Höhere technische Bundes- Lehr- und Versuchsanstalt Wiener Neustadt
Höhere Abteilung für Elektronische Datenverarbeitung und Organisation

Schuljahr 2003/04

DIPLOMARBEIT

ProgrammierTrainer & ExampleBuilder

von

Martin REITERER / 5BD

Erhard DINHOBL / 5BD

Betreuender Lehrer: Dipl.-Ing. Harald HABERSTROH

Schuljahr 2003/04

DIPLOMARBEIT

ProgrammierTrainer & ExampleBuilder

ausgeführt

an der

Höheren Abteilung für Elektronische Datenverarbeitung und Organisation

der

Höheren technischen Bundes- Lehr- und Versuchsanstalt Wiener Neustadt

von

Martin REITERER
Loipersbacherstrasse 9
2620 Neunkirchen-Peisching

Erhard DINHOBL
Gartengasse 3
2620 Neunkirchen-Mollram

_____ Unterschrift

_____ Unterschrift

unter Betreuung von

Dipl.-Ing. Harald HABERSTROH

Wiener Neustadt, am 22.05.2004

„Ein guter Programmierer erkennt nicht die Programmiersprache
sondern das Problem“

1 Kurzfassung (ED)

Viel wichtiger als die Syntax einer Programmiersprache zu beherrschen ist, dass man sich den Ablauf eines Programms vorstellen kann. Für Programmieranfänger ist es zu Beginn oft schwer sich auf das abstrakte Denken, welches zum Programmieren notwendig ist, einzustellen.

Im Wesentlichen besteht der Hauptaufwand beim Schreiben eines Programms im Analysieren der Aufgabenstellung und im Formulieren eines geeigneten Algorithmus.

Mithilfe des ProgrammierTrainers soll programmiersprachenunabhängig das Erstellen von Algorithmen in Form von Anweisungslisten geübt werden können. Gleichzeitig übt der Programmieranfänger logisches Denken und das Analysieren der Aufgabenstellung. Dies alles sind Fertigkeiten, die für das Programmieren notwendig sind.

Mit dem ProgrammierTrainer ist es möglich, sofort mit der Formulierung von Algorithmen zu beginnen. Es muss auch keine Programmiersprache erlernt werden, da die möglichen Anweisungen beliebigen Text enthalten können. Schüler wählen die benötigten Anweisungen, welche zum Formulieren eines Algorithmus benötigt werden, aus einer vorgegebenen Liste mit Anweisungen aus und können so durch Pseudo-Code-Anweisungen den Programmablauf bestimmen.

1.1 Abstract

Visualizing the flow of a program is far more important than mastering the syntax of a programming language. Beginners often find it difficult to focus on abstract reasoning, which is a necessary prerequisite for programming.

When a program is to be written most of the effort is required for the analysis of the set task and the formulation of a suitable algorithm.

The ProgrammierTrainer is to enable you to practice setting up algorithms as instruction list. Thus the beginner practices logical reasoning and analyzing the set task. These are skills absolutely necessary to programming.

The ProgrammierTrainer will make it possible to immediately start formulating the algorithms. Not even a programming language needs to be learned, all acceptable instructions may contain any kind of text. Students will select the necessary instructions required to formulate the algorithm from a pre-set table of instructions. Thus they can determine the flow of the program by means of pseudo-code instructions.

Inhaltsverzeichnis / Contents

1 Kurzfassung (ED) 4
1.1 Abstract 4
2 Einleitung (ED) 8
3 Produktumgebung (MR) 9
3.1 Hardware 9
3.1.1 Minimal-Anforderungen: 9
3.1.2 Empfohlene Hardware-Konfiguration: 9
3.2 Software 9
3.3 Entwicklungsumgebung 9
3.4 Organisationsumgebung 9
4 Installation (ED) 10
4.1 Wichtige Hinweise 10
4.1.1 Jython 10
4.2 Was wird alles benötigt? 10
4.3 Installieren 11
4.3.1 install.bat/install.sh ausführen 11
4.3.2 Pfadangabe für die jython.jar-Datei 11
4.3.3 ExampleBuilder installieren? 11
4.3.4 Installieren 11
4.3.5 Start unter anderen Betriebssystemen als Linux und Windows 11
4.4 Welche Dateien und Ordner werden erzeugt 11
4.4.1 Das Installationsprogramm InstallPT 11
4.4.2 Der ProgrammierTrainer und ExampleBuilder 11
5 Deinstallation (ED) 13
6 Systementwurf (ED) 14
6.1 Das Testen 14
6.1.1 Eingabe/Ausgabe-Werte definieren 14
6.1.2 Variablen-Endwerte 14
6.2 Schnittstelle: Jython / Java 15
7 ProgrammierTrainer (ED) 16
7.1 Hauptformular 16
7.2 Menüpunkte 17
7.2.1 Datei 17
7.2.2 Fenster 19
7.2.3 Interpreter 20
7.2.4 Hilfe 21
7.3 Aufgabendialog 22
7.4 Ausgabefenster 22
7.5 Aufgabe-Öffnendialog 24
7.6 Optionendialog 24
7.7 Editor 25
7.8 Hilfedialog 25
7.9 Infodialog 25
8 ExampleBuilder (ED) 26
8.1 Hauptformular 27
8.2 Menüpunkte 27
8.2.1 Datei 27
8.2.2 Aufgabe 28
8.2.3 Dienstprogramme 28
8.2.4 Hilfe 29
8.3 Ergebnissüberprüfungsdialog 30

8.4 Infodialog 31
9 Zusatzmodul – Hamster-Dialog (MR) 32
9.1 Einbindung in Jyhton 32
9.2 Die Oberfläche 32
9.2.1 Menüpunkte 33
9.2.2 Bilder laden 33
9.3 Schaltflächen 34
9.3.1 RESET 34
9.3.2 Reset Welt 34
9.3.3 Reset Hamster 34
9.3.4 Run-/Edit-Mode 34
9.4 Bedienung des Hamster-Dialogs 35
9.4.1 Edit-Mode 35
9.4.2 Run-Mode 35
9.5 Methoden-Übersicht 36
9.6 Exceptions 37
9.6.1 MauerException 37
9.6.2 KornException 38
10 Verschlüsselungsmodul (MR) 39
10.1 Verfahrensauswahl 39
10.1.1 Ausgangslage 39
10.1.2 Allgemeines zu DES 39
10.1.3 Wie entstand DES? 40
10.1.4 Eigenschaften von DES 40
10.1.5 Wie sicher ist DES? 40
10.1.6 Grafische Darstellung der Ver- und Entschlüsselung: 41
10.1.7 Beispiele zu DES 42
10.2 Modulerklärung 44
10.2.1 Konstruktor 44
10.2.2 entschluessle 44
10.2.3 verschluessle 44
10.3 Ersetzbarkeit 44
10.3.1 Aufbau der neuen Verschlüsselungsklasse 44
11 Produktdaten (MR) 46
11.1 Data Dictionary 46
11.1.1 Hamster-Dateien 46
11.1.2 E/A-Dateien 47
11.1.3 Optionen-Datei 48
11.1.4 Zuordnungsdatei 49
12 Benutzungskonzept (MR) 50
12.1 Use-Case-Diagramme 50
12.1.1 Übersicht aller Teilprogramme 50
12.1.2 Hauptprogramm – ProgrammierTrainer 51
12.1.3 Example-Builder 52
12.2 Ablaufdiagramme 53
12.2.1 Erstellen einer Aufgabe 53
12.2.2 Lösen einer Aufgabenstellung 54
12.3 Beschreibung der USE-CASE-Diagramme 55
12.3.1 ProgrammierTrainer 55
12.3.2 Example-Builder 63
13 Lösungsansatz (MR) 70
13.1 Klassendiagramm 70

13.1.1 ProgrammierTrainer 71
13.1.2 Attribute und Methoden der Klassen im ProgrammierTrainer 71
13.1.3 ExampleBuilder 72
13.1.4 Attribute und Methoden der Klassen im ExampleBuilder 72
14 Ausblick (ED) 73
15 Anhang A) komplette E/A-Aufgabe (MR) 74
16 Anhang B) komplette Hamster-Aufgabe (MR) 76
17 Danksagung 77
18 Literatur 78
19 Lebensläufe 79

2 Einleitung (ED)

Grundsätzlich besteht der Programmiertrainer aus zwei große Teile. Einerseits soll der ProgrammierTrainer dem Benutzer ein Interface zum Erstellen von Algorithmen bieten. Andererseits soll dem Lehrer ein Interface, der ExampleBuilder, zur Verfügung stehen, in welchem dieser seine Aufgabenstellungen definieren kann.
Der ProgrammierTrainer kann sowohl für schulische Zwecke als auch zum Erlernen von Programmieren eingesetzt werden. Der ExampleBuilder wird von denjenigen eingesetzt werden, die Aufgaben für Benutzer des ProgrammierTrainers entwerfen und zur Verfügung stellen.
Ein Benutzer des ProgrammierTrainers sollte zur Bedienung wissen, wie man ein normales Programm bedient (Und eine Portion Willen zum Erlernen von Programmieren).
Ein Benutzer des ExampleBuilders muss die im Hintergrund verwendete Sprache des ProgrammierTrainers, genannt Python.

3 Produktumgebung (MR)

Im nachfolgenden Abschnitt wird die Umgebung, in der das Produkt zum Einsatz kommt, beschrieben.

3.1 Hardware

Der ProgrammierTrainer ist grundsätzlich ein Einzelplatzsystem, welches jedoch seine Aufgabendateien auch von einem Server beziehen kann. Zu diesem Zweck wird ein PC mit Netzwerkanbindung benötigt, um die Funktionalitäten des ProgrammierTrainers voll nutzen zu können.

3.1.1 Minimal-Anforderungen:

Bildschirm-Auflösung:	800x600
CPU / MHz:	450 MHz
Festplattenspeicher:	10-15 MB freier Speicher
Netzwerkverbindung:	10 MBit

3.1.2 Empfohlene Hardware-Konfiguration:

Bildschirm-Auflösung:	1024x768 oder höher
CPU / MHz:	900 MHz – 1 GHz
Festplattenspeicher:	20 MB freier Speicher
Netzwerkverbindung:	100 MBit

3.2 Software

Da der ProgrammierTrainer in der Programmiersprache Java implementiert wurde, benötigt das Zielsystem, auf dem der ProgrammierTrainer letztendlich laufen wird, ein Java Runtime Environment in der Version 1.4 oder höher.

3.3 Entwicklungsumgebung

Zum Entwickeln und Testen des ProgrammierTrainers wurde der von Borland entwickelte JBuilder in der Version 9.0 verwendet. Wir garantieren, dass der ProgrammierTrainer mit dieser Version des JBuilders einwandfrei funktioniert.

3.4 Organisationsumgebung

Der ProgrammierTrainer soll im Programmierunterricht eingesetzt werden. Er wird für den Einstieg in das imperative Programmieren verwendet.

4 Installation (ED)

In den folgenden Unterkapiteln erfahren Sie alles über die Installation des ProgrammierTrainers und ExampleBuilders.

