence` bringt dem Fisch einen neuen Satz bei. `get_sentences` liefert einen `Vector` mit gelernten Sätzen zurück.
`get_Sprite` gibt einen `Sprite` mit korrekter Fischdarstellung zurück (Form, Farbe und Fett berücksichtigt) zurück, wird z. B. vom Füttern-Screen benötigt.
Die Methode `load` wird vom Konstruktor aufgerufen und lädt die gespeicherten Fischdaten aus dem RecordStore, `readValue` und `find` sorgen für die Rückumwandlung der Attribute aus dem `String` in `State`.
`set_event(String)` speichert ein Event, bzw. gibt beim Erreichen der Maximalen Eventzahl in einer Zeiteinheit 1 zurück. Wird z. B. dazu verwendet die Anzahl Fütterungs- und Trainingsaktionen auf vier pro Tag zu begrenzen.

### 4.6.1 Fischarchivierung




FishHistory


ArchivFish

Die Klassen `FishHistory` und `ArchivFish` dienen der Archivierung der ins Aquarium entlassener Fische. Die Klasse `ArchivFish` entspricht weitestgehend der `State`-Klasse, wobei die Anzahl der Attribute reduziert und die Methode `get_Sprite` hinzugefügt wurde.
Die Methode `add_fish` der Klasse `FishHistory` löscht den aktuellen Fisch aus `der_Fisch` und fügt ihn zum Archiv hinzu. Die Methode wird beim entlassen eines Fisches ins Aquarium ausgeführt.
`getFische` gibt den Vector mit Fischen aus dem Archiv zurück.

## 4.7 Implementierungen

### 4.7.1 Training (Andreas Bösselmann)

Das Training ist in zwei Teilbereiche untergliedert. Hierbei gibt es eine Trainingseinheit um dem Fisch die Möglichkeit zu geben mit seinem Besitzer und der Außenwelt zu

kommunizieren. Um essentielle Werte des Fisches zu trainieren wurde eine Trainingssimulation programmiert mit Hilfe derer man die Möglichkeit hat Fettwert und Kondition seines Schützlings zu verändern.

#### 4.6.1.1 Sprachtraining

Dadurch dass der Fisch nur eine begrenzte Zeit auf dem Handy des Benutzers verbringt und anschließend seinen Lebenszyklus im Aquarium vollenden wird, entstand die Idee, dass der Fisch weiterhin in Verbindung mit dem Benutzer steht, wenn er das Handy verlassen hat.


Mit Hilfe des Sprachtrainings hat der Benutzer die Möglichkeit seinem Fisch eine bestimmte Anzahl von Sätzen beizubringen, die der Fisch auf dem PublicScreen bei Annäherung des Besitzers von sich gibt. Beispielweise Grußformeln oder Liebesbeweise. Im aktuellen Prototypen ist die Anzahl der Sätze auf Drei beschränkt. Allerdings hat der Benutzer die Möglichkeit über einen Menüpunkt den letzten Satz zu löschen. Bei der Eingabe des Satzes wird ein eigener Bildschirm aufgerufen auf dem bis zu 20 Zeichen Text eingegeben werden können. Der Wert von 20 Zeichen kommt daher, weil bei dem Fisch auf dem Public Screen der Name gegen einen der gelernten Sätze ausgetauscht wird, bei zu langem Text würden andere Fische verdeckt werden.

Im Grunde besteht der Satztrainieren Bildschirm aus einer Klasse abgeleitet von Canvas, um einen möglichst großen Gestaltungsfreiraum zu haben. Da das Trainieren von Sätzen den Benutzer Geld kostet, wird der aktuelle Kontostand im Eck rechts oben angezeigt. Direkt darunter befindet sich die graphische Repräsentation des Fisches. Diese ist vom aktuellen Zustand des Fisches abhängig. Über ein der_fisch Objekt das beim Instanzieren den aktuellen Status aus dem Record Store des Handys lädt, wird die Graphik mithilfe der Funktion und der zugehörigen Instanzmethode *get_sprite(),* entsprechend angezeigt. Näheres dazu im Kapitel Record-Store.


Unter der Fischgraphik findet man eine Liste der gelernten Sätze oder, wenn keine vorhanden sind einen Platzhalter. Mittels der „Select“ Taste des Handys gelangt der Benutzer zum Eingabebildschirm. Vorher wird im State überprüft ob der Benutzer die maximale Anzahl an Sätzen überschreiten würde(aktuell 4), und ob ausreichend Geld vorhanden ist. Dieser Eingabebildschirm ist von TextBox des WTK’s abgeiletet. Dies

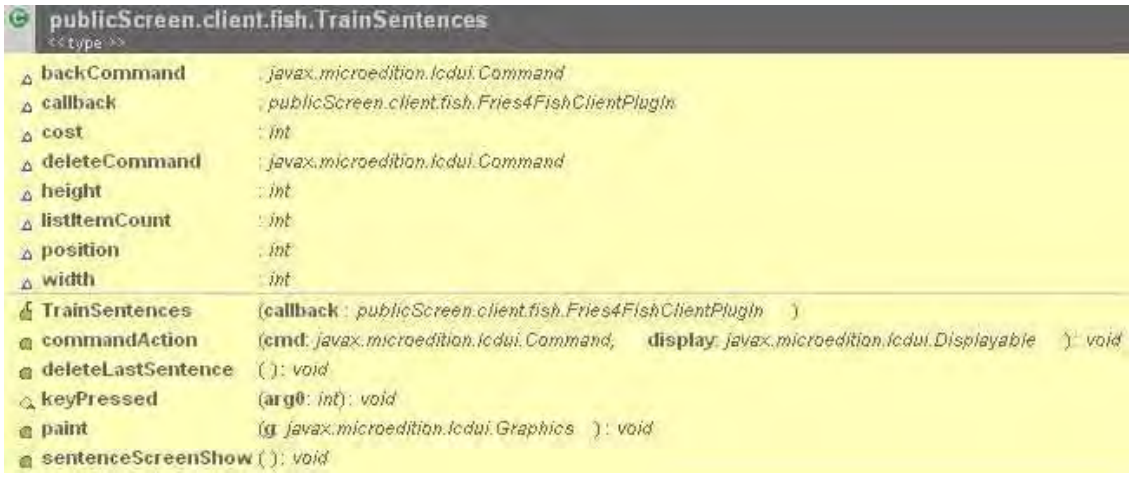
hat im Gegensatz zur manuellen Implementierung auf der Canvas den Vorteil, dass etwaige Texteingabeunterstützungsfunktionen des Handys verwendet werden können. Als bekanntestes Beispiel zählt hierzu die T9 Wörterbucheingabe. Über den Menübefehl „Satz lernen“ wird der Satz gespeichert und die anfallenden Kosten vom verfügbaren Geldbestand abgezogen.
Der Satz wird hierbei sofort über eine der_fisch Instanz im Recordstore abgelegt. Dazu wird die Instanzmethode *addSentence()* aufgerufen. Bei der Rückkehr zur vorherigen Canvas wird der soeben trainierte Satz nun in der Liste angezeigt.
Um nicht gewünschte Sätze in der Liste verändern zu können, steht Menüpunkt „letzten Satz löschen“ zur Verfügung. Dieser löscht wie der Name schon sagt das letzte Item aus der Liste. Intern fängt der Commandlistener den Menübefehl ab und ruft die Funktion *deleteLastSentence()* die Funktion sieht folgendermaßen aus:

```
public void deleteLastSentence(){
             der_fisch fishObj=new der_fisch();
             if(fishObj.get_sentences().size()>0){
             fishObj.get_sentences().removeElementAt(0);
             fishObj.save();
             }
             repaint();
}
```

Hier kann man gut das Prinzip des Fischstatuses erkennen. Über die Klasse der_fisch wird ein Fischobjekt erzeugt, dass sämtliche Eigenschaften des virtuellen Fisch repräsentiert. Nach Veränderung einer Variablen muss immer die save() Methode ausgeführt werden, damit die Änderungen in den Recordstore geschrieben werden und nicht verloren gehen. Wichtig ist, dass ein Abschließendes *repaint()* aufgerufen wird, so dass die Canvas neugezeichnet wird und die angezeigten Sätze mit den tatsächlich im State vorhanden wieder übereinstimmen.

**Weitere Realisierungsmöglichkeiten für spätere Versionen:**
Zum einen war es angedacht das der Benutzer maximal drei Aktionen pro Tag zu Verfügung hat um den Fisch zu füttern trainieren usw. um das Langzeitinteresse zu erhalten. Ein Ansatz hiervon ist bereits im der_fisch Objekt vorhanden. Eine Abfrage auf die verfügbaren Aktionen könnte also auch hier geschehen.

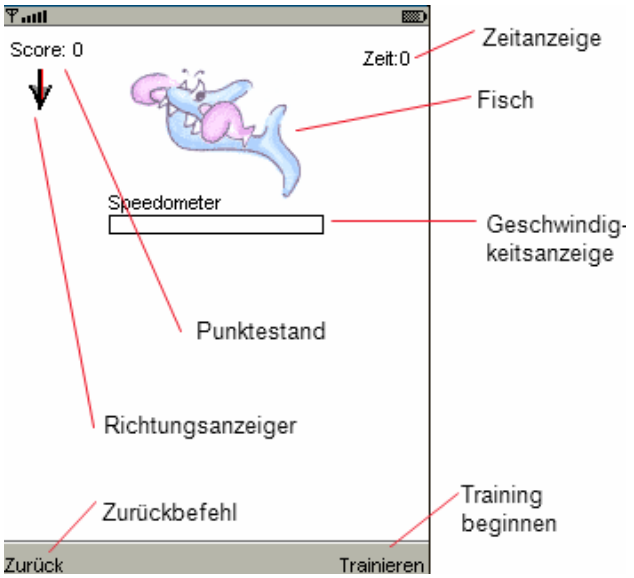
Die Klasse im Überblick:




### 4.7.1.2 Fitnesstraining

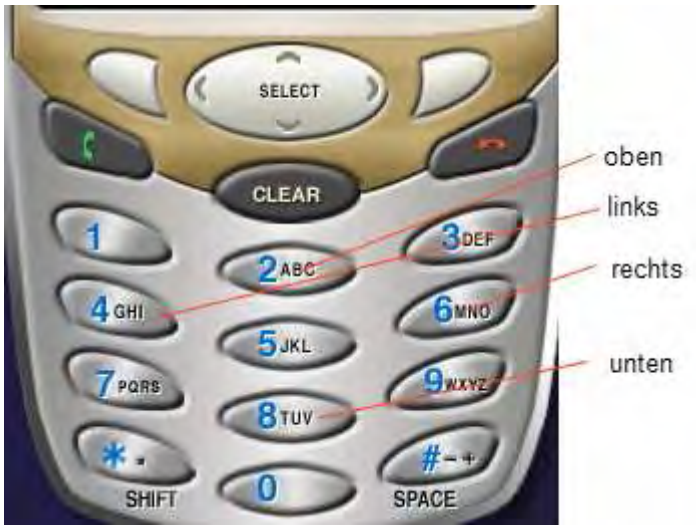

Durch die reichhaltige und fetthaltige Kost die der Fisch bei der Fütterung oft zu sich nimmt, steigt der Fettgehalt des Fisches stark an. Dies Führt zu einer physischen Ausdehnung des Fisches, und durch den dadurch verbunden höheren Wasserwiderstand sinkt die Fortbewegungsgeschwindigkeit des Fisches, was ihn im Aquarium leichte Beute für schnelle Raubfische macht. Dem Entgegenzuwirken hat der Trainer die Möglichkeit seine Zucht im virtuellen Fitnessstudio zu trainieren. Während des Trainings sinken Fettwerte bis zu einem Gewissen Prozentsatz vom Fisches, in Abhängigkeit der Geschicklichkeit des Trainers. Natürlich versteht sich von selbst, dass ein schwer Übergewichtiger Fisch nicht nach einer Trainingseinheit zum flinken, schlanken Hecht wird, sondern nur ein ausgewogenes Fütterungsprogramm und konstantes Training zum Erfolg verhilft.