4.1 Wichtige Hinweise

In diesem Kapitel werden Hinweise besprochen, die zur Ausführung der beiden Applikationen einzuhalten sind:

- Die Package-Verzeichnisse des ProgrammierTrainers und ExampleBuilders sollten im selben Verzeichnis liegen
- Falls die im folgenden beschriebene Installation nicht funktioniert, da z.B. die Applikationen unter MacOS oder anderen Betriebssystemen durchgeführt wird, kann der ProgrammierTrainer sowie der ExampleBuilder trotzdem gestartet werden. Zu diesem Zweck muss das JAR-Archiv entpackt werden. Um den ProgrammierTrainer dann zu starten dient die Klasse `programmiertrainer.Hauptformular` und für den ExampleBuilder die Klasse `examplebuilder.Hauptformular`. Als erster Parameter muss beim Aufruf der jeweiligen Klasse der Pfad des Verzeichnisses in dem die Package-Verzeichnisse liegen, angegeben werden (weiteres siehe 4.3.5).
- Der ProgrammierTrainer und ExampleBuilder verwenden Jython.
- Um den das Archiv mit den Applikationen zu entpacken und diese danach auch zu installieren, wird die Installation von JDK vorausgesetzt. Das Entpacken und die Ausführung greift auf die JVM (Java-Virtual-Machine) zu, welche die ausführbare Datei, z.B. `java.exe`, ist und auf den Javaarchiver, z.B.: `jar.exe`, zu. Um die Verwendung dieser Programm zu gewährleisten sollte beachtet werden, dass ein Verweis auf den Ort dieser in der Systemvariable „`path`“ vorhanden ist. Falls dies nicht der Fall ist, muss dieser erweitert werden.

4.1.1 Jython

Jython ist ein in Java implementierter Interpreter der Programmiersprache Python. Mithilfe diesem ist möglich auf Java-Applikationen zuzugreifen. Um Jython von Java aus zu nutzen, braucht man das Archiv „`jython.jar`“. Dazu es muss ein Eintrag für dieses Archiv im `CLASSPATH` vorhanden sein. Die Java-Klassen in diesem Archiv nutzen die bei der Installation von Jython erstellten Python-Bibliotheken, welche im Verzeichnis „`Lib`“ liegen. Diese sind aber nicht so umfangreich wie bei einer Python-Installation. Wie später erklärt, können die von einer Standard-Python-Installation erstellten Bibliotheken trotzdem verwendet werden.

4.2 Was wird alles benötigt?

Um den ProgrammierTrainer und ExampleBuilder auszuführen, werden folgende Komponenten **unbedingt** benötigt:

- Java Development Kit (JDK) 1.4 oder höher
- Jython (jython.jar-Archiv)
- Python-Bibliotheken

Folgende Komponenten können zur Erweiterung verwendet werden:

- neues jython.jar-Archiv (ersetzt dann das oben erwähnte Archiv
- Python-Bibliotheken von einer vorhandene Python- oder Jython-Installation

4.3 Installieren

Im folgenden Kapitel wird die Installation Schritt für Schritt erklärt. Diese ist für Windows als auch Linux konzipiert.

4.3.1 install.bat/install.sh ausführen

Nachdem eine dieser Dateien (.bat-Datei für Windows und .sh-Datei für Linux) ausgeführt wurde, ist das Archiv mit den 2 Programmen und allen Komponenten entpackt und es wird automatisch das Installationsprogramm InstallPT aufgerufen, welches die Startdateien für den ProgrammierTrainer und ExampleBuilder generiert..

4.3.2 Pfadangabe für die jython.jar-Datei

Diese Datei ist nach dem Entpacken, was durch das Ausführen von install.bat oder install.sh passiert im Verzeichnis vorhanden. Um eine andere Version von Jython zu verwenden muss bei der Pfadangabe das entsprechende Archiv ausgewählt werden. Jedoch muss darauf geachtet werden, dass Jython auf die Python-Bibliotheken zugreifen kann (siehe 4.1.1 „Jython“).

4.3.3 ExampleBuilder installieren?

Die Checkbox, ob der ExampleBuilder installiert werden soll oder nicht, hängt davon ab, ob das Package-Verzeichnis „examplebuilder“ existiert oder nicht.

4.3.4 Installieren

Um nun den Vorgang abzuschließen, muss der Button „Installieren“ geklickt werden. Hierbei werden die unter 4.4 beschriebenen Datei erzeugt

4.3.5 Start unter anderen Betriebssystemen als Linux und Windows

Die Installationsroutine InstallPT wurde nur für Windows und Linux entwickelt. Das Ausführen vom ProgrammierTrainer und ExampleBuilder unter einem anderen Betriebssystem, ist nicht vorgesehen und es wird auch nicht weiter darauf eingegangen, es werden nur ein paar Hinweise gegeben:

- Zum Ablauf werden die gleichen Komponenten wie für Windows und Linux benötigt
- Es muss das Archiv entpackt werden
- Als erster Parameter beim Aufruf der Applikationen muss der Pfad, in dem sich die Package-Verzeichnisse befinden, angegeben werden.
- Kapitel 4 „Installation“ lesen

4.4 Welche Dateien und Ordner werden erzeugt

In den folgenden Unterkapitel wird beschrieben, welche Datei und Ordner von den einzelnen Applikationen, dem Installationsprogramm, dem ProgrammierTrainer und ExampleBuilder, erstellt werden.

4.4.1 Das Installationsprogramm InstallPT

Vom Installationsprogramm werden die zwei Start-Dateien für den ProgrammierTrainer und den ExampleBuilder erzeugt. Unter Windows sind das „`ProgrammierTrainer.bat`“ und „`ExampleBuilder.bat`“ und unter Linux „`ProgrammierTrainer.sh`“ und „`ExampleBuilder.sh`“.

4.4.2 Der ProgrammierTrainer und ExampleBuilder

Vom ProgrammierTrainer und ExampleBuilder wird nur ein Verzeichnis und eine darin liegende Datei erzeugt. Dieses Verzeichnis liegt im User-Verzeichnis und hat dem Namen

„programmiertrainer“. Unter Windows wäre dies z. B.: „`C:\Dokumente und Einstellungen\User1\programmiertrainer`“, oder unter Linux „`/home/User1/programmiertrainer`“. In diesem Verzeichnis liegt die Datei „`optionen.xml`“. Sie enthält Informationen über Einstellungen im ProgrammierTrainer und ExampleBuilder. Diese Datei sollte nicht Schreibgeschützt sein, da sonst keine Einstellungen von den Applikationen geschrieben werden können. Die beiden Programme greifen über die Java-System-Variable „`user.home`“ auf das Verzeichnis zu. Die Datei kann vom Benutzer editiert werden, solange gültige Einstellungen geschrieben werden und die Datei eine gültige XML-Datei bleibt.

5 Deinstallation (ED)

Das folgende Kapitel sollte vor der Durchführung komplett gelesen werden.

Um den ProgrammierTrainer und ExampleBuilder zu deinstallieren, muss das Verzeichnis wie unter 4.4.1 beschrieben und das Verzeichnis im welchem sich der ProgrammierTrainer und ExampleBuilder befinden, gelöscht werden. Achtung zuletzt erwähnten Verzeichnis liegt wahrscheinlich auch das Archiv „`ptvX.X.jar`“ welches für eine Neuinstallation benötigt wird. Daher sollte diese gesichert werden, falls es noch einmal benötigt wird.

6 Systementwurf (ED)

In den folgenden Unterkapiteln werden Schwerpunkte dieser Diplomarbeit genauer beschrieben.

6.1 Das Testen

Ein wesentlicher Punkt des ProgrammierTrainers ist die Überprüfung, ob ein vom Benutzer entwickelter Algorithmus korrekt ist oder nicht. Im folgenden Kapitel wird die genaue Funktionsweise dieses Mechanismus beschrieben.

Für die folgenden Erläuterungen wird folgendes Programm verwendet:

```
1: gib „Quadrat berechnen“ aus
2: lese vom Benutzer in Variable a ein
3: lese vom Benutzer in Variable b ein
4: speichere in Variable c das Produkt aus a und b
5: gib das Produkt von a und b aus
```

Grundsätzlich können zum Überprüfen eines Algorithmus, beliebig viele Tests definiert werden. Ein Test kann aus Eingabe- und Ausgabe-Werten und Variablen-End-Werten bestehen. Anhand von Eingabewerten wird dem ausgeführten Programm ein Benutzer „simuliert“ der es mit den definierten Werten füttert. Der ProgrammierTrainer fängt dann die vom Programm produzierten Ausgaben ab und vergleicht diese mit den definierten Ausgabewerten. Sind diese identisch, wird der ausgeführte Test mit „OK“ bewertet, andernfalls mit „Nicht OK“.

6.1.1 Eingabe/Ausgabe-Werte definieren

Wird mindestens ein Eingabe- oder Ausgabewert definiert, wird beim Ausführen eines Tests dieser Wert verwendet. Verlangt das Programm dann weitere Eingaben vom Benutzer oder produziert weitere Ausgaben, wird der ausgeführte Test als „Nicht OK“ beurteilt. Als Ausgabewerte kann ein „*“ angegeben werden, d.h. die Ausgabe an dieser Stelle ist irrelevant und kann irgendetwas sein.

Beispiel für einen Test für das oben genannte Programm:

Eingabewerte	**Ausgabewerte**
2 4	8

6.1.2 Variablen-Endwerte

Abhängig von den im Test definierten Eingabewerten können Endwerte für Variablen definiert werden. Am Ende eines Programmdurchlaufes werden dann die angegebenen Variablen mit den Endwerten verglichen. Sind die Werte der Variablen mit den End-Werten identisch, wird der Test als „OK“ beurteilt andernfalls mit „Nicht OK“.

Beispiel für Variablen-Endwerte für das oben genannte Programm, abhängig von den obigen unter 6.1.1 definierten Eingabewerten:

Variablenname	**Endwert**
c	8

Erklärung: in der Variable c wird auch das Produkt der beiden Variablen gespeichert, welches in diesem Test 8 wäre (2 * 4)

Bei der Endwertüberprüfung werden die Werte der entsprechenden Variablen mittels `str()` in Strings umgewandelt und dann mit den definierten Endwerten verglichen. D.h.

als Endwerte dürfen alle Ausdrücke verwendet werden, die auch in Strings zu dessen Vergleich in Python zugelassen sind. Der String wird mit einen einfachen Hochkomma verglichen.

6.2 Schnittstelle: Jython / Java

Damit das ausgeführte Programm unter anderem auch den Hamster bewegen kann muss es auf das Hamster-Panel zugreifen. Währendessen ein Programm ausgeführt wird, stehen folgende Objekte zur Verfügung:

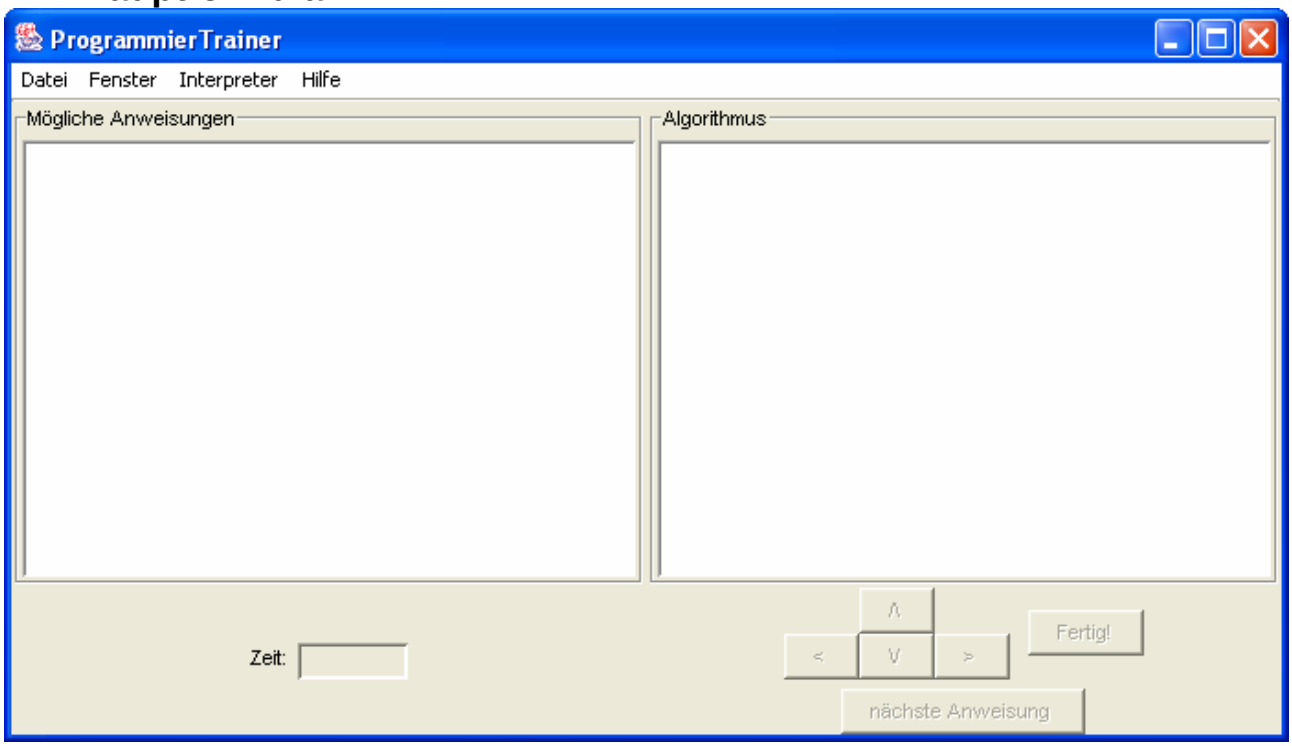
- `io`
  Dieses Objekt dient zur Eingabe, Ausgabe und Hamster-Steuerung. Die enthaltenen Methoden sind in der Javadoc dokumentiert.
- `hamster`
  Dieses Objekt ist identisch mit dem „io“-Objekt. Es wurde zur Vollständigkeit hinzugefügt.
- `trace`
  Dieses Objekt dient zur Programmausführung und ist für den Benutzer nicht weiter wichtig. Jedoch sollte keine Variable den Namen „trace“ tragen.

Um aber das Entwickeln eines Programms zu vereinfachen, wurden Funktionen in der Datei „`initscript.py`“, welche im Verzeichnis „`pycode`“ liegt, definiert. Diese Datei wird bei jedem Start eines Programms ausgeführt. Diese Methoden greifen lediglich auf die im Objekt „`io`“ bzw. „`hamster`“ definierten Methoden zu und liefern deren Wert zurück. Diese Datei kann beliebig erweitert werden, um möglicherweise selbst entwickelte Funktionen zur Verfügung zu haben. In dieser Datei können auch die zu importierenden Bibliotheken eingetragen werden.

7 ProgrammierTrainer (ED)

In den folgenden Unterkapiteln wird die Funktionsweise des ProgrammierTrainers beschrieben.

7.1 Hauptformular



Das Hauptformular soll folgende, der Aufgabe entsprechende, Funktionen haben:

Eingabe/Ausgabe-Aufgabe

Die gestellte Aufgabe muss gelöst werden, indem man die richtigen Anweisungen in der richtigen Reihenfolge in der Algorithmus-Listbox anordnet. Alle Anweisungen (in zufälliger Reihenfolge in Pseudo-Code verfasst) werden separat angezeigt (evtl. auch nicht benötigte Anweisungen). Die Anweisungen werden mittels Drag&Drop in die Algorithmus-Listbox gezogen. Die verschobene Anweisung soll dann in der ursprünglichen Listbox, mit den möglichen Anweisungen, ausgeblendet werden. Es kann auch im umgekehrten Weg erfolgen, d.h. es kann eine Anweisung die im Algorithmus nicht erwünscht ist, wieder in die Listbox mit den möglichen Anweisungen, mittels Drag&Drop, verschoben werden. Hierbei wird die Anweisung analog zu vorhin von der Algorithmus-Listbox ausgeblendet und in die Listbox mit den möglichen Anweisungen verschoben.

Hamster-Aufgabe

Die gestellte Aufgabe muss gelöst werden, indem man die richtigen Anweisungen in der richtigen Reihenfolge in der Algorithmus-Listbox anordnet. Alle Anweisungen (in zufälliger Reihenfolge in Pseudo-Code verfasst) werden separat angezeigt (evtl. auch nicht benötigte Anweisungen). Die Anweisungen werden mittels Drag&Drop in die Algorithmus-Listbox gezogen. Die verschobene Anweisung soll bei den Hamsterbeispielen NICHT aus der Listbox, welche eine Sammlung aller Anweisungen beinhaltet, ausgeblendet werden. Ein Grund dafür ist: Wenn der Hamster einen Weg gehen soll, gibt es verschiedene Möglichkeiten wie er diesen beschreitet. Deshalb soll z.B. eine Anweisung, die den Hamster einen Schritt nach vorne schickt „en' masse“ vorhanden sein. Es kann auch im umgekehrten Weg erfolgen, d.h. es kann eine Anweisung die im Algorithmus nicht erwünscht ist wieder in die Listbox mit der Sammlung der Anweisung, mittels Drag&Drop, verschoben werden. Hierbei wird die Anweisung analog zu vorhin von der Algorithmus-

Listbox ausgeblendet. Da diese Anweisung jedoch beim verschieben in die Algorithmus-Listbox aus der Sammlung der Anweisungen nicht ausgeblendet wurde, muss sie auch nicht wieder dorthin aufgenommen werden.

Shortcuts

Tasten	**Funktion**
Strg + O	Öffnet den Aufgabeöffnen-Dialog (siehe 7.5)
Strg + E	Öffnet den Editor (siehe 7.7)
Strg + Q	Beenden den ProgrammierTrainer
Strg + A	Öffnet den Aufgabendialog (siehe 7.3)
Strg + S	Öffnet den Optionendialog (siehe 7.6)
Strg + 5	Startet das Programm im Ausführen-Modus (siehe 7.2.3)
Strg + 6	Startet das Programm im Debug-Modus (siehe 7.2.3)
Strg + 7	Testet das Programm (siehe 7.2.3 und 6.1)
Strg + X	Führt einen Anweisung während des Debug-Modus aus (Eval – siehe 7.2.3)
Strg + N	Führt die nächste Anweisung im Debug-Modus aus (siehe 7.2.3)
/\ (Nach-Oben-Taste)	Verschiebt die markierte(n) Anweisung(en) der Algorithmus-Listbox nach oben
\/ (Nach-Unten-Taste)	Verschiebt die markierte(n) Anweisung(en) der Algorithmus-Listbox nach unten
< (Nach-Links-Taste)	Rückt die markierte(n) Anweisung(en) der Algorithmus-Listbox um 1 Block nach außen
> (Nach-Rechts-Taste)	Rückt die markiere(n) Anweisung(en) der Algorithmus-Listbox um 1 Block nach innen

7.2 Menüpunkte

In den folgenden Unterkapiteln wird beschrieben, welche Funktionen die einzelnen Menüpunkte der Menüeinträge haben.

7.2.1 Datei

Menüeintrag Datei.

Aufgabe öffnen

Es wird der Aufgabe-Öffnendialog (beschrieben in 7.5) angezeigt und die dort ausgewählte Aufgabe geöffnet.

Python – Code

Es wird ein Editor geöffnet (siehe 7.7) in dem der Benutzer nach seinen eigenen Vorstellungen die Aufgabe in Python lösen kann. Beim Schließen des Editors wird nachgefragt ob der Benutzer den Programm-Code in den ProgrammierTrainer übernehmen will. Wenn diese Frage mit Ja beantwortet wird, wird in der Algorithmus-Listbox der geschriebene Programmcode angezeigt. Ist für eine im Python-Editor geschriebene keine Pseudo-Code-Anweisung vorhanden, wird im Hauptformular in den beiden Listboxen der Python-Code angezeigt und der Menüpunkt, durch welchen man zwischen der Anzeige vom Pseudo-Code und Python-Code umschalten kann, deaktiviert.

Drucken

Es soll das entworfene Programm in 3 Formaten gedruckt werden können (Sicht A4):

Format 1

Aufgabentitel: Aufgabentitel
Autor: [Name]
Zeit: xx:xx
Anz. Hinweise: xx
Musterlösung verwendet: [ja/nein]

Z#	*Programm-Code*	*Pseudo-Code*
1:	Anweisung1	Pseudo-Code1
2:	Anweisung2	Pseudo-Code2
3:	Anweisung3	Pseudo-Code3

...

Tests
1. erfolgreich
2. fehlgeschlagen
...

Seite i/n

Format 2

Aufgabentitel: Aufgabentitel
Autor: [Name]
Zeit: xx:xx
Anz. Hinweise: xx
Musterlösung verwendet: [ja/nein]

Z#	*Programm-Code*
1:	Anweisung1
2:	Anweisung2
3:	Anweisung3

...

Tests
1. erfolgreich
2. fehlgeschlagen
...

Seite i/n

Format 3

Aufgabentitel: Aufgabentitel
Autor: [Name]
Zeit: xx:xx
Anz. Hinweise: xx
Musterlösung verwendet: [ja/nein]

Z# *Pseudo-Code*
1: Pseudo-Code1
2: Pseudo-Code2
3: Pseudo-Code3
...

Tests
1. erfolgreich
2. fehlgeschlagen
...

Seite i/n

Beschreibung der Blattinformationen
Auf dem Blatt sollen (von oben nach unten) folgende Aufgabeninformationen abgebildet sein:

1. der Aufgabentitel (der Titel der bearbeiteten Aufgabe)
2. der Autor (der Namen des Autors ist der Name des eingeloggten Benutzers)
3. die Zeit (die Zeit die vom Autor benötigt wurde um dieses Programm zu erstellen)
4. die Anz. der Hinweise (Anzahl der benötigten Hinweise – siehe Menüeintrag „Hinweis“)
5. Musterlösung verwendet (es sollen nur die Werte „ja“ oder „nein“ ausgedruckt werden, je nach dem ob die Musterlösung vom Benutzer eingesehen wurde oder nicht – siehe Menüeintrag „Lösung“)
6. Je nach Format ist der folgende Abschnitt am ausgedruckten Blatt unterschiedlich:
   Format 1: die Zeilennummer, der Programm-Code mit dem Pseudo-Code
   Format 2: die Zeilennummer und nur der Programm-Code
   Format 3: die Zeilennummer und nur der Pseudo-Code
7. die Status der Tests. Hier wird eine Liste aller Testfälle angezeigt.
   Wurde ein Test erfolgreich durchgeführt lautet die Ausgabe in dieser Zeile: „erfolgreich“. Ist ein Test fehlgeschlagen so lautet die Ausgabe: „fehlgeschlagen“. Wurde nicht getestet, so lautet die Ausgabe: „nicht durchgeführt“.
8. am Schluss jeder Seite soll die Seitenzahl (aktuelle Seite) und die gesamte Anzahl der Seiten stehen

Beenden
Das Programm wird, nach der Rückfrage an den Benutzer, ob wirklich beendet werden soll, inkl. aller Dialoge komplett beendet.