Aber nicht nur der Fett Gehalt kann reduziert werden, auch die körperliche Fitness (die den Fisch in Späteren Versionen vor Krankheiten bewahren soll und ihn im Kampf mit anderen Fischen zu einer gewissen Überlegenheit  verhelfen soll), kann im Training rasch erhöht werden.
Die Trainingsumgebung ist folgendermaßen aufgebaut:
Nachdem der Trainer das virtuelle Studio betreten hat, sieht er einen Fisch in der Vorbereitungsphase. Bis zum Start des Trainingsprogrammes über den Menüpunkt „Trainieren“  hat der Benutzer keine Eingabemöglichkeiten.
Kurz zu den einzelnen Anzeigeelementen:
In der Ecke links oben sieht der Trainer den Punktestand der momentan auf 0 steht da ja nicht trainiert worden ist. Auf selber Höhe rechts davon Sieht er die Zeitanzeige die Momentan auf Null steht. Darunter bekommt der Nutzer eine abstrakte Repräsentation des Fisches zu sehen (Hier ein Boxerfisch). In der aktuellen Version wird keine Abbildung des tatsächlichen Fischtyps gezeigt, da dies einen sehr hohen gestalterischen Aufwand mit sich bringen würde. Der Fisch wird in Form eines Sprites  dargestellt.
Der Aufbau dieser Sequenz aus png-Einzelbildern, die alle dieselbe Größe besitzen wird in Kapitel 3 „Das Aussehen – Designaspekte“ näher beschrieben.
Auf dem Startbildschirm ist das Bild natürlich noch statisch und wird erst beim aktivieren des Trainings animiert. Unter der Fischgraphik befindet sich das sog. Speedometer. Dieses gibt dem Benutzer eine graphisches Feedback über den Trainingserfolg des Fisches.
Links neben der Fischgraphik ist der Richtungsanzeiger vorhanden der dem Spieler die zu drückende Taste anzeigt.
Der Pfeil besteht wie auch der Fisch aus einem Sprite. Allerdings besteht der Pfeil aus nur einer Graphik die mithilfe der Sprite_Transformation Funktionen gedreht oder gespiegelt wird.



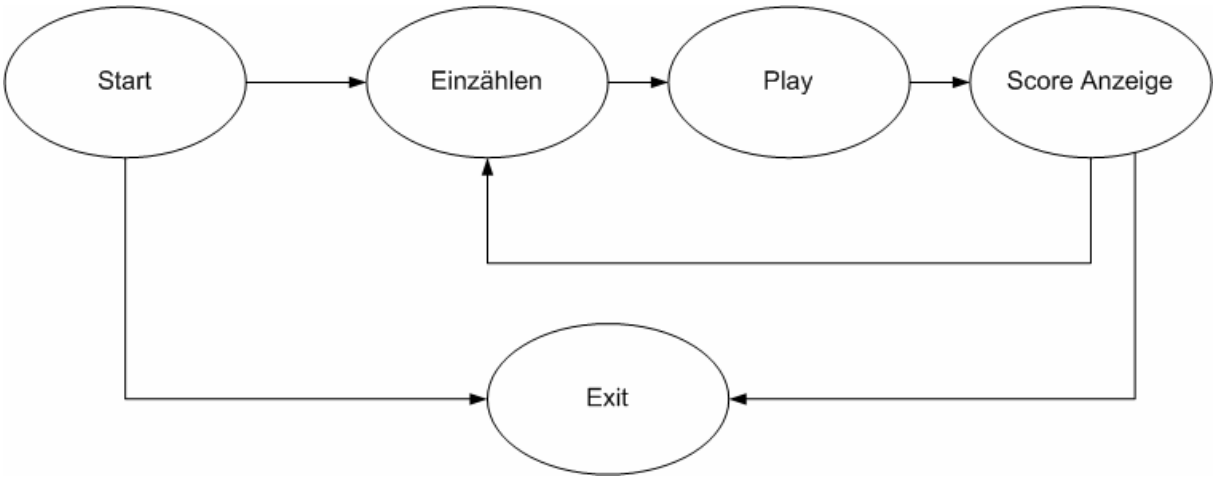

Trainingbildschirm bei Start

Zur Spielidee:
Der Spieler muss in einer bestimmten Zeit einen möglichst hohen Punktestand erreichen. Während des Spielverlaufes muss die Trainingsgeschwindigkeit auf einem hohen Level gehalten werden, damit der Spieler Punkte bekommt. Je höher der Level desto höher sind die Punkte die dem Gesamtpunktestand pro Sekunde hinzuaddiert werden. Der Spieler kann den Geschwindigkeitslevel erhöhen indem er die Taste auf seinem Handy drückt die dem Pfeil im linken oberen Eck entspricht. Der Pfeil wird nach drücken einer Taste per Zufallsgenerator auf eine neue Richtung gestellt.
Also wenn der Pfeil nach unten zeigt muss die Taste 8 auf der Handytastatur gedrückt werden.




Tastatur

Je schneller die richtigen Tasten hintereinander gedrückt werden desto höher wird der Speedlevel graphisch ist dies über das Speedometer sichtbar. Mit ihm steigt auch die Durchschaltgeschwindigkeit der Sprite-Frames, so dass der Spieler den Eindruck hat, dass der Fisch schneller boxt. Um das Spiel etwas schwieriger zu gestallten sinkt  mit steigendem Geschwindigkeitslevel die Zeit, die der Nutzer zur Verfügung hat um die nächste Taste zu drücken. Wenn er längere Zeit keine Taste drückt sinkt der Speedlevel schließlich auf Null und es werden keine Punkte mehr addiert. Auch wenn eine falsche Taste gedrückt wird sinkt der Geschwindigkeitslevel um eine Einheit.
Maximal kann der Level auf 10 gesteigert werden.
Nach Ablauf der Spielzeit bekommt der Trainer den Ergebnissbildschirm angezeigt. Auf diesem sieht man den erreichten Score sowie Fatgehalt und Kondition vor und nach dem trainieren. Von diesem Bildschirm kann der User entweder das Training verlassen und zum Hauptmenü zurückkehren oder eine weitere Trainingseinheit starten.
Die verschiedenen Stati sind in folgender Graphik veranschaulicht.



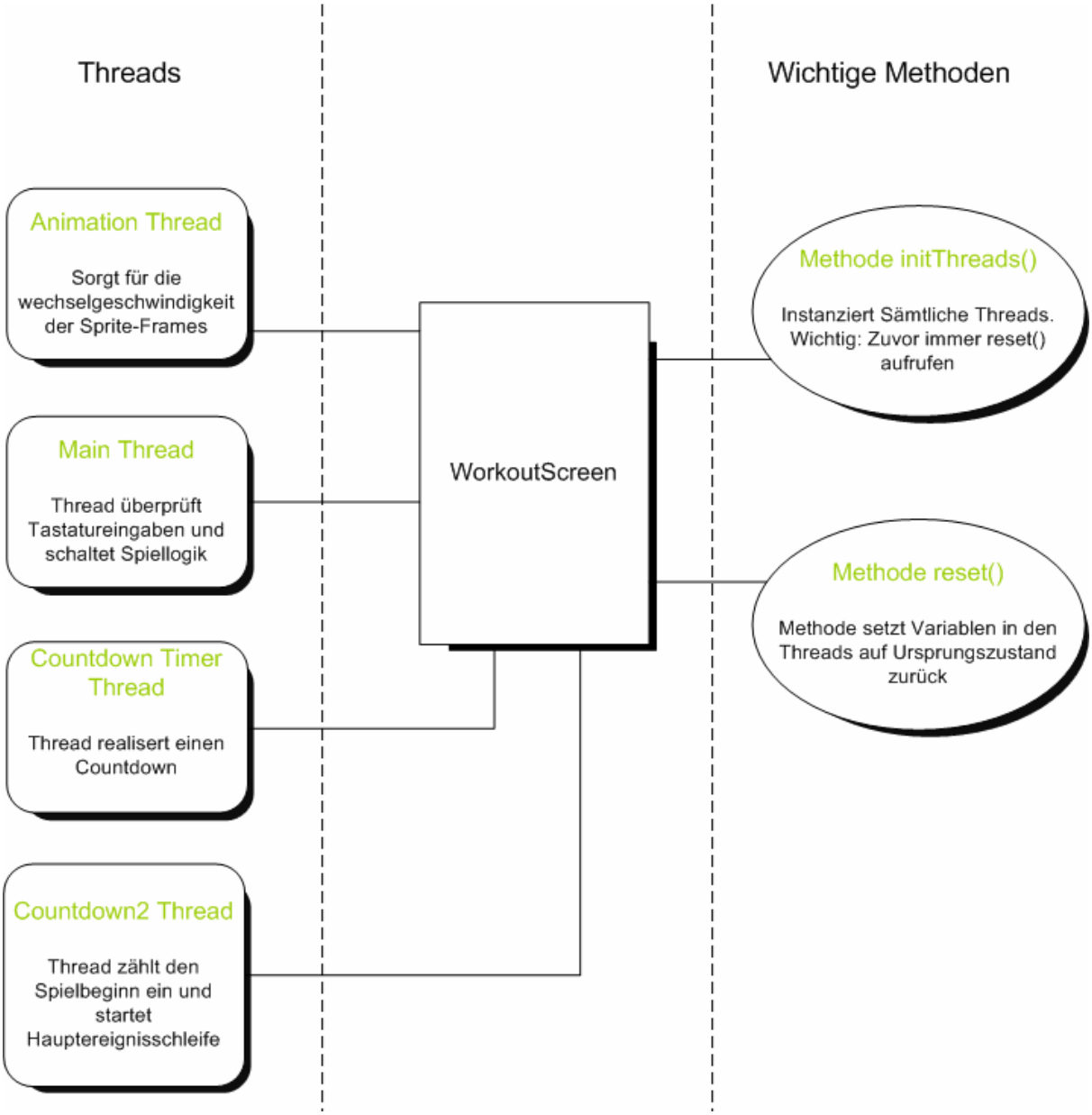

#### 4.7.1.2 .1 technische Realisierung

Der WorkoutScreen ist von der Klasse Gamecanvas abgeleitet und implementiert das Interface Runnable sowie CommandListener. Der Konstruktor lädt die Graphiken des Fisches und des Pfeiles in die Entsprechenden Variablen und erzeugt die dazugehörigen Sprites. Weiterhin werden die Commands zum starten des Trainings und zur Rückkehr zum Hauptmenü registriert. Anschließend wird der Bildschirm einmal mit der Methode render() gezeichnet um die Obeflächenelmente wie den Fisch usw. anzuzeigen.
Das Spiel läuft in einer Hauptereignisschleife ab, die Benutzereingaben überprüft und den Bildschirm entsprechend des gerade aktuellen Status zeichnet. Die Hauptereignisschleife läuft sobald der Hauptthread gestartet worden ist. Dies erfolgt nachdem der Menüpunkt „Trainieren“ vom Benutzer gewählt worden ist und die Funktionen initThreads() und reset() aufgerufen worden sind.

```
public void run() {
            while (isPlay) {
            Graphics g = getGraphics();
            if(state==PLAY)
                  handleInput();
            render(g);
            try {
                  Thread.sleep(200);
            } catch (Exception e) {
            }
      }
}
```

Hier kann man leicht erkennen, dass die Benutzereingaben nur dann verarbeitet werden wenn der aktuelle Zustand auf Play gesetzt ist. Die Funktion render() wird hingegen immer durchlaufen so lange das Spiel läuft. Diese Funktion ist für die graphische Darstellung des Trainings verantwortlich.
Bei J2Me Applikationen muss darauf geachtet werden, dass die Threads über ein boolean variable beendet werden kann, da ansonsten der Thread in einer Endlosschleife gefangen wäre und der meist knappe Speicher nicht freigegeben würde. Im konventionellen Java gibt es dafür die Methode isInteruppted(); die Überprüft ob der Thread von aussen eine Nachricht bekommen hat sich zu unterbrechen. Da es diese Methode im J2ME nicht gibt muss in der Hauptschleife eine boolean Variable irgendwann auf false gesetzt werden. Im obigen Code ist dies die Variable isPlay. Zum Thema Threads in J2ME Applikationen gibt es unter http://developers.sun.com/techtopics/mobility/midp/articles/threading2/ ein sehr gutes technisches Manual, das dem geneigten Leser dringend empfohlen wird.
Die Hauptereignisschleife wird wie oben gesagt durch den Countdown2 Thread ausgelöst. Dieser Wird wiederum dann gestartet, wenn der Spieler den Menüpunkt „Trainieren“ auslöst.

```
countdown2=new Thread(){
              public void run(){
                     for (int i=3;i>0;i--){
                            begingamesec=i;
                            try {
                                   Thread.sleep(1000);
                            } catch (Exception e) {
                            }
                     }
                            state=PLAY;
                            countdownTimer.start();
                            animationControl.start();
              }
};
```




Nachdem die for Schleife beendet ist,  wird der Status des Spieles auf PLAY gesetzt und somit die Tastatureingaben verarbeitet. Daraufhin werden die beiden Threads countdownTimer und animationControl gestartet. Diese werden im folgenden Diagramm kurz beschrieben.

Jetzt wird die Funktion solange der Countdown Timer nicht abgelaufen ist die Funktion handleInput() aufrufen. Diese überprüft ob die richtigen Tasten gedrückt worden sind und wie lang der letzte Tastendruck her ist.
Interessant ist die Funktion checkKey die die gedrückte Taste mit der zu drückenden Vergleicht. Wenn die Taste richtig war erhöht sich der Geschwindigkeitslevel und eine neue Pfeilrichtung wird generiert.
Wenn die Taste Falsch war wird der Geschwindigkeitslevel herabgesetzt

```
public void checkKey(int key) {
		if ((key == randNr)) {
			if (speed < 10)
				speed++;
			indicator=0x0000FF00;
			randNr = getRand(4);
			setVars();
		}
		else {
			if (speed > 0)
				speed--;
			indicator=0x00FF0000;
			setVars();
		}
	}
```

Anschließend werden die Variablen für die Durchschaltgeschwindigkeit der Sprites sowie die Zeitspanne wie lange der Benutzer Zeit hat die nächste taste zu drücken bevor die Geschwindigkeit herabgesetzt wird, aktualisiert. Nachdem der Countdowntimer abgelaufen ist wird der state von PLAY auf SCORE gesetzt und die Ergebnisse des Trainings werden angezeigt und das Fischobjekt wird mit seinen neuen Werten aktualisiert (updateFishStats()). Nun kann der Benutzer das Spiel von vorn starten oder die Trainingsumgebung verlassen. Dann wird die boolean Variable isPlay auf false gesetzt um den Hauptthread zu beenden, schließlich gelangt der User durch einen Callback aufruf von callback.mainMenuShow() zurück ins Hauptmenü.

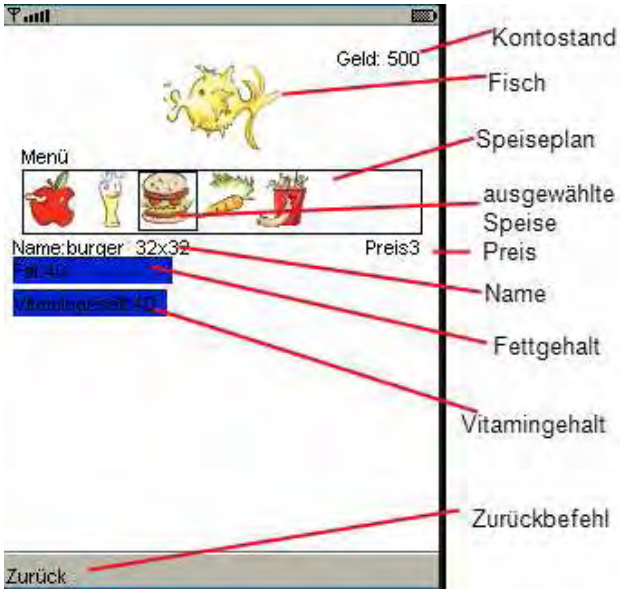
**Weitere Realisierungsmöglichkeiten für spätere Versionen:**
In der aktuellen Version wird dem Spieler für das trainieren bisher kein Geld abgezogen, was in späteren Versionen geändert werden sollte. Weiterhin wäre es interessant eine Möglichkeit einzubauen, dass das trainierende Fishobjekt die selbe Farbe und den Typ annimmt, wie der Benutzer es erstellt hat.

Die Klasse WorkoutScreen im Überblick:

### 4.6.2 Füttern (Andreas Bösselmann)


Um einen Fisch im Aquarium überlebensfähig machen zu können muss dieser mit der richtigen Nahrung aufgezogen werden. Dazu steht dem Züchter ein reichhaltiger Speiseplan aus vielen verschiedenen Nahrungselementen zur Verfügung. Diese Unterscheiden sich hinsichtlich der Portionsgrösse , des Fettgehalts, dem Vitamingehalt, des Preises. Natürlich wirkt sich der Fütterungsplan direkt auf den Lebenszyklus des Fisches aus, so dass man die Möglichkeit hat den Fisch Dick oder Dünn werden zu lassen, auch die Farbintensität des Fisches im Aquarium ist von diesen Faktoren abhängig und in späteren Version wird wohl noch so einiges hinzukommen.
Das Prinzip der Fütterung ist sehr einfach nachdem der Benutzer aus dem virtuellen Kühlschrank die zu Fütternde Speise ausgewählt hat, wird dessen Kontostand und die Anzahl der noch verfügbaren Tagesaktionen überprüft. Anschließend. Wird das Geld direkt vom Konto abgebucht und die graphische Repräsentation des Fisches aktualisiert.
Auch dieser Bildschirm ist von der WTK Klasse Canvas abgeleitet um einen hohen gestalterischen Freiraum zu gewährleisten. In der Ecke rechts oben wird dem Benutzer sein Kontostand angezeigt. Darunter ist die graphische Repräsentation des Fisches. Anschließend folgt das Menü mit den möglichen Speisen.

Mit den Cursortasten des Handys hat man die Möglichkeit den Fokus eines angewählten Nahrungobjekts zu verschieben. Das fokussierte Element wird mit einem Schwarzen Rahmen hervorgehoben. Darunter folgen die textuelle Beschreibung und Eigenschaften des fokussierten Elements.




### 4.7.2.1 technische Umsetzung

Hinter einem Nahrungsobjekt verbirgt sich immer ein Objekt der Klasse FoodItem. Dieses kapselt die Nahrungeigenschaften und stellt die Verbindung zur dazugehörigen Grafik mit Hilfe der Funktion getImage().
Im Konstruktor werden folglich zunächst die Speisen initialisiert. Der Konstruktor des Fooditems übernimmt in der Reihenfolge Werte für 1. Fettgehalt, Vitamine, Portionsgrösse, Kosten und den Namen. Momentan ist der Name der Speise gleich dem Namen der zugehörigen .png-Datei :

```
food=new Vector();
                //fat vitamine grösse kosten
                FoodItem f1=new FoodItem(32,49,29,5,"apfel_32x32");
                FoodItem f2=new FoodItem(10,30,10,11,"bier_32x32");
                FoodItem f3=new FoodItem(40,39,1,3,"burger_32x32");
                FoodItem f4=new FoodItem(20,80,4,10,"karotte_32x32");
                FoodItem f5=new FoodItem(40,4,20,6,"pommes_32x32");
                food.addElement(f1);
                food.addElement(f2);
                food.addElement(f3);
                food.addElement(f4);
                food.addElement(f5);
```

Anschließend werden diese in einer Vektor Datenstruktur gespeichert.
Da die Canvas die Methode keyPressed( int arg0 ) implementiert, die immer dann aufgerufen wird wenn eine Taste gedrückt worden ist, ist es leicht eine Ereignisbehandlungsroutine zu schreiben. Da bei einem Tastendruck nach links oder Rechts das Fokussierte Element zum nächsten springen soll muss eine Variable für das aktive Element vorhanden sein, die in der keyPressed(int arg0) Methode aktualisiert wird. Dazu wird der Code der gedrückten Taste mit:

```
int key = getGameAction(arg0);
```

In einen für den Vergleichsoperator verständlichen Integer konvertiert.
Dieser wird dann mit den Konstanten für die möglichen Tasten verglichen und die passende Aktion ausgeführt. Die Konstanten für die Tasten sind statische Variablen der Klasse Canvas. Wie z. B.: Canvas.LEFT für die Taste links oder Canvas.Fire für den Select Taste.

```
protected void keyPressed(int arg0) {
		int key=getGameAction(arg0);
		switch (key){
		//case Canvas.UP:repaint();break;
		//case Canvas.DOWN:break;
		case Canvas.LEFT:
			if (highlight>0)highlight--;
			repaint();
			break;
		case Canvas.RIGHT:
			if (highlight<food.size()-1)
				highlight++;
			repaint();
			break;
		case Canvas.FIRE:
			giveFood();
			repaint();
			break;
		}
	}
```

Anschließend muss der Bildschirm mit den aktualisierten Variablen neu gezeichnet werden. Bei einem Druck auf die Taste Links wird z.B. die Variable highlight – für das aktuell fokussierte Essensobjekt um 1 verringert. In der Paint Methode die durch repaint() aufgerufen wird, wird dann der Rahmen an der Stelle mit dem highlight Wert gezeichnet. Falls man das gewünschte Essen füttern möchte muss die Taste Select gedrückt werden. Dies führt dazu das bei der abfrage Canvas.FIRE die Methode giveFood() aufgerufen wird. Hier wird überprüft ob der Züchter genügend Geld hat und die Verfügbaren Aktionen nicht überschritten wurden und aktualisiert schließlich das der_fisch Objekt und speichert dies

wieder im RecordStore des Handys. Anschließend wird mittels repaint() der Bildschirm neugezeichnet.