7.2.2 Fenster

Menüeintrag Fenster.

Aufgabe
Es soll der Aufgabedialog (siehe 7.3) angezeigt werden.

Optionen
Es soll der Optionendialog (siehe 7.6) angezeigt werden.

Python-Code anzeigen
Wenn diese Option aktiviert ist, soll in den Listboxen nicht der Pseudo-Code der Anweisungen stehen, sondern der konkrete Programm-Code. Wenn sie deaktiviert ist soll wiederum der Pseudo-Code der Anweisungen in den Listboxen stehen.

Zeit anzeigen
Wenn diese Option aktiviert ist, soll die seit dem Zeitpunkt des Öffnens der Aufgabe verstrichene Zeit angezeigt werden. Wenn diese Option deaktiviert ist, wird die verstrichene Zeit nicht angezeigt.

7.2.3 Interpreter

Menüeintrag Interpreter.

Run
Das entworfenen Programm wird ausgeführt unter der Berücksichtigung des Timeouts (siehe 7.6 Optionendialog/Einstellung: Timeout). Es kann via Programm-Code eine Eingabe (z.B.: Zuweisung eines Werts einer Variable) vom Benutzer verlangt werden. Wenn eine Eingabe verlangt wird, muss das zuvor erwähnte Timeout angehalten werden, da eine Eingabe durch den Benutzer auch länger als das Timeout dauern kann.

Debug
Beim Debuggen wird das Programm wie mit dem Menüpunkt „Run“ ausgeführt. Jedoch wird jede Zeile die als nächstes ausgeführt wird markiert und erst dann ausgeführt, wenn der Benutzer den Button „nächste Anweisung“ drückt.

Abbruch
Das gerade ausgeführte Programm, egal ob via „Run“ oder „Debug“ gestartet, soll abgebrochen werden.

Test
Folgender Menüpunkt kann nur bei einem E/A-Beispiel zur Verfügung stehen, da die Lösung bei einem Hamster-Beispiel durch das Ergebnis (z.B.: wo der Hamster steht, ob er alle Körner aufgesammelt hat, …) sich selbst beschreibt und der Hamster von Lösung zu Lösung im Endeffekt auf verschiedenen Feldern stehen kann. Bei einem E/A-Beispiel müssen aber bestimmte Variablen bei einem bestimmten Initialisierungswert oder Input vom Benutzer, einen bestimmten Endwert haben. Weiters kann das Programm unter Umständen einen bestimmten Output liefern. Die Endwerte und der Output des Programms können somit überprüft werden.

Beispiel:
Ein Programm führt folgende Anweisungen aus:

```
1: b auf 2 setzen
2: in die Variable a vom Benutzer einlesen
3: gib a / b aus
4: b auf b + 1 setzen
5: gib a / b aus
```

Ein Test könnte folgendermaßen aussehen:

- Zum Testen können Eingabewerte definiert werden

  Es können unabhängig von den Variablen Eingabewerte definiert werden, welche, anstatt den Benutzer zu fragen (Zeile 2), verwendet werden. Diese Werte sind in derselben Reihenfolge im Interface anzugeben, wie sie der Benutzer eingibt. So kann im obigen Beispiel ein Eingabewert wie: 3, 9 oder 12 usw. definiert werden. Dazu können...

- Ausgabewerte definiert werden

  Die Ausgaben des Programms werden mit den Ausgabewerten verglichen, die in der Aufgabe abgespeichert sind. Diese Werte sind in der selben Reihenfolge im Interface anzugeben, wie sie vom Programm produziert werden. So liefert das Beispielprogramm in der 3. Zeile bei einem Eingabewert von 3 die Ausgabe 1.5, bei 9 4.5, usw. Der 2. Ausgabewert wäre dann jener von Zeile 5. Dieser wäre bei einem Eingabewert von 9 3 oder bei 12 4 usw. Um aber „auf Nummer sicher zu gehen“, können…

- Variablen-Endwerte definiert werden

  So kann für das Beispiel von oben für die Variable b ein Endwert definiert werden, um zu garantieren, dass das Programm die Variable nur soweit ändert, als erwünscht ist. Im Beispiel wäre der Endwert für die Variable b 3. Dieser Mechanismus ist sinnvoll, um z.B. zu überprüfen, ob ein Programm eine Variable richtig verarbeitet.

  Die Ergebnisse der Testläufe werden dann in einem Fenster angezeigt.

letzte Testergebnisse
Beim Anklicken dieses Menüpunktes werden die letzten Testergebnisse die mittels des Menüpunktes Testen (voriger Punkt) gemacht wurden, noch mal angezeigt.

Eval
Es kann ein Programm-Code direkt dem Interpreter übergeben werden, welcher diesen dann ausführt.

nächste Anweisung
Mithilfe dieses Menüpunktes führt man den Button „nächste Anweisung“ aus.

7.2.4 Hilfe
Menüeintrag Hilfe.

Hilfe zu ProgrammierTrainer
Es wird der Hilfedialog (siehe 7.8) mit einer HTML-Seite, deren Inhalt die Hilfe zum ProgrammierTrainer beinhaltet, angezeigt.

Info
Es wird der Infodialog (siehe 7.9) angezeigt.

Hinweis
Falls in dieser Aufgabe Hinweise zur Verfügung stehen (wird bei der Aufgabenerstellung im ExampleBuilder durch den Ersteller festgelegt), soll beim Ausführen dieses Menüpunktes (anhand der Musterlösung) dem Benutzer folgendes gezeigt werden:

- wenn der bisherige Algorithmus mit dem der Musterlösung übereinstimmt, soll in der Listbox, welche die Sammlung der verfügbaren Anweisungen enthält, die nächste markiert werden, um ihm so den Hinweis auf die nächste Anweisung zu geben
- wenn der bisherige Algorithmus NICHT mit dem der Musterlösung übereinstimmt, soll die letzte Anweisung in der Algorithmus-Listbox markiert werden, um den Benutzer zu zeigen, dass der Algorithmus möglicherweise[1] so nicht das gewünschte Ergebnis liefert.

Lösung
Falls in dieser Aufgabe die Musterlösung zur Verfügung, soll beim Ausführen dieses Menüpunktes das Ausgabefenster (siehe 7.4) geöffnet werden und dort in der vorhandenen Listbox die Musterlösung angezeigt werden. Ob die Musterlösung zur Verfügung steht, wird bei der Aufgabenerstellung im ExampleBuilder durch den Ersteller festgelegt.

7.3 Aufgabendialog

Der Aufgabendialog soll folgende Funktionen der Aufgabe betreffend haben:

Eingabe/Ausgabe-Aufgabe
Es wird lediglich die Beschreibung der zu lösenden Aufgabe dargestellt.

Hamster-Aufgabe
Es wird die Beschreibung der zu lösenden Aufgabe dargestellt. Zusätzlich wird ein Hamster angezeigt, der den Ablauf der zu lösenden Aufgabe ausführt. Hierbei wird die Musterlösung vom Aufgabenersteller verwendet.

7.4 Ausgabefenster

Wenn ein entworfenes Programm ausgeführt wird, wird zuerst das Ausgabefenster geöffnet. Je nach Aufgabenart hat dieses Fenster erweiterte Aufgaben:

Eingabe/Ausgabe-Aufgabe
Je nach Art des Modus durch den das entworfene Programm ausgeführt wird, soll:

- Debug-Modus

  Beim Debuggen wird jede Ausgabe die das Programm produziert in die Listbox des Ausgabefensters aufgenommen.

- Ausführen

  Während des Ausführens, geschieht das selbe wie während des Debuggen.

---

[1] Falls der Algorithmus nicht identisch mit der Musterlösung ist, muss das aber nicht heißen, dass er falsch ist. Die Musterlösung ist lediglich die Lösung des Aufgabenerstellers und ein Programm kann vom Algorithmus verschieden sein aber dasselbe Ergebnis liefern.

Hamster-Aufgabe
Je nach Art des Modus durch den das entworfene Programm ausgeführt wird, soll:

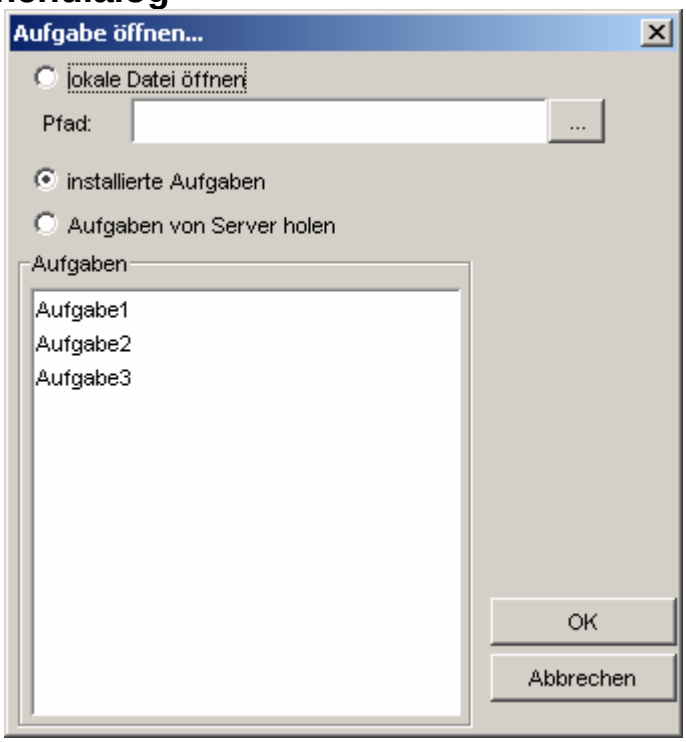
- Debug-Modus

  Beim Debuggen wird jede Ausgabe die das Programm produziert in die Listbox des Ausgabefensters aufgenommen.
  Zusätzlich soll der Hamster angezeigt werden und durch die Anweisungen bewegt werden, wodurch der Benutzer die Auswirkungen seiner Anweisungen sehen kann.

- Ausführen

  Beim Ausführen wird das Hamster-Programm ausgeführt. Jede Anweisung wird nach der Anzahl der Millisekunden, die im Optionendialog (siehe 7.6) angegeben wurden, ausgeführt. Es werden Ausgaben die das Programm produziert in die Listbox des Ausgabefensters aufgenommen.