Interessante Stellen der Paint Methode:
Die graphische Darstellung der Fooditems erfolgt durch das sukzessive durchgehen aller im Vektor vorhandenen Fooditems. Es muss darauf geachtet werden, dass nur so viele Items angezeigt werden, wie sie auch auf dem Bildschirm des Handys nebeneinander platz haben. Deshalb wird zunächst die Anzahl der anzeigbaren Elementer ermittelt indem die Gesamtbreite des Bildschirms (width) durch die Breite des FoodItems plus der Abstände zum nächsten Objekt geteilt wird (FOOD_IMG_WIDTH+10)

```
int anz=width/(FOOD_IMG_WIDTH+10);
             for (int i=0;i<anz;i++){
                    Image img=((FoodItem)food.elementAt(i)).getImage();
      g.drawImage(img,i*img.getWidth()+10,actualFish.getHeight()+marginTop+5,Graphi
cs.TOP|Graphics.LEFT);
```

Anschließend werden die Graphiken der FoodItems aus dem Vektor mithilfe der Funktion food.elementsAt(i) geladen und mit der Instanzmethode .getImage der Klasse FoodItem in eine temporäre Variable img gespeichert.
Mit der Methode drawImage kann die Graphik an eine bestimmte Stelle auf dem Bildschirm gezeichnet werden. Näheres dazu in der Dokumentation des WTK's.
Unter dem Speiseplan werden die Werte des gerade fokussierten Elements angezeigt. Um dem Benutzer die Auswahl etwas zu erleichtern werden die Werte dazu in Balkenform dargestellt. Z. B.: Je höher der Fettgehalt ist, desto breiter ist der Balken. Dies wird mit der Funktion fillrectRect(….) realisiert, die ein gefülltes Rechteck an eine bestimmte Position zeichnet. Zuvor wird noch die Farbe mit der gezeichnet werden soll mit der Funktion .setColor(…..) auf ein blau gesetzt. Diese Funktion nimmt RGB werte als Parameter entgegen, so dass ein schwarz mit setColor(0,0,0) und ein Weiss mit setColor(255,255,255) gewählt werden kann. Die Breite des Balkens wird dadurch ermittelt, dass der Fettwert des aktuellen FoodItems durch den Maximalwert(100) geteilt wird und anschliessend mit der gewünschten maximalen Balkenlänge multipliziert wird.

```
g.setColor(0,30,200);
      g.fillRect(5,POS_DESC+font.getHeight()+abstand,balkenlaenge*fatLaenge/100,fon
t.getHeight());
      g.fillRect(5,POS_DESC+2*(font.getHeight())+abstand+3,balkenlaenge*vitaminLaen
ge/100,font.getHeight());
```

Anregungen für nachfolgende Versionen:
- Anzeigemöglichkeit der aktuellen Fischwerte
- Wenn Fisch längere Zeit nicht gefüttert wurde Stimmung verändern (aggressiv , beleidigt)
- Wenn Fisch zu lange nichts gefüttert wurde evtl. Tod durch verhungern

Die Zusammenfassung der Klasse FeedFish:


## 4.8 Design-Optimierung des Menüs

### 4.8.1 CustomCanvas-Klassen (Dieter Hofbauer)

Nachdem J2MEpolish aufgrund der schlechten Integrierbarkeit ins Plugin-Konzept der Technikgruppe als Option wieder verworfen werden musste, suchten wir nach Alternativen. Also generierten wir ein eigenes ausbaubares und modulares Konzept, welches auf Basis der Canvas-Klasse höhere grafische Spielräume bieten sollte.

#### 4.8.1.2 Konzept

Die grundlegende Idee besteht darin, die Canvasgröße des aktuellen Ausgabegerätes zur Laufzeit des Midlets als Grundlage der Anordnung von dargestellten Elementen zu nehmen. Diese Elemente werden innerhalb einer logischen Gruppierungseinheit zusammengefasst, mit deren Hilfe einerseits die Anordnung auf dem Display und bei Bedarf auch die Funktionalität der Gruppeneinheit (zum Beispiel AnimatedMenuItem in einer MenuItemGroup) bewerkstelligt werden kann. So kann mit einer einzigen Version der Handyapplikation automatisch auf unterschiedliche Displaygrößen reagiert werden.

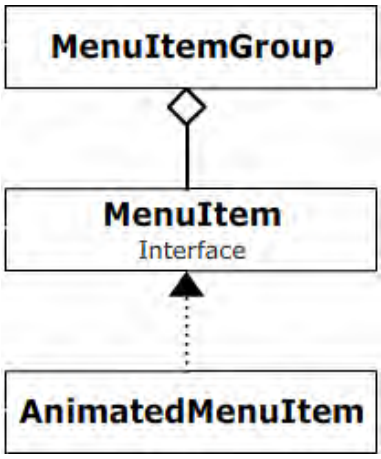
Die momentane Funktionalität ist ausschließlich auf die hier vorgestellte Anwendung "Fries4Fish" ausgerichtet. Wie bereits angesprochen ist die Planung des Systems stark auf die leichte Erweiterbarkeit durch weitere individualisierte Komponenten ausgelegt.

#### 4.8.1.3 Umsetzung

Wie bereits angedeutet, handelt es sich bei der aktuellen Version um einen Konzeptprototyp, der jedoch mit den bisher vorhandenen Klassen schon einen Ausblick auf weitere Ausbaumöglichkeiten gibt. Im Nachfolgenden sollen die bisherigen zwei realisierten Anordnungseinheiten kurz erläutert werden.

##### 4.8.1.3.1 MenuItemGroup

Die MenuItemGroup ist eine Einheit, die mehrere grafisch repräsentierte Items inhaltlich zusammenfassen soll und die Funktionalität der darin gespeicherten MenuItems bewerkstellt.




Momentan werden gemäß der Anforderungen von "Fries4Fish" nur AnimatedMenuItems in einer MenuItemGroup gruppiert. Ein AnimatedMenuItem ist dabei wiederum eine gruppierte Einheit aus einem oder mehreren Sprites, also animierbaren Images. Je nachdem, welcher Item in der MenuItemGroup gerade aktiv ist, wird innerhalb der Sprites eine Animation in Richtung der Repräsentation des aktiven oder inaktiven Bildes angeregt. Je mehr Einzelbilder das Sprite generieren, desto mehr erscheint das Menu animiert.

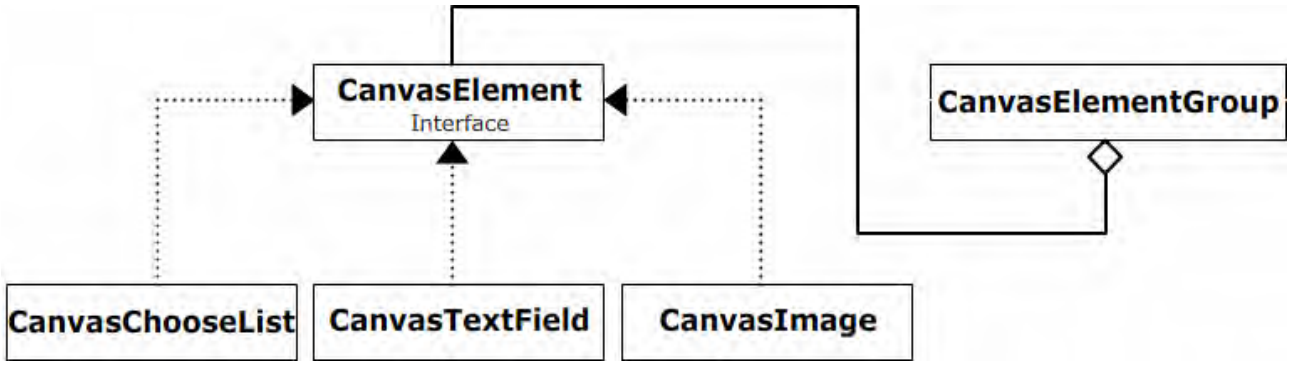
```
MenuItemGroup menu = new MenuItemGroup((Canvas)this);
```

```
menu.setDistanceY(10);

AnimatedMenuItem feedFish = new AnimatedMenuItem((Canvas)this, 5);
feedFish.setDistanceX(10);
feedFish.addElement("/publicScreen/client/fish/res/icon1.png");
feedFish.addElement("/publicScreen/client/fish/res/fuettern_bt.png");

AnimatedMenuItem trainFish = new AnimatedMenuItem((Canvas)this, 5);
trainFish.setDistanceX(10);
trainFish.addElement("/publicScreen/client/fish/res/icon1.png");
trainFish.addElement("/publicScreen/client/fish/res/training_bt.png");

menu.setMenuItem(sendFish);
menu.setMenuItem(trainFish);

feedFish.start();
trainFish.start();

menu.setActiveStartItem (0);
```

Mittels der oben beschriebenen Aufrufe wird zuerst eine neue MenuItemGroup erzeugt und der Standardabstand zwischen den Elementen der Gruppe auf 10 Pixel eingestellt. Danach werden zwei Instanzen von AnimatedMenuItem erzeugt und auch hier der Abstand zwischen den Elementen des AnimatedMenuItems auf 10 eingestellt. Dann werden jedem der AnimatedMenuItems zwei Images oder vielmehr die URL von diesen übergeben. Diese Images werden intern zu Sprites einer bestimmten Animationslänge (hier 5) umgewandelt und können so animiert werden. Schließlich werden die beiden Items noch der ItemMenuGroup untergeordnet, die die Animation abrufenden internen Threads gestartet und der aktive StartItem festgelegt (optional). Mit den Methoden setNextSelected und setBeforeSelected kann nun der gerade aktive Item beeinflusst werden. Dieser ist in der MenuItemGroup als Index abgespeichert und kann seitens der Anwendung für die weitere Eventbehandlung abgerufen werden.

#### 4.8.1.3.2 CanvasElementGroup

Auf demselben Prinzip wie die MenuItemGroup baut auch die CanvasElementGroup auf, allerdings stellt sie keine funktionale Einheit in Bezug auf die darin abgelegten Elemente dar, sondern bewerkstelligt nur deren zusammengehörige Anordnung auf dem Handydisplay.
In eine CanvasElementGroup können Elemente abgelegt werden, die das Interface CanvasElement implementieren. Momentan sind dies CanvasImage, CanvasChooseList und CanvasTextField.




Ein CanvasImage ist eine spezielle Repräsentation eines Images, welches eine Integration in eine CanvasElement ermöglicht. Eine CanvasChooseList ist eine unanimierte, textbasierte Auswahlliste und schließlich ein CanvasTextField repräsentiert ein Textfeld, welches Text auf dem Display in einem bestimmten Ordnungsstil anordnet (momentan nur zentriert).

```
CanvasElementGroup chooseFishElementGroup = new CanvasElementGroup(this);
chooseFishElementGroup.setDistanceY(5);

Font font = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_PLAIN,Font.SIZE_MEDIUM);
Font font3 = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD,Font.SIZE_LARGE);

Vector testList = new Vector();
testList.addElement("testelement 1");
testList.addElement("testelement 2");

CanvasImage icon  = new CanvasImage("/publicScreen/client/fish/
               res/raubfisch_auswahl_32x32.png", this);

CanvasTextField textField = new CanvasTextField("blindtext", font, this);
textField.setColor(0, 0, 0);

CanvasChooseList list = new CanvasChooseList (testList, font3, this);

chooseFishElementGroup.add(icon);
chooseFishElementGroup.add(textField);
chooseFishElementGroup.add(list);
```

Zunächst wird eine neue CanvasElementGroup erzeugt. Danach werden zwei Fonts und eine Vectorliste generiert, die für die nachfolgenden Operationen benötigt werden. Nacheinander werden nun ein CanvasIcon, ein CanvasTextfield und eine CanvasChooseList generiert, die abschließend der CanvasElementGroup zugeordnet werden.