7.5 Aufgabe-Öffnendialog



Der Aufgabe-Öffnendialog soll folgenden Funktionen bieten:

- Es soll möglich sein eine lokale Aufgaben-Datei zu öffnen
- Es soll möglich sein alle Aufgaben-Dateien, die bereits installiert sind[2], aufzulisten. Hierbei wird der jeweilige Titel der Aufgabe und in Klammer der Dateiname in der Listbox angezeigt. Davon kann man eine Aufgabe auswählen, die geöffnet werden soll.
- Es soll möglich sein, zu einem Server[3] zu verbinden. Dort wird eine Datei[3] geöffnet und diese dann automatisch ausgelesen und analysiert. Diese Datei enthält die Titel der Aufgaben (um nicht jede Aufgabendatei auszulesen) und den jeweiligen Pfad zur Aufgaben-Datei (Pfad am Server). Aus dieser Liste kann der Benutzer eine Aufgabe zum öffnen auswählen. Diese Datei wird beim Öffnen vom Server heruntergeladen.

7.6 Optionendialog

Folgende Optionen werden festgelegt:

- welches lokale Verzeichnis (bezieht sich auf 7.5) das Programm nach Aufgaben durchsucht.
- welchen Server (bezieht sich auf 7.5) sich das Programm verbindet um eine Datei[4] auszulesen, welche die Informationen zu den am Server liegenden Aufgaben enthält.
- einen externen Programm-Editor anzuführen, um statt des internen Editors (siehe 7.7), der mit dem Menüpunkt „Python-Code“ gestartet wird, aufgerufen zu werden. Der interne Editor wird aufgerufen, wenn das Feld leer bleibt.

---

[2] Alle installierten Aufgaben liegen in einem Verzeichnis, welches in der optionen.xml-Datei definiert ist.

[3] in der optionen.xml-Datei definiert

[4] Diese Datei muss den Name „`pt_aufgaben.con`“ besitzen und in dem Verzeichnis liegen, welches als Server angegeben ist.

- wie lange ein Programm, das entworfen wurde maximal laufen darf. Diese Option wird in Sekunden angegeben. Sie ist notwendig, falls im entworfenen Programm z.B.: eine Endlosschleife auftritt. Falls dem so ist wird das Programm nach den angegebenen Sekunden beendet.
- welche Zeit, in Millisekunden, bei der Ausführung einer Hamster-Aufgabe zwischen der Abarbeitung der Einzelnen Anweisungen vergehen soll. Diese Option dient dazu, um den Ablauf, welchen Weg der Hamster durch das entworfene Programm, durch die Hamster-Welt nimmt, verfolgen zu können.

Diese Optionen werden in der `optionen.xml` abgespeichert.

7.7 Editor

Der Editor bietet lediglich eine Art Notepad (Textarea) in dem man wie in einer Entwicklungsumgebung den Programm-Code schreibt. Es werden bei Einrückungen im Programm-Code nicht ein Tabulator sondern drei Leerzeichen eingefügt. Beim Schließen des Editors wird nachgefragt, ob die Änderungen übernommen werden sollen.

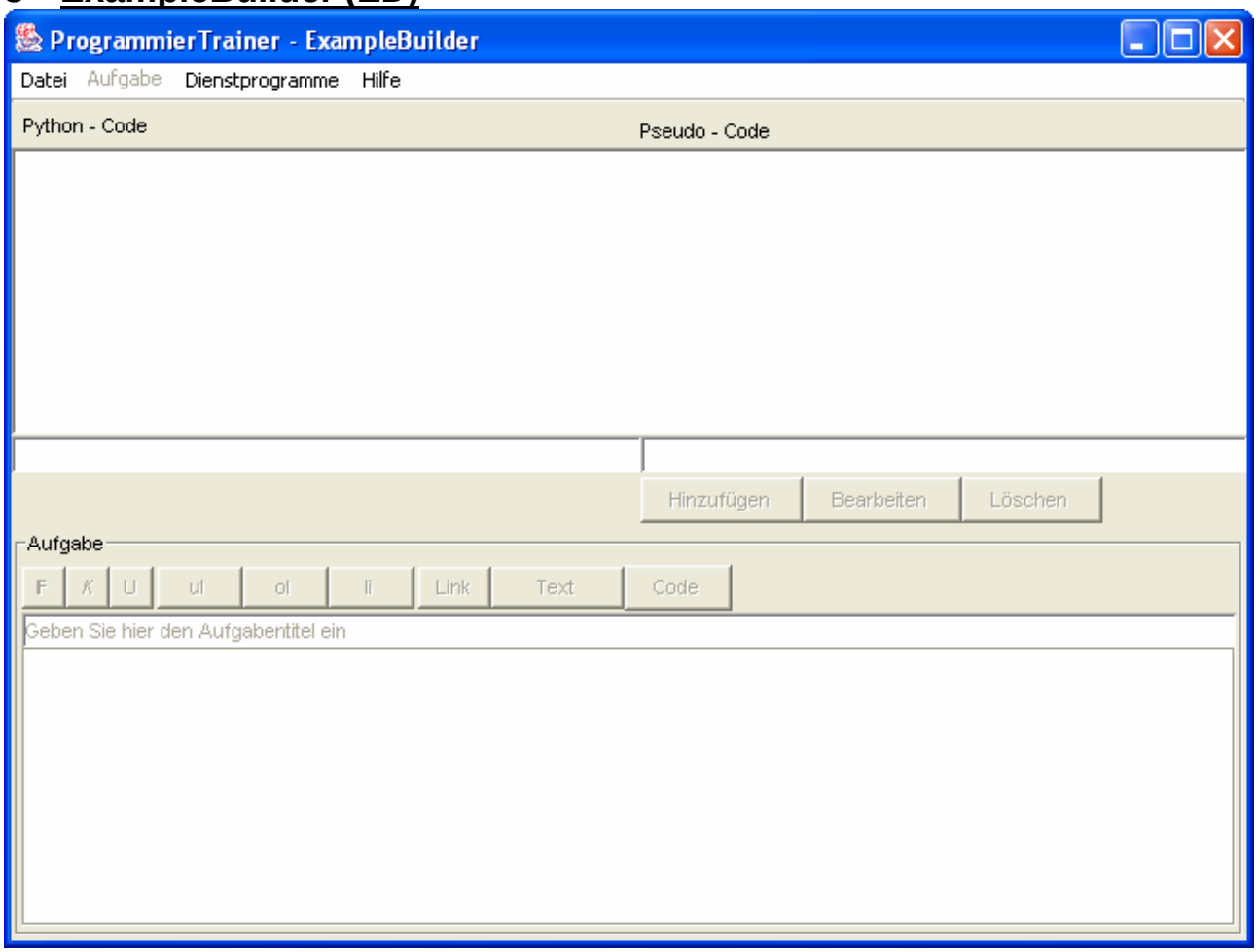
7.8 Hilfedialog

Dieser Dialog zeigt lediglich eine HTML-Seite an, welche eine Beschreibung zum gerade verwendeten Interface beinhaltet.

7.9 Infodialog

Es werden Programminformationen (Autoren, Versionsnummer, etc.) angezeigt.

8 ExampleBuilder (ED)



In den folgenden Unterkapiteln wird die Funktionsweise des Tools, welches zur Erstellung einer Aufgabe für den ProgrammierTrainer dient, beschrieben. Oft erwähnte Dialoge wurden bereits im ProgrammierTrainer (Kapitel 7) beschrieben.

Shortcuts

Tasten	**Funktion**
Strg-O	Menüpunkt: Datei öffnen
Strg-S	Menüpunkt: Datei speichern
Strg-T	Menüpunkt: Template öffnen
Strg-V	Menüpunkt: Testen
Page-Up in der Anweisungsliste	Die momentan markierte Anweisung wird um eine Anweisung nach oben verschoben
Page-Down in der Anweisungsliste	Die momentan markierte Anweisung wird um eine Anweisung nach unten verschoben
Left in der Anweisungsliste	Bei der momentan markierten Anweisung wird eine Einrückung von drei Blanks entfernt
Right in der Anweisungsliste	Bei der momentan markierten Anweisung wird die Einrückung um drei Blanks erweitert.
Entf in der Anweisungsliste	Die momentan markierte Anweisung wird aus der Musterlösung entfernt
Enter in der Anweisungsliste	Die momentan markierte Anweisung wird kopiert

8.1 Hauptformular

Das Hauptformular gliedert sich in 2 Teile. Im oberen Teil kann man die Lösung entwerfen. Zu jeder Anweisung, die man in der entsprechenden Textbox eingibt, kann man einen Pseudo-Code, in einer dafür vorgesehenen Textbox angeben. Spätestens beim abspeichern muss jede Anweisung eine zugehörige Pseudo-Codeanweisung haben. Jede schon eingetragene Anweisung kann man auch modifizieren und löschen. Eine Anweisung kann aber auch eine Zufalls-Anweisung sein. D.h. sie ist nur zur Verwirrung des Benutzers, der die Aufgabe löst, vorhanden. Eine solche hat eine „#“-Symbol am Zeilenbeginn, d.h. man muss im Python-Code-Textfeld als erstes Zeichen das „#“-Symbol schreiben. Somit ist eine Anweisung beim Hinzufügen eine Zufalls-Anweisung.
Im unteren Teil des Formulars kann man den Text zur Aufgabe erstellen, der dann später im Aufgabendialog (siehe 7.3) angezeigt wird. Man kann einen markierten Text fett, kursiv und unterstrichen schreiben. Außerdem kann man Aufzählungen und einen Link erstellen. Weiters lässt sich der Text im Code-Textformart (Courier New) schreiben.

8.2 Menüpunkte

In den folgenden Unterkapiteln wird beschrieben, welche Funktionen die einzelnen Menüpunkte der Menüeinträge im Hauptformular haben.

8.2.1 Datei

Menüeintrag Datei.

Neu
Bevor eine neue Aufgabe angelegt wird, muss nachgefragt werden ob die aktuell bearbeitete Aufgabe abgespeichert werden soll. Es gibt 2 Arten von Beispielen die man erstellen kann:

- Hamster – Beispiel

  Beim Erstellen eines Hamster-Beispieles wird zusätzlich ein Hamster-Dialog geöffnet. In diesem Hamster-Dialog ist es möglich die Hamsterwelt zu editieren.

  Beim Editieren der Hamster-Welt ist jedoch zu beachten, dass maximal neun Körner auf einem Feld liegen dürfen.

- E/A – Beispiel

  Beim erstellen eines E/A-Beispiels wird im Hauptformular nur die Listbox mit dem Programm und das Textfeld mit der Beschreibung der Aufgabe angezeigt.

Aufgabe öffnen…
Es öffnet sich ein Datei-Öffnen-Standarddialog, bei dem man eine Aufgaben-Datei öffnen kann.

Aufgabe speichern…
Es öffnet sich ein Datei-Speichern-Standarddialog, mit dem man eine Aufgaben-Datei speichern kann.

Template öffnen…
Es wird eine Aufgabe geöffnet. Diese Aufgabe wird aber nicht als Aufgabe zum bearbeiten geöffnet, sondern wird als neue Aufgabe erstellt. D.h. diese Aufgabe muss unter einem neuen Namen gespeichert werden.

Testen
Um das Programm testen zu können, müssen zuvor Testfälle im Ergebnisüberprüfungsdialog angelegt werden. Wenn Testfälle vorhanden sind, wird das gleiche abgearbeitet wie im ProgrammierTrainer Kapitel 7.2.3 „Test“ beschrieben ist.

Beenden
Vor dem beenden des Programm wird nachgefragt, ob die aktuelle Aufgabe noch gespeichert werden soll. Anschließend wird das komplette Programm, inkl. Aller Dialoge, beendet.

8.2.2 Aufgabe
Menüeintrag Aufgabe.

Ergebnisüberprüfung
Es wird der Ergebnisüberprüfungsdialog (siehe 8.3) angezeigt.

Pseudo-Code erzeugen
Es muss vom Benutzer eine Datei ausgewählt werden, welche beschreibt wie die Programmanweisungen in einen Pseudo-Code übersetzt werden. Auf diese Weise ist es möglich, den Python-Code in verschiedene Pseudo-Codeformen zu übersetzen, wie formale Sprache oder auch eine Programmiersprache als Pseudo-Code.

Zufalls-Anweisungen erzeugen
Es werden die „Fake“-Anweisungen automatisch generiert. Dazu werden die vorhandenen Anweisungen analysiert und die logischen und operationalen Operatoren verändert[5] und als neue „Fake“-Anweisung eingefügt.
Es werden in einem vorhandene Statement, z.B. `ifs` oder `whiles`, vorhandene „`and`“ durch „`or`“, „==“ durch „`!=`“ und „>“ durch „<“ und jeweils umgekehrt, ersetzt.

Hinweise verfügbar
Mit dieser Option kann festgelegt werden, ob der Benutzer beim Lösen der Aufgabe Hinweise (siehe 7.2.4) bekommen kann.

Musterlösung
Mit dieser Option kann festgelegt werden, ob der Benutzer die Musterlösung (siehe 7.2.4) einsehen darf oder nicht.

8.2.3 Dienstprogramme
Die Dienstprogramme sind für allgemeine Aufgaben zuständig und sind nur indirekt mit der Erstellung der Aufgabe verbunden.

Verzeichnis Indizieren
Mit diesem Dienstprogramm ist es möglich, automatisch eine Datei (siehe 7.5) für einen Server, zu dem bestimmte ProgrammierTrainer-Applikationen verbinden, zu erstellen, um nicht alle Aufgaben eines Verzeichnisses manuell in die Datei einzutragen. Hierzu muss aber der Benutzer dieses Dienstprogramms den Server betreiben, d.h. dieses

[5] Es werden lediglich logische durch andere logische und operationale durch andere operationale Operatoren verändert.

Dienstprogramm muss am Server laufen, um ein dort liegendes Verzeichnis zu indizieren. Es wird somit eine Datei des folgenden Formats erzeugt:

```
[PTAufgaben]
Aufgabe1 # /Pfad/Aufgabe1.pt
Aufgabe2 # /Pfad/Aufgabe2.pt
```

Es werden alle Dateien mit der Endung „.pt“ geöffnet, der Titel extrahiert und zusammen mit der absoluten Pfadangabe[6] in die Datei eingetragen. Um das www-Rootverzeichnis festzulegen und nachher auf den absoluten Pfad zu schließen, muss der Benutzer des Dienstprogramms zuerst sein www-Rootverzeichnis auswählen und so wird danach, wenn das Verzeichnis mit den Aufgaben bekannt ist, der Teil des Pfades, der mit dem www-Rootverzeichnis identisch ist, mit „/“ substituiert.
Anmerkung: Die Installation und alle zugehörigen Arbeiten zur Aufstellung eines Webserver fällt nicht unter die Aufgabe der Autoren des ProgrammierTrainers.

8.2.4 Hilfe

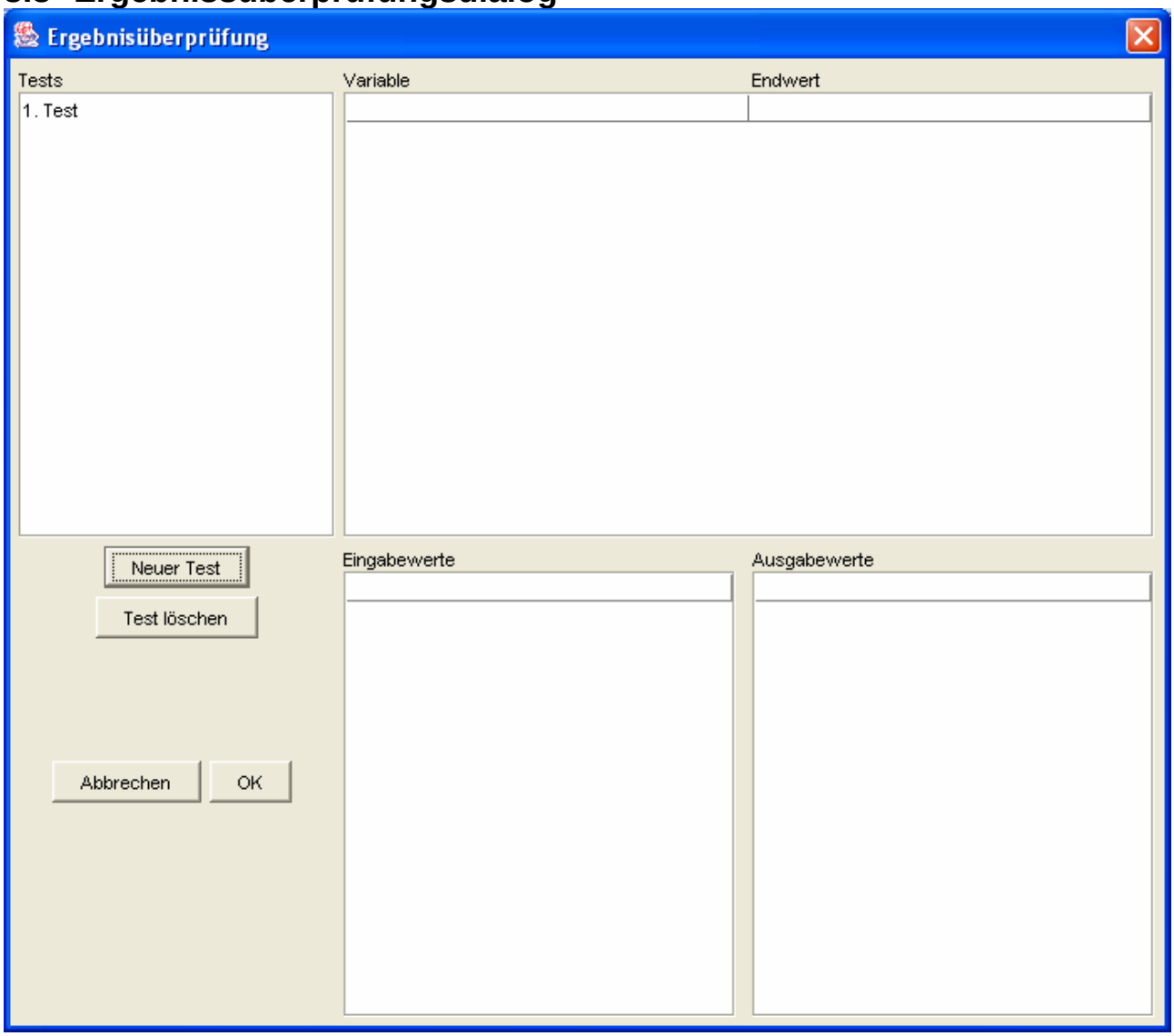
Menüeintrag Hilfe.

Hilfe zu ExampleBuilder
Es wird der Hilfedialog (siehe 7.8) mit einer HTML-Seite, deren Inhalt die Hilfe zum ExampleBuilder beinhaltet, angezeigt.

Info
Es wird der Infodialog (siehe 8.4) angezeigt.

[6] Die Pfadangabe bezieht sich auf den Pfad am Webserver. D.h. das Verzeichnis welches indiziert wird, am Webserver liegen muss.

8.3 Ergebnissüberprüfungsdialog



Shortcuts

Tasten	Funktion
Entf	löscht die markierte Zeile aus der Eingabe-, Ausgabe- oder Variablenendwertetabelle
Enter	fügt eine neue Zeile in die Eingabe-, Ausgabe- oder Variablenendwertetabelle ein

Mit Hilfe der Ergebnisüberprüfung können für die Aufgabe Testwerte erstellt werden. Es kann für eine Variable ein Endwert festgelegt werden. Nach dem Ablauf des von Benutzer entworfenen Programms werden die Werte mit denen in diesem Dialog eingetragene Endwerte vergleichen (genaueres siehe 7.2.3 „Test“). Weiters können auch Eingabe- und Ausgabewerte definiert werden, womit dem abgelaufenen Programm ein Benutzer „simuliert“ wird.

Beispiel
Für das im Folgenden stehende Programm wären der Testfall Nr. 1 und Testfall Nr. 2 sinnvoll.

```
Variable x vom Benutzer einlesen
multiplizieren der Variable x mit 2 und in y speichern
y ausgeben
```

Testfall Nr. 1:

Variable : y	Endwert : 6
Eingabewerte: 3	Ausgabewerte: 6

Testfall Nr. 2:

Variable : y	Endwert: 12
Eingabewerte: 6	Ausgabewerte: 12

Anlegen von Tests und den Testwerten
Um einen neuen Test anzulegen muss der Button „Neuer Test“ betätigt werden. Alle angelegten Tests werden in der „Tests“-Listbox angezeigt. Wenn ein Test markiert wird, werden die dazu gehörigen Daten in den jeweiligen Wertetabellen angezeigt. Durch einen Doppelklick in eine Zelle kann der Wert dieser verändert werden. Anschließend muss mit Enter bestätigt werden um die Änderung abzuschließen. Ist eine Zelle markiert, kann durch betätigen von Enter eine neue Zeile in der jeweiligen Wertetabelle eingefügt werden.

8.4 Infodialog

Es werden Programminformationen (Autoren, Versionsnummer, etc.) angezeigt.

9 Zusatzmodul – Hamster-Dialog (MR)

Dieser Teil der Diplomarbeit entstand erst während der Implementierungsphase. Zu dieser Zeit wurde uns nämlich das volle Ausmaß der Vorteile eines Hamster-Panels klar. Deshalb äußerte unser betreuender Lehrer, Herr Dipl.-Ing. Harald Haberstroh, den Wunsch das Panel zu einem eigenständigen Dialog weiter zu entwickeln.

Ein vom ProgrammierTrainer unabhängiger Hamster-Dialog bringt enorme Vorteile mit sich:

- Es muss nicht mehr der ganze ProgrammierTrainer geladen werden
- Das Einsatzgebiet des Hamsters ist somit uneingeschränkt:
  Eine mögliche Anwendung ist die Einbindung des Hamsters als Steuerelement in andere Java-Programme
- Der Hamster-Dialog kann aus JYTHON heraus direkt angesprochen werden
- Dem Programmieranfänger werden die Auswirkungen seiner Programmiertätigkeiten sofort angezeigt

9.1 Einbindung in Jyhton

Um den Hamster-Dialog in der Jython-Konsole verwenden zu können, sind verschiedene Vorbereitungsschritte notwendig.