## 5. Die Schnittstelle Handyapplikation – Server (Dieter Hofbauer)

### 5.1 Basisarbeit

Für eine Kommunikation zwischen Java und Flash bemühte ich als Informationsquelle hauptsächlich das Internet. Meine Recherche dort ergab eine relativ große Anzahl an Kommunikationswegen von Flash nach außen. Will man mit Flash von extern kommunizieren, werden die Informationen schon dürftiger. Ein möglicher Kommunikationsweg wäre eine Anbindung über das in der Version 8 von Flash neu eingeführte Konzept des "External-Interface" gewesen. Hier wird ein Anbindung an eine Flashanwendung über den Umweg Javascript geschaffen, mit dessen Hilfe man Funktionsaufrufe in Flash nachbilden kann. Mit einem in Java integrierten Flashplayer wäre auch ein Aufruf von setVariable() in Flash möglich gewesen. Dabei hätte man dann einen Listener implementieren müssen, der die jeweilige von außen veränderte Variable (z.B. ein XML-String, der die nötigen Informationen enthält) auf Veränderungen überwacht und im Falle eines neuen Zustandes die neue Information in einen Informationsaufruf umwandelt. Beide beispielhaften Wege schienen nicht besonders ausgereift, umständlich und wurden stets als Notlösung gesehen.

#### 5.1.1 Proprietäre Client-Server-Lösungen

Es gibt jedoch diverse kommerzielle Client-Server-Lösungen die eine Kommunikation von Flash nach Java zulassen (z. B. ElectroServer, Unity2, Offbeat). Aus diesen Lösungen möchte ich im Besonderen den Offbeat-Server (http://www.beamjive.com/offbeat.php) hervorheben, der nach Studium diverser Bedienungs- und Installationsanleitungen relativ leicht zu bedienen und an persönliche Bedürfnisse anzupassen schien.
Eine proprietäre Lösung wurde aber von Anfang an nicht favorisiert, da eine kostenneutrale Lösung angestrebt wurde.

#### 5.1.2 XML-Socket

Der Wendepunkt bei der Suche stellte sich erst bei der Eruierung der XML-Socket-Lösung ein. Hier werden auf dem Wege einer Socketverbindung, die auf Ports oberhalb 1024 stattfinden kann und die natürlich auch über Java angebunden werden kann, Daten an Flash übertragen. Im Prinzip wird ein langer String übertragen, der die XML-Information enthält. Dieser String kann selbstverständlich auch die für einen Funktionsaufruf in Flash nötigen Daten enthalten (Funktionsname, Objektname, Parameter).

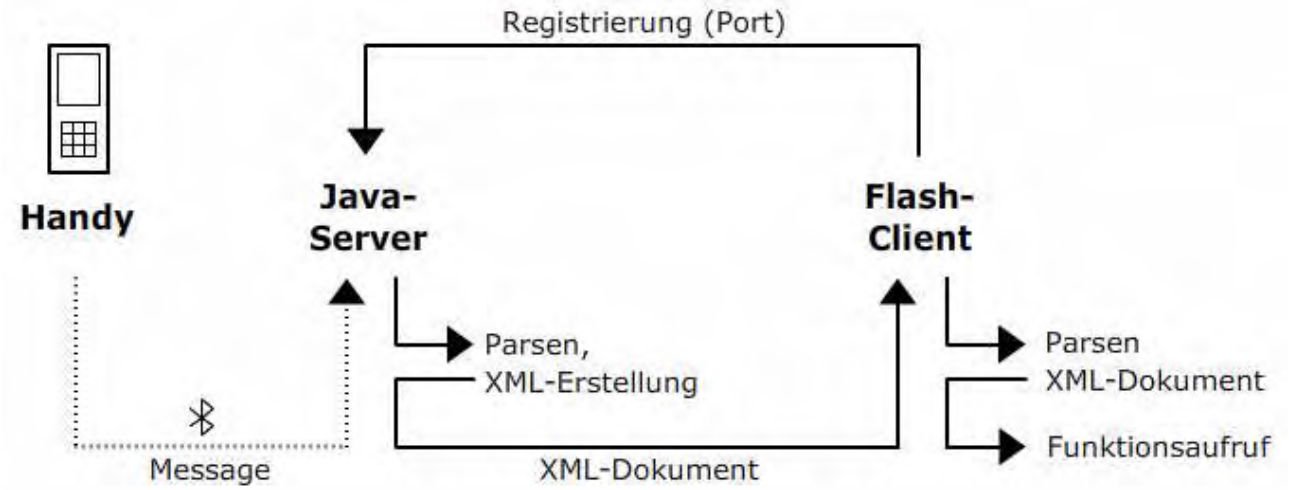
### 5.1.3 Flashplayer für Java

Auch für die Integration einer swf-Datei in eine Java-Applikation gibt es sowohl kommerzielle als auch Open-Source-Lösungen. Die hier zum Einsatz kommende Variante gehört komplett zum kostenneutralen Bereich und integriert einen Java-Flash-Player [DV06] in eine SWT-Umgebung. SWT (Standard Windows Toolkit) ist ein GUI-Framework, welches betriebssystemeigene Features verwenden kann. In diesem Fall ist es ein über ActiveX angesprochener FlashPlayer, der auf dem ausführenden System bereits installiert sein muss.

## 5.2 Umsetzung

### 5.2.1 Grundlegendes Funktionsprinzip und Umsetzung der Kommunikation

Damit eine Kommunikation überhaupt stattfinden kann, muss eine Kommunikationsverbindung zwischen der Flashapplikation – im Nachfolgenden Flash-Client genannt – und der Java-Anwendung – im Nachfolgenden Java-Server genannt – aufgebaut werden.



#### 5.2.1.1 Flash-Client

Wie schon zuvor erläutert wird auf der Flashseite eine Verbindung mittels eines sogenannten XMLSockets aufgebaut.

Die Flash-Seite ist dabei so gegliedert, daß in eine Flash-Container-Datei dynamisch die drei swf-Dateien der jeweiligen Gruppen geladen werden. Jede dieser Dateien baut intern autark eine Verbindung zum Server auf und agiert unabhängig vom Container.
Die Einbindung in die jeweilige Flash-Anwendung sollte sich möglichst einfach vollziehen und passiert nur durch eine einzige Zeile im ActionScript-Teil des ersten Schlüsselbildes der Zeitleiste:

```
var  flashClientFish : JavaFlashClient = new JavaFlashClient("#Fish", "flashClientFish");
```

Sie erzeugt eine neue Instanz von JavaFlashClient, deren Konstruktor als ersten Parameter eine ID (dieselbe wie auf der Javaseite) und als zweiten Parameter den Namen der eben erzeugten Instanz des JavaFlashClienten übergeben wird.
Beim Starten des swf-Films wird automatisch eine Verbindungsanfrage auf einem bestimmten Port gestellt (hier 9999). Nach dem Prinzip von EventListenern stehen einer XMLSocket-Verbindung verschiedene Events zur Verfügung, auf die sie reagiern kann:

```
socket.onConnect = function(success) { ... }
socket.onClose = function() { ... }
socket.onXML = function (xml) { ... }
```

Das für die Kommunikation mit Java entscheidende Event ist "onXML". Bei Eingang einer XML-Datei an der aufgebauten XMLSocket-Verbindung wird die XML-Datei als String an die dem Event zugeordnete Funktion übergeben, die den "XML-String" weiterverarbeitet.
Da die oben genannten Events in einer init-Funktion seitens des Konstruktors gestartet werden, können sie in Flash prinzipbedingt nicht mit "this" auf die Variablen der Klasse, in der sie erzeugt wurden, zugreifen. Mit der Übergabe des Instanznamens an den Konstruktor kann der Name der Instanz an die init-Funktion übergeben werden. Mit dem Instanznamen können die "EventListener" dann auch auf die Attribute der Klasse JavaFlashClient zugreifen.
Wird nun ein Funktionsaufruf in Form einer XML-Datei an das XMLSocket übertragen, so hat die eingehende XML-Datei folgendes Format:

```
<?xml version="1.0" encoding="UTF-8" ?>
<flash>
  <functionCall>
    <object val = "objectName">
    </object>
```

```
    <name val = "functionName">
    </name>
    <parameter val1 = "parameter1" val1 = "parameter2">
    </parameter>
  </functionCall>
  <identification val = "#Fish1234567890">
  </identification>
</flash>
```

Diese Datei benützt kein DTD und liegt in Flash als ein großer String vor, aus dem die einzelnen Inhalte ausgelesen werden können. Mit den nun im Flash-Client vorliegenden Daten kann der Funktionsaufruf gestartet werden.

#### 5.2.1.2 Java-Server

Auf der Javaseite wird ein Konstrukt zur Verfügung gestellt, welches eingehende Socketverbindungen annimmt und verwaltet. Dieser als eigenständiger Thread laufende FlashManager nimmt mithilfe eines ServerSocket, das auf einem festen Port (hier 9999, Ports ab 1024 aufwärts können verwendet werden) eingehende Verbindungsanfragen abwartet, nur Verbindungen dergestalt entgegen, daß sie beim Verbindungsaufbau ihre ID mitteilen, welche serverseitig registriert sein muss. Diese Registrierung erfolgt gemäß den Gepflogenheiten der Technikgruppe in der jeweiligen ServerComponentIDList im Shared-Paket jeder Gruppe:

```
public static final String FISH_FLASH_ID = "#Fish";
```

Diesen Beschränkungen entsprechendende Socketverbindungen werden als Instanz der Klasse FlashClient in einer internen HashMap über ihre ID verwaltet und abgespeichert. Dabei können Socketverbindungen natürlich in der Theorie auch von nicht Flash-Clienten angefragt werden, für diese Anwendung sollen es allerdings ausschliesslich auf swf-Dateien basierende Verbindungen sein.
Die eigentliche Kommunikation zum Flash-Clienten erfolgt in den einzelnen Server-Plugins. Eine Instanz der Klasse "FlashFunctionCaller", die bei der Initialisierung des jeweiligen Plugins erzeugt wird, bewältigt dabei die Kommunikation mit dem dem Server-Plugin zugehörigen Flash-Client.

```
private FlashFunctionCaller caller;
caller = Settings.getFlashManager().setFFC(ServerFishComponentIDList.FISH_FLASH_ID);
```

Prinzipiell wäre geplant gewesen, in dieser Instanz des FlashFunctionCaller bereits die gekapselte Instanz des dem Plugin anhand der ID zugeordneten FlashClient abzuspeichern,

um eine schnelle Kommunikation zu gewährleisten. Doch genau dort lag zu diesem Zeitpunkt das Problem: das SWT-Fenster und damit die zugehörigen Flash-Dateien werden erst sehr spät geöffnet und die Verbindungsanfragen von der Flash-Seite gehen erst nach dem init()-Aufruf und sogar nach dem run()-Aufruf des jeweiligen Plugin ein. Somit kann die Instanz des FlashClient nicht abgespeichert werden, weil es sie zu diesem Zeitpunkt noch nicht gibt. Somit ist temporär der Zugriff auf den jeweiligen FlashClienten nur über den Umweg des Zugriffs über die HashMap des FlashManagers möglich. Eine abschliessende Klärung zum Zeitpunkt der Einbindung ins Techniksystem sollte dieses Problem aus der Welt schaffen. Verabredungsgemäß sollte zunächst das gesamte Java-Flash-Kommunikations-System fertig sein, bevor die abschliessende Integration in die Technik-Infrastruktur angegangen werden sollte. Dennoch war mein Bestreben, so wenig wie möglich Anpassungsarbeit im Nachhinein, insbesondere durch die Technikgruppe, leisten zu müssen. Doch dazu später mehr.
Will nun ein Plugin eine Funktion in Flash aufrufen, braucht die zugehörige Funktion zumindestens einen Parameter, nämlich den Funktionsnamen in Flash in Form eines Strings. Soll zusätzlich noch der Name des Objekts festgelegt werden, muss ein weiterer String übergeben werden. Standardmässig ist "this" als Objekt der Zeitleiste festgelegt. Um der Funktion Parameter zu übergeben, muss zuerst eine Instanz von ParameterList erzeugt werden. Dieser können dann in der Reihenfolge des Auftretens in der Flashfunktion Parameter übergeben werden.

```
ParameterList parameter = new ParameterList();
parameter.setParameter("parameter1");
caller.callFunction("objectName", "functionName", parameter);
```

Mithilfe des Pakets jDom wird aus diesen Daten eine XML-Datei generiert, die mithilfe des FlashFunctionCallers über die serverseitig abgespeicherte Socketverbindung den Funktionsaufruf zur Flashseite übertragen kann. Die mitübertragene ID der Nachricht, die aus der ID der Verbindung und der aktuellen Zeit generiert wird, wird wieder zurückübertragen und auf der Javaseite mit der Ursprungs-ID verglichen. Stimmen beide überein, gibt die Methode "callFunction" true zurück und belegt so den Erfolg der Übertragung.