Folgende Schritte sind notwendig:

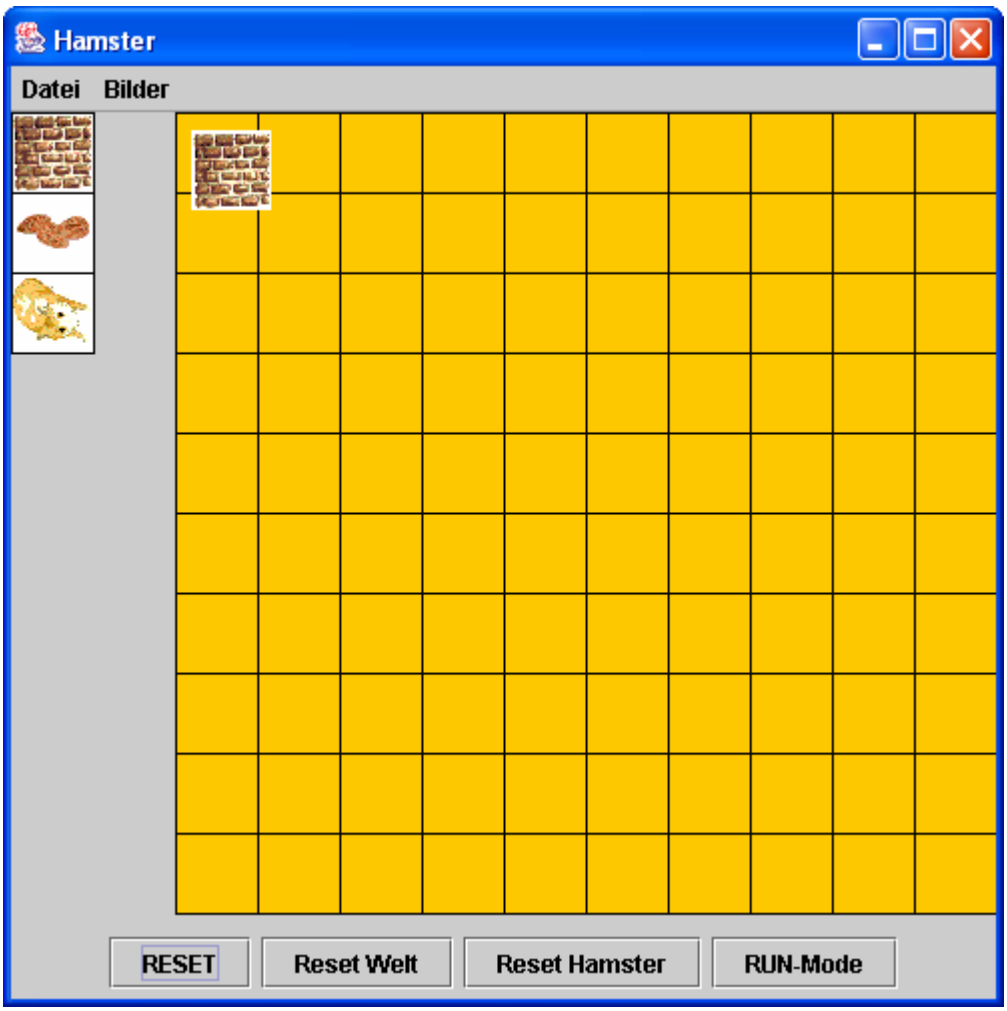
1. Die Umgebungsvariable CLASSPATH muss um das Java-Archiv des Hamster-Dialoges erweitert werden.
2. Im nächsten Schritt muss die Jython-Konsole gestartet werden.
3. Ist die Jython-Konsole gestartet, so muss jetzt nur noch die Hamster-Dialog-Klasse importiert werden.

   Dies geschieht mit dem folgenden Kommando:

   ```
   >>> from hamster import *
   ```

9.2 Die Oberfläche

Die folgende Abbildung zeigt den Hamster-Dialog während des Erstellens einer neuen Hamster-Welt. Der Benutzer verschiebt in diesem Moment ein Mauer-Element in die Welt.



9.2.1 Menüpunkte

Die Menüpunkte sind folgendermaßen gegliedert:

→ Datei
  → Öffnen (Shortcut: Strg-O)
  → Beenden

→ Bilder
  → laden

Aufgabendatei öffnen
Mit diesem Menüpunkt kann eine Aufgaben-Datei geladen werden. Das Format der Aufgabendatei entspricht dem Format der Hamster-Aufgabendateien des ProgrammierTrainers. Somit können die bereits bestehenden Aufgaben verwendet werden.
Weiters können diese Dateien mit dem ExampleBuilder des ProgrammierTrainers erstellt und verschlüsselt werden.

9.2.2 Bilder laden

Mit diesem Menüpunkt ist es dem Benutzer möglich, die Standard-Bilder des Hamster-Dialoges durch eigene, individuelle Bilder zu ersetzen. Wenn man diesen Menüpunkt auswählt, öffnet sich ein Datei-Öffnen-Dialog zur Auswahl des Verzeichnisses, in welchem sich die Bilder befinden.

Damit die Standard-Bilder des Dialoges ersetzt werden können, müssen die Dateinamen der Bilder folgender Namensvereinbahrung entsprechen.

Dateiname	Beschreibung
`mauer.gif`	Das Bild des Begrenzungselementes. Das Standard-Element des Hamster-Dialoges ist eine Mauer.
`korn.gif`	Das Bild des Elements, welches von der Figur gesammelt werden soll. Das Standard-Element des Hamster-Dialoges ist ein Korn.
`hamstero.gif`	Das Bild der Figur mit der Blickrichtung nach rechts. Das Standard-Element für die Figur ist im Hamster-Dialog der Hamster.
`hamstern.gif`	Das Bild der Figur mit der Blickrichtung nach oben.
`hamsterw.gif`	Das Bild der Figur mit der Blickrichtung nach links.
`hamsters.gif`	Das Bild der Figur mit der Blickrichtung nach unten.

9.3 Schaltflächen

Der Hamster-Dialog stellt folgende Schaltflächen bereit:

RESET
Reset Welt
Reset Hamster
Run-/Edit-Mode

9.3.1 RESET

Betätigt der Benutzer diese Schaltfläche, so wird die Hamster-Welt mit den Standardwerten initialisiert. Je nach dem, ob der Benutzer die Hamster-Welt aus einer Aufgabendatei geladen hat oder nicht, werden verschiedene Standardwerte geladen.

Situation	Standardwert
Der Benutzer hat die editierte Hamster-Welt aus einer Aufgabendatei geladen.	Die Hamster-Welt der Aufgabendatei.
Der Benutzer hat die Hamster-Welt neu erstellt	Die Hamster-Welt wird geleert.

Durch diese Aktion wird automatisch in den Edit-Mode gewechselt. Nähere Informationen zum Run- und Edit-Mode finden Sie in den Abschnitten 9.4.1 und 9.4.2.

9.3.2 Reset Welt

Betätigt der Benutzer diese Schaltfläche, so wird die Hamster-Welt geleert. Durch diese Aktion wird automatisch in den Edit-Mode gewechselt. Nähere Informationen zum Run- und Edit-Mode finden Sie in den Abschnitten 9.4.1 und 9.4.2.

9.3.3 Reset Hamster

Betätigt der Benutzer diese Schaltfläche, so wird der Hamster aus der Hamster-Welt entfernt. Durch diese Aktion wird automatisch in den Edit-Mode gewechselt. Nähere Informationen zum Run- und Edit-Mode finden Sie in den Abschnitten 9.4.1 und 9.4.2.

9.3.4 Run-/Edit-Mode

Diese Schaltfläche dient dem Benutzer zum Umschalten zwischen Run- und Edit-Mode. Die Beschriftung dieser Schaltfläche entspricht immer der Bezeichnung des Modus, der

gerade NICHT aktiv ist. Nähere Informationen zum Run- und Edit-Mode finden Sie in den Abschnitten 9.4.1 und 9.4.2.

9.4 Bedienung des Hamster-Dialogs

Grundsätzlich kann der Hamster-Dialog in zwei verschiedenen Bedienungsmodi betrieben werden:

- Edit-Mode
- Run-Mode

9.4.1 Edit-Mode

Im Edit-Mode ist es dem Benutzer möglich seine Welt selbst zu erstellen. Befindet sich der Hamster-Dialog in diesem Mode, so werden auf der linken Seite des Dialoges die drei Grundbauelemente angezeigt. Diese Elemente können vom Benutzer mittels Drag&Drop in die Hamster Welt verschoben werden.
Hat der Benutzer den Hamster in der Hamster-Welt platziert, so kann er jetzt auch seine Blickrichtung ändern. Um die Blickrichtung zu ändern ist ein einfacher Mausklick mit der linken Maustaste auf das Hamster-Element, welches sich in der Hamster-Welt befindet, nötig.
Weiters kann der Benutzer auch Korn- und Mauerelemente in die Hamster-Welt verschieben.

Achtung: Wie schon beim Example-Builder erläutert können nur maximal neun Korn-Elemente auf das selbe Feld platziert werden!

9.4.2 Run-Mode

Im Run-Mode ist es dem Benutzer möglich, den Hamster in einer bereits existierenden Welt zu bewegen. In diesem Modus kann der Benutzer die Hamster-Welt nicht mehr verändern.

9.5 Methoden-Übersicht

Um den Hamster in der Welt bewegen zu können, sind eine Reihe von Methoden notwendig. Die folgende Tabelle soll ihnen als Übersicht über die verfügbaren Methoden dienen.

Methodenname	Return-wert	Beschreibung	Mögliche Fehler
`setAppPath(String sPath)`	`void`	Ändert den Pfad, in welchem die Bilder gesucht werden.	Keine
`fileOpen(String sPath)`	`bool`	Lädt eine neue Aufgaben-datei. Ist das Laden erfolgreich, so liefert diese Methode „1“ zurück. Schlägt das Laden fehl, so liefert diese Methode „0“ zurück.	Die Datei ist keine gültige Aufgabendatei.
`vor()`	`void`	Rückt die Figur um einen Platz nach vor.	Eine Mauer oder ein Spielfeldrand stehen im Weg
`linksUm()`	`void`	Dreht die Figur um 90° nach links.	Keine
`vorneFrei()`	`int`	Überprüft, ob ein weiterer Schritt vorwärts möglich ist.	Keine
`vornFrei()`	`int`	Entspricht `vorneFrei()`	Keine
`kornDa()`	`int`	Überprüft, ob am Platz, wo sich der Hamster befindet, ein Korn liegt.	keine
`maulLeer()`	`int`	Überprüft, ob das Maul des Hamsters leer ist.	keine
`backenLeer()`	`int`	entspricht `maulLeer()`	keine
`gib()`	`void`	Der Hamster legt ein Korn ab	Es befinden sich bereits 9 Körner auf diesem Platz
`nimm()`	`void`	Der Hamster nimmt ein Korn auf	Auf dem aktuellen Platz befindet sich kein Korn

Weiters stehen folgende Konstruktoren zur Verfügung:

`dlgHamster();`

Dieser Konstruktor erstellt einen Hamster-Dialog mit einer Standardgröße von 10x10 Feldern.

`dlgHamster(int iSpalten, int iZeilen);`

Dieser Konstruktor erstellt einen Hamster-Dialog mit der übergebenen Anzahl von Feldern.

Beispiel:

```
>>> from hamster import *
>>> dlg = dlgHamster(5,4)
```

In diesem Fall wird ein Dialog mit 5 Spalten und 4 Zeilen erstellt.