### 5.2.2 Integration in die Infrastruktur der Technikgruppe

Von Anfang an galt es als essentiell für alle Arbeiten im Bereich Java-Flash-Kommunikation, die Integration in das System der Technikgruppe miteinzuplanen. Nach eingehenden und erfolgreichen Tests mit drei unterschiedlichen Plugins und den drei swf-Dateien schien alles bereit für eine Integration. Sie sollte im Wesentlichen basierend auf den beschriebenen

Schritten und der Installation eines weiteren Pakets (publicScreenServerFlashCommunication) bestehen, welches alle nötigen Klassen und Dateien in Bezug auf die Java-Flash-Kommunikation beinhaltete.
Seitens der Technikgruppe wurde die Integration dann doch anders, sprich tiefergehend vollzogen. Zeitgleich tauchten nun Probleme auf, wie sie in den Tests vorher nicht aufgetreten waren. Beispielsweise wurden nun Funktionsaufrufe im falschen Film gestartet, ein Verhalten, das nur durch die falsche Zuordnung von Sockets entstehen kann und vor der Integration in den Tests unter ähnlichen Bedingungen nicht vorzufinden war. Scheinbar teilten sich nun die drei swf-Dateien in der Container-Datei eine XMLSocketverbindung, eine Beobachtung, wie sie zuvor nicht aufgetreten war und auch jetzt nicht. Die Technikgruppe sah sich daraufhin veranlasst, das Konzept der Java-Flash-Kommunikation ihrerseits und ohne meine Mitwirkung grundlegend zu ändern und zu verbessern, was ihr in zahlreichen Punkten sicher auch gelungen ist.
Eine Hauptänderung bestand darin, den Java-Server nicht nur einen festen Port nach Verbindungen abhören zu lassen, sondern die den jeweiligen Plugins zugehörigen, jetzt unterschiedlichen Ports. Somit war das Problem der falsch zugeordneten Verbindungen beseitigt. Außerdem wurde die XML-Struktur verändert und bei der Kommunikation die Rückantwortfunktion als Zeichen der positiven Übertragung gestrichen.
Aufgrund der Neuausrichtung der Flash-Kommunikation wurde die bisherige Lösung als "deprecated" deklariert und nicht weiterentwickelt. Ausstehende Probleme und TODOs blieben ungelöst oder bestehen.

## 6. Serverseitige Implementierung mit Flash (Alena Schneider)

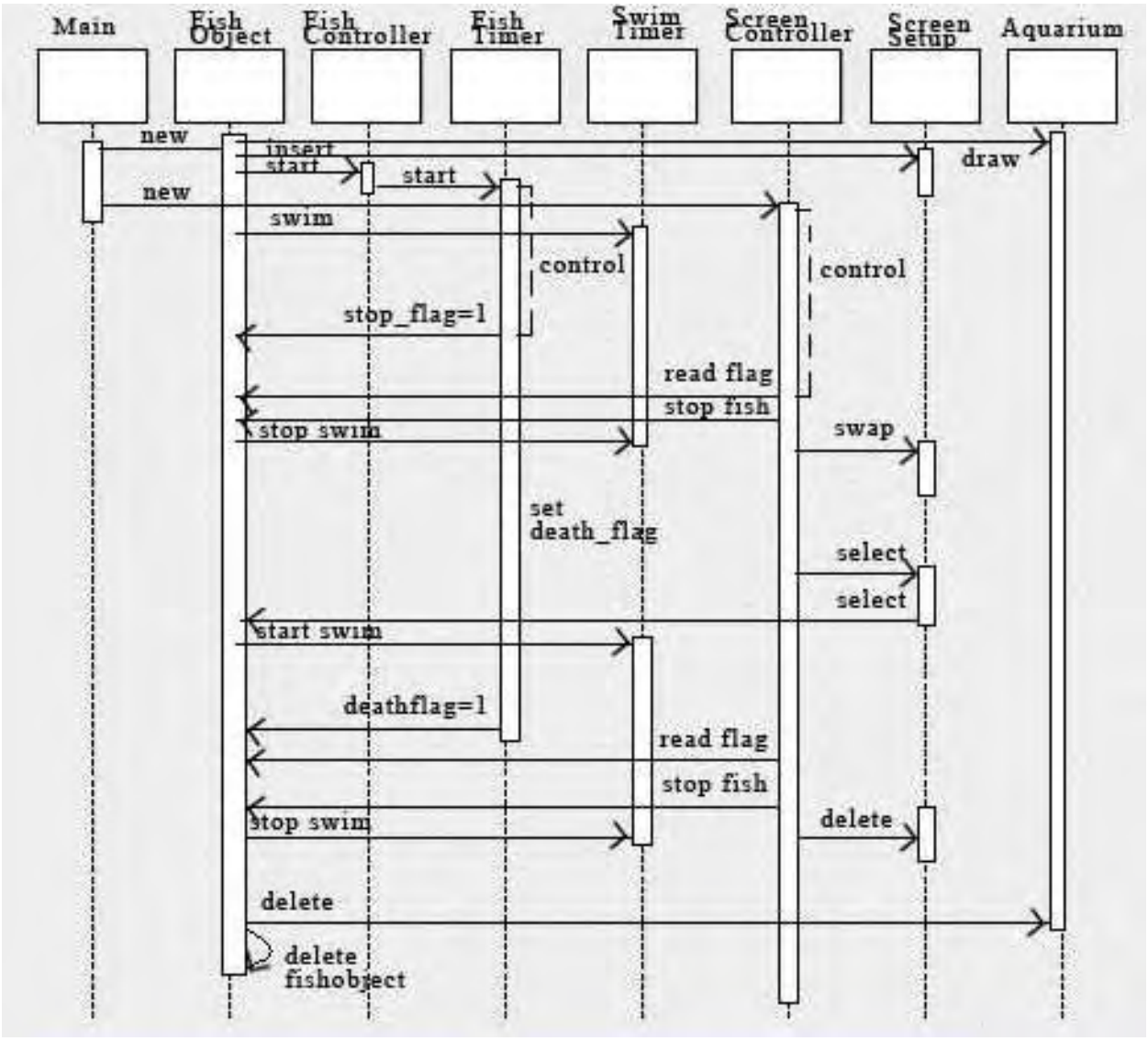
### 6.1 Idee und Sequenzdiagramm

Idee:

- Aquarium in Flash (Auflösung: 944 px  x 355 )
- am Screen wird nur ein Teilbereich des Aquariums dargestellt
- Übermittlung der Fisch-Parameter über eine Flash - Java-Schnittstelle
- die Parameter werden als String übertragen und für Flash aufbereitet
- Flash bekommt Parameter (name, id, farbe ...) und erzeugt einen Fisch im Aquarium
- andere Fischparameter (scale, muster …) werden errechnet
- im Aquarium existieren unterschiedliche Fischarten
- diese zeichnen sich durch unterschiedliche Verhaltensmuster und Aussehen aus
- die Lebensdauer der Fische ist begrenzt
- der Fisch entwickelt sich im Aquarium weiter
- es gibt Fütterungszeiten

- Interaktion , Fisch – Benutzer

Sequenzdiagramm:



## 6.2 Die wichtigsten Klassen und Funktionen im Überblick

### 6.2.1 FishObject

**Beschreibung (FishObject – Klasse):**

FishObject ist die Oberklasse und hat 3 Unterklassen:

- Kugelfish
- Knochenfish
- Raubfish

**Attribute (FishObject – Klasse):**

Parameter (z.B. Fischname, Fischart, Farbe …) werden als String über die Flash - Java - Schnittstelle übermittelt und ein Fisch - Objekt erzeugt.
Andere Parameter (z.B. Geschwindigkeit, Farblichkeit usw.) werden intern berechnet.
Ein Fisch - Objekt setzt sich aus 3 MovieClips zusammen:

*private var mc:String;* //Fischgrafik
*private var mc2 :String ;* //Name
*private var mc3 :String ;* //Begrüßung

Pro Fisch - Objekt werden jeweils 2 Timer gestartet:

- Swim-Timer
- FishController-Timer

Es werden Farbanteile der Matrix für rot, grün und blau errechnet:

- *rgb_r, rgb_g, rgb_b*

Außerdem werden das maximale Fischalter und die mittlere Geschwindigkeit gesetzt.
Fische werden mit Hilfe von ActionScript eingefärbt (Zufallsmuster, abhängig von Fischparametern). Alle Farbfilter werden in einem Array (FarbFilterListe) festgehalten, die auf den Fisch angewandt werden sollen. Anschließend wird activate_filter() aufgerufen, was dazu führt dass der Fisch eingefärbt wird ( mit oder ohne Muster ).
Jeder Fisch hat 2 Flags, die vom FishController-Timer gesetzt werden:

- *death_flag : Boolean;*
- *stop_flag : Boolean;*

**abstrakte Funktionen ( FishObject – Klasse ):**

- *get_y_draw()* //wird definiert wo d. Fisch im Aquarium platziert wird
- *swimKI()* //Schwimmverhalten des Fisches
- *update_colour()* //wird definiert welche Farben und Farbmuster der Fisch annimmt

**die wichtigsten Funktionen (public, FishObject – Klasse ):**

*swimFish(id:Number)*

- Swim-Timer starten
- ScreenSetup.fish_start_swim(this,id) aufrufen
- FishController-Timer starten

*stopFish(id:Number)* , sobald Fisch im unsichtbaren Bereich

- Swim-Timer anhalten
- ScreenSetup.fish_stop_swim(this,id) aufrufen
- FishController-Timer anhalten

*updateFish()*

- Fischgröße anpassen
- Farbmuster anpassen
- Geschwindigkeit anpassen

*drawFish()*

- Y-Position bestimmen
- Fisch zeichnen

*remove_fish()*

- Swim-Timer beenden
- Grafik löschen
- FishController-Timer beenden
- ScreenSetup.fish_stop_swim(this,id) aufrufen
- ScreenSetup.deleteFish() aufrufen

*fish_hallo()*

- Benutzer wird begrüßt

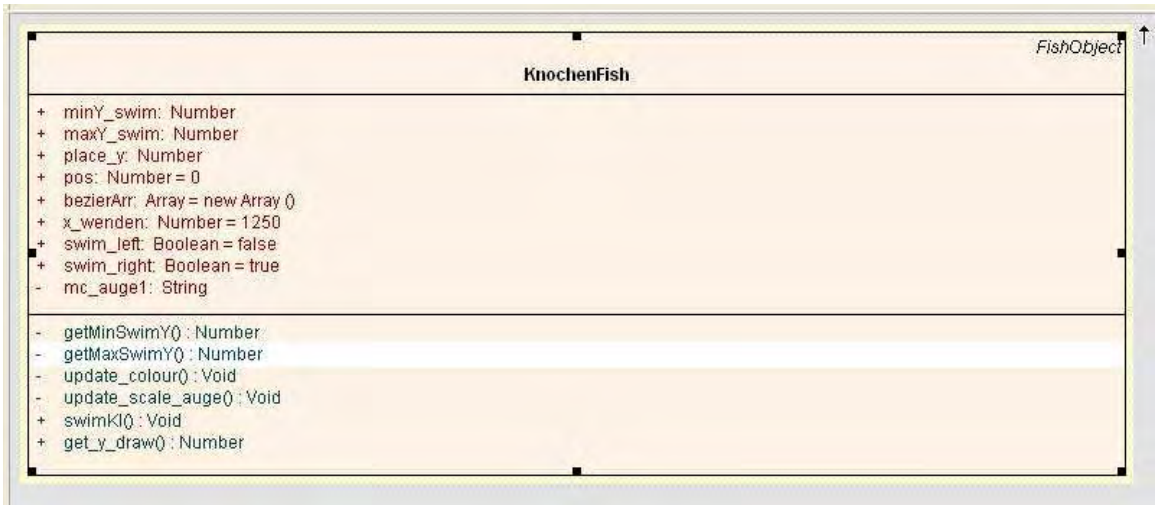
**weitere Funktionen ( FishObject – Klasse ):**

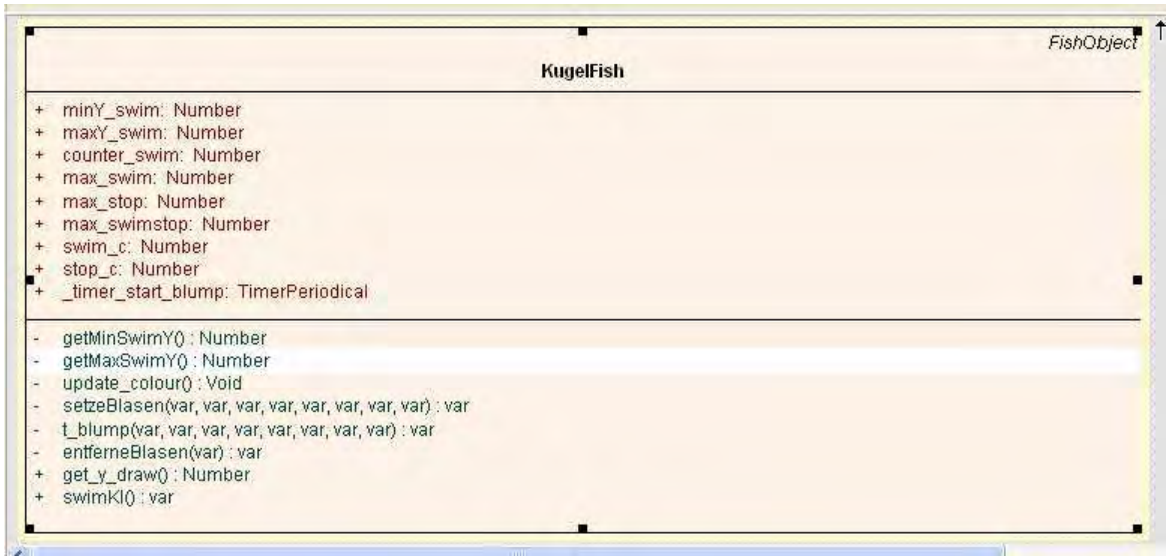
*randRange(min:Number,max:Number) :Number* //Zufallszahlen ermitteln
*update_scale ()* //skalieren
*update_speed()* //Geschwindigkeit anpassen
*farbe_ohne_filter()* //nicht mit anderen Filtern kombinierbar
*farbe_matrix_filter(r,g,b,alpha)* //Filter
*farbe_bevel_filter(…..)* //Filter
*activate_filter()* //nachdem man alle Filter zusammengefügt hat, aktivieren

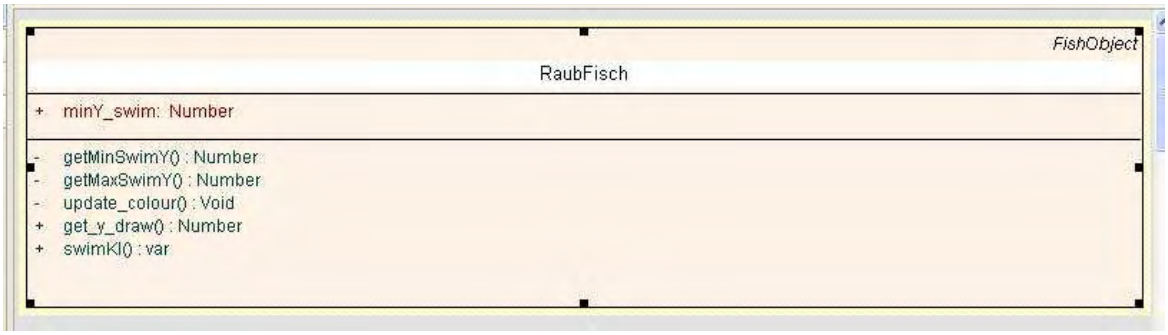
Jeder Fisch wird der Liste hinzugefügt (*ScreenSetup.insertFish(this)*), dient zur Verwaltung der schwimmenden und der nicht schwimmenden Fische (kommt später!).
Bei der grafischen Umsetzung ist es wichtig zu beachten, dass Vektorgrafiken verwendet werden, sonst ist die Farbanpassung der Fische nicht möglich (Farbfilter)!
Bei diesem Prototyp habe ich testweise 3 Fischarten vektorisiert und teilweise animiert.

### 6.2.2 KnochenFish-, Kugelfish- und RaubFish-Klasse

abgeleitete Klassen von FishObject






**Funktionen:**
swimKI() //wird definiert wo d. Fisch gezeichnet wird
get_y_draw() //Schwimmverhalten des Fisches
update_colour() //wird definiert welche Farben und Farbmuster der Fisch annimmt

**Merkmale : Knochenfisch**
swimKI(): es werden unterschiedliche Bezierkurven vorausberechnet. Aus diesen kleinen Kurven wird per random() eine Schwimmkurve des Fisches zusammengesetzt.
Das Wendeverhalten des Fisches wurde hier ebenfalls implementiert.

get_y_draw() : schwimmt im oberen 2/3 des Aquariums
update_colour() : kann unterschiedlichste Muster annehmen, hängt vom Vitamingehalt ab, hier Kombination aus 2 Filter

**Merkmale : Kugelfisch**
*swimKI()* : gleichmäßige Schwimmbewegung. Der Fisch schwimmt x Sekunden und kommt dann für z Sekunden zum Stehen. In der Zeit wo der Fisch steht werden Luftblasen erzeugt ( hier ebenfalls mit ActionScript implementiert)

*private function setzeBlasen(obj, xPos, yPos, anzahl, zHoehe, zBreite, Hoehe, Breite),*
*private function t_blump(obj, xPos, yPos, anzahl, zHoehe, zBreite, Hoehe, Breite):*
Luftblasen erzeugen

get_y_draw(): schwimmt im unteren 2/3 des Aquariums
update_colour(): kann unterschiedlichste Muster annehmen, hängt vom Vitamingehalt ab, hier Kombination aus 2 Filter

**Merkmale : Raubfisch**
swimKI() : schnelle, gleichförmige Bewegung

get_y_draw()  : schwimmt im oberen 2/3 des Aquariums

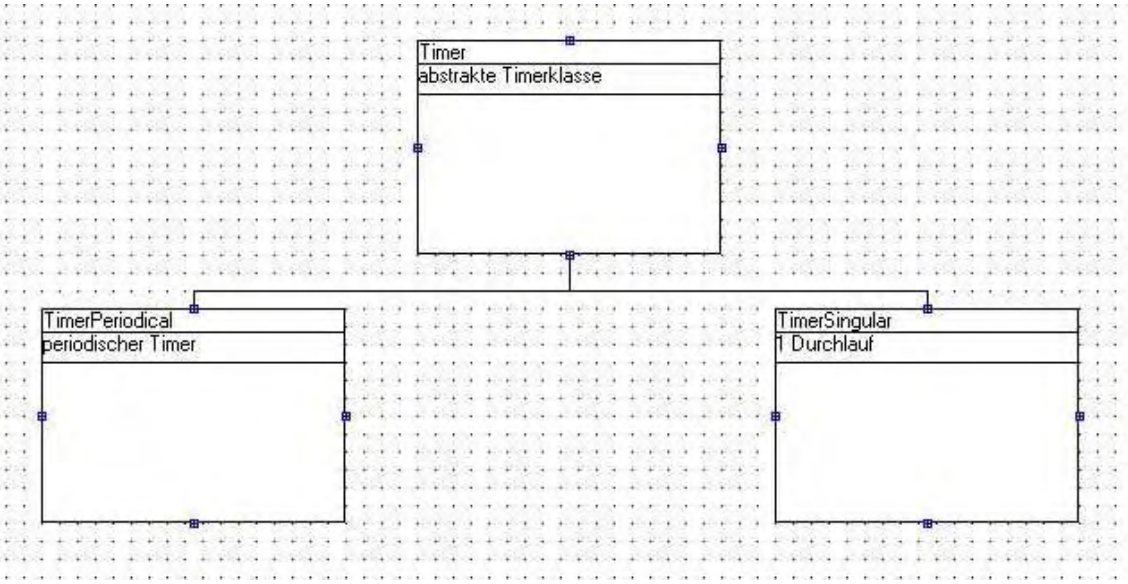
update_colour()  : kann keine Muster annehmen, einfarbig

### 6.2.3 alle Timer im Überblick

#### 6.2.3.1 Basisfunktionalitäten

bieten 3 Klassen**:**

- Timer
- TimerSingular
- TimerPeriodical




**Timer-Klasse :**

**Beschreibung  (Timer-Klasse):**

bietet zusammen mit den Unterklassen eine schnelle und einfache Möglichkeit einen Timer zu definieren.

**Attribute (Timer-Klasse):**

*var v_duration:Number* //Intervall

*var v_receiver:Object* //Instanz, wo die Methode definiert ist, die ausgeführt werden soll

*var v_methodname:String* //Methodenname von der Methode, die ausgeführt wird

*var v_arguments:Array* //Methoden-Argumente

**abstrakte Funktion (Timer-Klasse):**

*private var doAction: Function;*

Konstruktor: *public function Timer ( v_dur: Number, v_receiver:Object,*
*v_methodname:String, v_arguments:Array )*

**Methoden (Timer-Klasse):**
start() //Timer starten
stop() //Timer stoppen
start(): *v_interval = setInterval(this, „doAction“, v_duration);*
stop(): *clearInterval(v_interval);*

**TimerPeriodical** ( abgeleitete Klasse von der Timer-Klasse):

**Beschreibung (TimerPeriodical):** ein periodischer Timer
doAction() //Methode der übergebenen Instanz soll hier aufgerufen werden
v_receiver[v_methodname].apply(v_receiver, v_arguments);

**TimerSingular**( abgeleitete Klasse von der Timer-Klasse):

**Beschreibung (TimerSingular):** einmaliger Timer, wird nur 1 mal ausgeführt
doAction() //Methode der übergebenen Instanz soll hier aufgerufen werden
Timer wird gestoppt: *this.stop();*

### 6.2.3.2 Timer in der FishObject – Klasse

werden für jeden Fisch gestartet:

- Swim-Timer
- FishController-Timer

**Swim-Timer:**
*_timer_startSwim = new TimerPeriodical(speed, this, "swimKI" , []);* //Schwimmverhalten
*swimKI()* ist eine abstrakte Funktion und wird in den Unterklassen (Kugelfish, Knochenfish, Raubfish ) implementiert

**FishController-Timer** benötigt die **FishController-Klasse** und die **Fish-Timer-Klasse**:

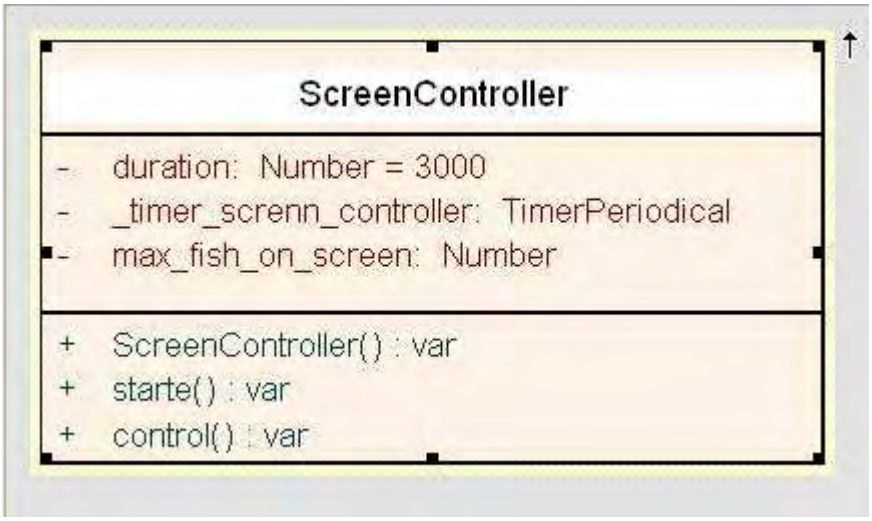


**Beschreibung (FishController-Timer):**

- Alter der Fische wird verändert
- Flags werden gesetzt (stop-Flag, death-Flag)

### 6.2.3.3 ScreenController-Timer

benötigt die **ScreenController-Klasse**:




**Beschreibung (ScreenController-Timer):**

- falls zu wenig Fische schwimmen, neue erlauben ( jedoch max. 10 gleichzeitig)
- Flags prüfen, nur von den schwimmenden Fischen (stop-Flag, death-Flag), bei Flagänderung reagieren

### 6.2.4 ScreenÜberwachung

**ScreenSetup-Klasse (statische Klasse)**

**Beschreibung (ScreenSetup-Klasse):**

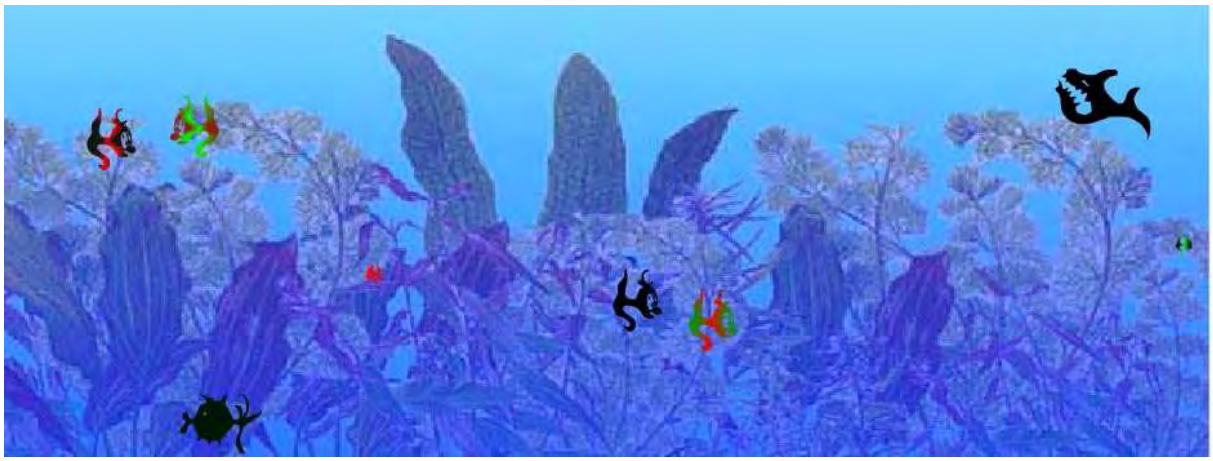
ScreenSetupKlasse ist eine statische Klasse, d.h. man braucht keine Instanz. Der Aufruf der Funktionen erfolgt über ScreenSetup.Funktionsname(Parameter);
Die Klasse stellt Funktionen bereit, die zur Screenüberwachung notwendig sind.
Diese Funktionen werden hauptsächlich vom ScreenController aber auch von anderen Klassen verwendet.

**Attribute (ScreenSetup-Klasse):**

var fishList : Array //nicht schwimmende Fische im Aquarium
var fishList_swimming : Array // schwimmende Fische im Aquarium

**Funktionen (ScreenSetup-Klasse):**

insertFish(fish:FishObject) // neuen Fisch -> FischListe
deleteFish(fish:FishObject) //Fisch aus der FischListe löschen
select_new_fish() //per random e. Fisch aus der FischListe auswählen -> darf schwimmen
stop_swimming_fish(id:Number) // per random e. Fisch aus der FischListe auswählen ->
// darf nicht mehr schwimmen
fish_start_swim(fish:FishObject,id:Number) // ein bestimmter Fisch darf schwimmen
fish_stop_swim(fish:FishObject,id:Number) // ein bestimmter Fisch darf nicht schwimmen



Screenshot des Aquariums

## 7. Ausblick und Verbesserungen

### 7.1 am Handy

Neben den schon angesprochenen Verbesserungsmöglichkeiten unter Menüpunkt *4.6 Implementierungen* ist die Mindmap (in Kapitel *1.2 Entstehung der Idee*) für weitere Vorhaben mit Sicherheit eine gute Inspirationsquelle.
Auch eine Datenbank in der man den wöchentlichen Speiseplan der Mensa ablegt, um die Gerichte dann auch an die Fische verfüttern zu können, wäre eine lustige und spannende Idee. Das Projekt bietet unzählige Möglichkeiten einer Fortführung an, und mit ein bisschen Brainstorming lassen sich gewiss noch unzählige Variationen finden.

Möglicherweise sehen wir dieses oder ähnliche Bilder vor der Mensa oder der Cafeteria in Zukunft häufiger

### 7.2 serverseitig

**FishObject + Unterklassen**

- Schwimmverhalten optimieren ( Beschleunigung usw. )
- Weitere Fischarten mit unterschiedlichen Merkmalen

- Schwarmbildung
- Geburten
- Fische fressen
- Weitere Animationen einbauen

**FishLiveCycleTimer**

- Fisch soll sich im Aquarium weiter entwickeln, KI
- Abhängigkeit der Parameter wie speed, colour usw. implementieren (z. B. von der Fütterung abhängig)
- Aussehen des Fisches verändert sich (abhängig vom Alter, Vitamine…)

**FishKollision**

- Fisch - Fisch
- Fisch – Futter
- Kollisionen bewirken Veränderung der Fisch – Parameter (z.B. Fitness - Level) aber auch Verhaltensaenderung wie z.B. Schwarmbildung oder Animationen.

**FishLiveGuard**

- Aktuelle Fisch-Parameter regelmäßig in d. DB speichern, Timer (Programmabsturz)

**FoodTimer**

- Regelt die Fütterung der Fische ( z.B.  täglich um 12 Uhr)

**InfoScreen**

- Infoscreen implementieren
- Lebt mein Fisch noch?  Wie alt ist er geworden?  Warum ist er gestorben?

**DataBase**

- Informationen im Internet verwalten
- neue Möglichkeiten der Interaktion ( z.B. der Benutzer kann eine neue Fischart kreieren und sie in das System einpflegen ) -> dynamische Weiterentwicklung

## 8. Fazit

Die Beschäftigung mit der Programmierung mobiler Geräte hat einen sehr interessanten Einblick in einen Bereich ermöglicht, den man in diesem Umfang so schnell sicher nicht wieder bekommt, wenn man nicht beruflich direkt in diese Sparte einsteigt.

Die Hintergründe der einzelnen Gruppenmitglieder, sowie die vertretenen Studiengänge Informatik, Multimedia, Wirtschaftsinformatik und Gestaltung waren sehr vielfältig, und eine so umgreifende Zusammenarbeit ist im Studium eher selten. Da dieses Projekt ein Schwerpunktfach im Hauptstudium darstellt, muss ich an dieser Stelle im Vergleich zu anderen Schwerpunktseminaren feststellen, dass man in Projektgruppen wesentlich mehr lernt, und vor Allem fürs Leben lernt. Gelernte Inhalte für Prüfungen geraten schneller in Vergessenheit, und zudem hat man am Semesterende nichts vorzeigbares in der Hand. Das Arbeiten in Teams, sowohl was die Ideenfindung als auch das Arbeiten mit Subversion betrifft, war sehr lehr- und abwechslungsreich. Die Anpassung und Einbindung eigener erstellter Dateien in einen technischen Rahmen, die Fehlersuche und das Testen der Applikation mit verschiedenen Emulatoren, sowie verschiedenen Handy-Modellen waren oft sehr zeitintensiv und nicht immer leicht.
Aber insgesamt haben wir einen intensiven Einblick in die Anwendungsentwicklung für mobile Geräte und interessante Erkenntnisse im Bereich der Software-Entwicklung unter stark eingeschränkten Bedingungen und mit in hohem Maße begrenzten Ressourcen, bekommen. Aber das Schönste ist die Tatsache, dass unser Projektteam es geschafft hat, der Zielsetzung gerecht zu werden, einen Prototyp zur Handy-Fischzüchtung und dessen Verhalten im Aquarium, zu realisieren.
Auch die anderen Teams haben ihre Zielsetzungen erreicht, und durch die Vielfalt der Projekte, die in diesem Kurs entstanden sind, lässt sich erahnen was noch alles möglich ist, und schon in naher Zukunft realisiert werden wird.
Als nächsten Schritt freuen wir uns auf die Umsetzung unserer Projekte auf dem Public Screen im Eingangsbereich der neu entstehenden Mensa.

## 9. Quellen

[BL06] Projekt Blinkenlights, http://www.blinkenlights.de, Abruf vom 04.06.2006

[J2P06] J2ME Polish, http://www.j2mepolish.org, Abruf vom 07.04.2006

[CL06] J2ME Polish Installations-how-to,
http://www.fh-augsburg.de/~curly/Mob/how_to.pdf, Abruf vom 10.06.2006

[KDS04] Klaus -Dieter Schmatz, Java 2 Micro Edition, dpunkt.verlag, 2004, Seite 75

[DV06] http://www.docuverse.com/blog/donpark/EntryViewPage.aspx?guid=b2e7afb7-f54a-4f1f-872d-f3081ca0804a, Abruf vom 12.07.2006

## weiterführende Informationen:

zu empfehlende Literatur:
J2ME: Klaus-Dieter Schmatz, Java 2 Micro Edition, dpunkt.verlag, 2004
J2ME Polish: Robert Virkus, Pro J2ME Polish, Apress Verlag, 2005

J2ME Polish Übersicht für Einsteiger:
www.sigs.de/publications/js/2005/02/kraft_JS_02_05.pdf, Abruf vom 12.07.2006

J2ME Polish Tutorials: www.java2s.com/java/j2me, Abruf vom 12.07.2006