9.6 Exceptions

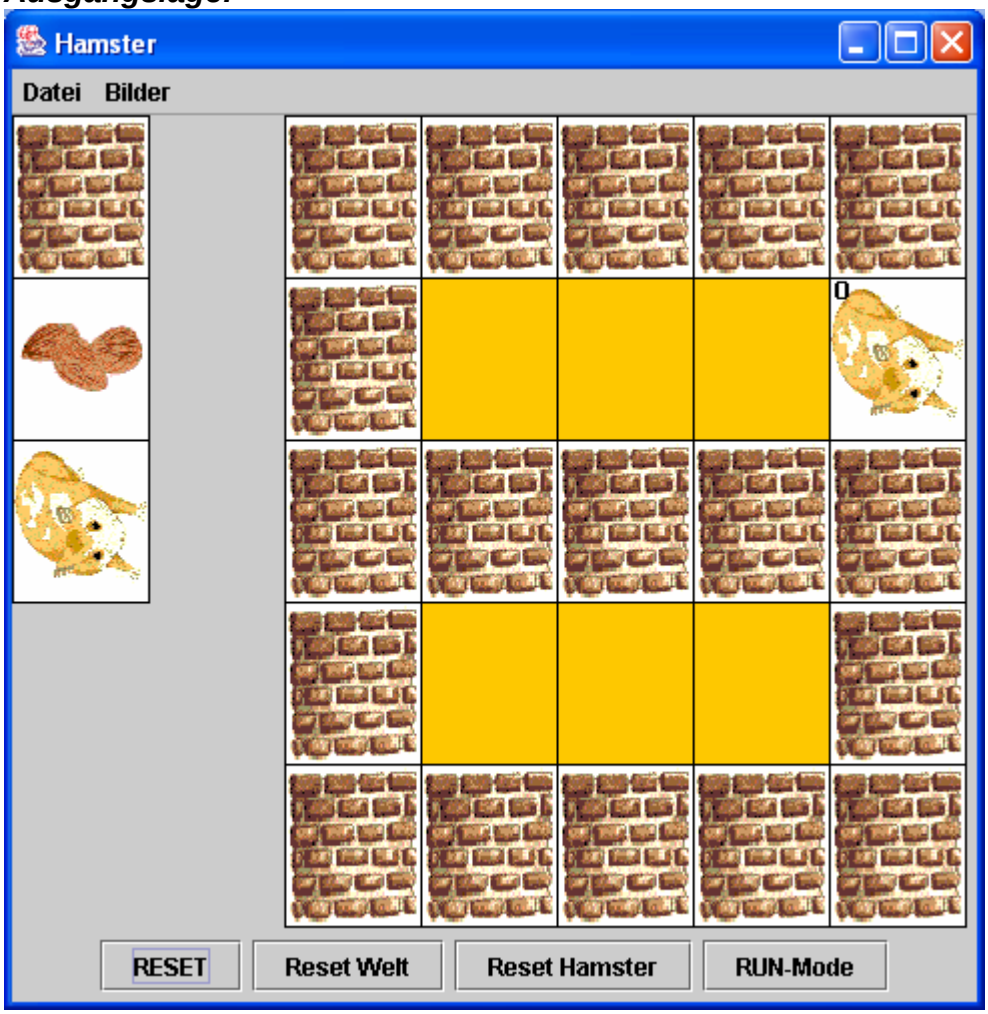
Während der Benutzer den Hamster in der Welt umher bewegt, können die unterschiedlichsten Fehlersituationen auftreten. Damit der Benutzer diese Fehler abfangen und kategorisieren kann, wurden folgende Exceptions für den Hamster entwickelt.

9.6.1 MauerException

Eine MauerException wird ausgeworfen, wenn der Benutzer versucht, den Hamster gegen eine Mauer oder einen Spielfeldrand zu bewegen.

Beispiel:
Folgendes Beispiel soll diese Situation veranschaulichen.

Ausgangslage:



Aktion

Beim Versuch den Hamster in dieser Situation vorwärts zu bewegen, würde eine MauerException auftreten. Dasselbe würde auch passieren, wenn der Benutzer den Hamster zuerst um 90° nach links dreht und erst dann nach vor bewegt.

Mögliche Vorgehensweise
Will der Benutzer verhindern, dass das laufende Programm wegen einer solchen Exception beendet wird, muss er folgendermaßen vorgehen.

Programmausschnitt (Python):
```
>>> from hamster import *
>>> dlg = dlgHamster()
>>> try:
>>>    # Programmcode (z.B.: dlg.vor())
>>> exception MauerException, e:
>>>    print e.getMessage();
```

9.6.2 KornException

Eine KornException wird ausgeworfen, wenn es Probleme mit den Körnern gibt. Eines davon könnte sein, wenn der Benutzer versucht ein Korn von einem Feld aufzunehmen, auf dem sich in Wirklichkeit jedoch keines befindet.
Weiters kann es zu einer KornException kommen, wenn der Benutzer versucht mehr als neun Körner auf einem Feld zu positionieren.

Beispiel:
Folgendes Beispiel soll diese Situation veranschaulichen.

Ausgangssituation:
Der Benutzer hat eine neue Hamster-Welt erstellt und den Hamster auf ein leeres Feld positioniert.

Aktion:
Versucht nun der Hamster ein Korn aufzunehmen, so wird eine KornException ausgeworfen. (Selbes würde auch passieren, wenn der Hamster versuchen würde, mehr als neun Körner auf ein Feld zu positionieren)

Mögliche Vorgehensweise
Will der Benutzer verhindern, dass das laufende Programm wegen einer solchen Exception beendet wird, muss er folgendermaßen vorgehen.

Programmausschnitt (Python):
```
>>> from hamster import *
>>> dlg = dlgHamster()
>>> try:
>>>    # Programmcode (z.B.: dlg.nimm(), dlg.gib())
>>> exception KornException, e:
>>>    print e.getMessage();
```

10 Verschlüsselungsmodul (MR)

Im folgenden Abschnitt gehe ich näher auf die Anforderungen des Verschlüsselungsmoduls ein. Es wird beschrieben für welches Verschlüsselungsverfahren wir uns entschieden haben und was unsere Gründe dafür waren. Weiters beschreibt dieses Kapitel die genaue Vorgangsweise des Verschlüsselns und die Ersetzbarkeit dieses Moduls.

10.1 Verfahrensauswahl

Im nachfolgenden Abschnitt wird erläutert, welche Überlegungen auf die Auswahl des Verschlüsselungsverfahrens einen großen Einfluss ausübten.

10.1.1 Ausgangslage

Mit dem ProgrammierTrainer soll es möglich sein Tests zu veranstalten. Die in der Aufgabendatei stehende Musterlösung darf von den Benutzern deshalb nicht eingesehen werden.
Aufgrund dessen ist eine Verschlüsselung dieser Dateien erforderlich. Da es jedoch eine Vielzahl an Verschlüsselungsalgorithmen gibt, mussten wir einen, unseren Anforderungen entsprechenden, auswählen.

Folgende Anforderungen wurden an den Verschlüsselungsalgorithmus gestellt:

- Der Verschlüsselungsalgorithmus muss nicht unbedingt wieder neu erfunden werden. Es genügt, die Auswahl eines bereits bestehenden Algorithmus.
- Die Aufgabendatei soll innerhalb einer akzeptablen (kurzen) Zeit nicht geknackt werden können
- Die Ver- und Entschlüsselung soll nicht zu aufwändig sein, da dadurch das Zeitverhalten des Systems enorm beeinflusst werden könnte.
- Die Verschlüsselung soll als eigenständiges Modul existieren, damit dieses bei Bedarf jederzeit durch ein anderes Verschlüsselungsmodul ersetzt werden kann.

Aus diesem Grund ist unsere Entscheidung auf den DES (Data Encryption Standard) – Algorithmus gefallen. Die folgenden Punkte sollen zur Begründung unserer Entscheidung dienen:

- DES ist ein standardisiertes Verschlüsselungsverfahren, welches teilweise von der Hardware unterstützt wird. Dies hat zur Folge, dass die Ver- und Entschlüsselung nur wenig Zeit in Anspruch nimmt.
- DES ist ein sicheres Verschlüsselungsverfahren.
- Die verschlüsselten Dateien können ohne bekannten Verschlüsselungs-Schlüssel, innerhalb einer akzeptablen Zeitspanne, von den Schülern nicht geknackt werden.
- Im Allgemeinen nicht sehr anfällig auf Hack-Angriffe.
  Begründung:
  Es gibt 2^{56} (~ 70 Billiarden) verschiedene Schlüssel. Wenn pro Sekunde eine Million Überprüfungen durchgeführt werden können und im Durchschnitt die Hälfte der möglichen Schlüssel ausprobiert werden, dann dauert dies mehr als 1000 Jahre. Bei 10000 solcher Chips, in einem Parallelcomputer vereinigt, würde man ca. 40 Tage brauchen.

10.1.2 Allgemeines zu DES

Folgende Informationen stammen von Reinhard Wobst aus dem Buch „Abenteuer Kryptographie“.

Der Data Encryption Standard, ist ohne Frage das am besten untersuchte kryptografische Verfahren. Obwohl der Verdacht besteht, die NSA könnte in DES eine Hintertür eingebaut haben, ist bis heute kein praktisch verwertbarer Angriffspunkt gefunden worden. Die Sicherheit von DES hat sich jedoch im Laufe der Zeiten geändert, da der Brute-Force-Angriff heutzutage technisch möglich geworden ist.

10.1.3 Wie entstand DES?

Vom National Bureau of Standards (NBS, heute NIST) wurde 1973 eine öffentliche Ausschreibung zum Entwurf eines einheitlichen, sicheren Verschlüsselungsalgorithmus gestartet. Durch die Entwicklung und Verbreitung von Rechentechnik und Kommunikation war ein allgemein zugängliches und sicheres Verfahren notwendig geworden. Das Echo auf diese Ausschreibung zeigte ein sehr starkes Interesse an einem solchen Standard. Mehr als das kam leider nicht dabei heraus. Nicht eine der Einsendungen genügte auch nur annähernd den gestellten Anforderungen!

Erst auf eine zweite Ausschreibung 1974 hin reichte ein IBM-Team einen brauchbaren Vorschlag ein. Er basierte auf dem bei IBM betriebenen Projekt Lucifer, nach dem auch mindestens ein Algorithmus benannt wurde.

Da von nun an die Kompetenz der NBS nicht mehr ausreichte, wurde die NSA herangezogen. Nach reichlichen Untersuchungen durch die NSA, wurde DES freigegeben.

DES wurde Ende 1976 zum offiziellen Verschlüsselungsstandard erklärt. Das Verfahren ist für normale Geheimhaltung gedacht, nicht zum Schutz von Informationen höchster Sicherheitskategorie.

Um DES gibt es jedoch einige Gerüchte, welche eine sehr starke Beweiskraft haben. Es wird nämlich behauptet, dass die ursprüngliche Schlüssellänge von 128 Bit von der NSA auf 56 Bit reduziert wurde. Der Grund weshalb dies angeblich geschah ist, dass DES für die NSA knackbar bleibt.

10.1.4 Eigenschaften von DES

Die folgenden Punkte dienen zur Beschreibung der Eigenschaften von DES:

- Schlüssellänge von 56 Bit
- es werden jeweils 64 Bit Klartext in 64 Bit Geheimtext überführt; dies geschieht
- in 16 schlüsselabhängigen transformationen.
- zu Beginn erfolgt eine Anfangstransposition
- zum Abschluss wird eine zur Anfangstransposition inverse Transposition durchgeführt.

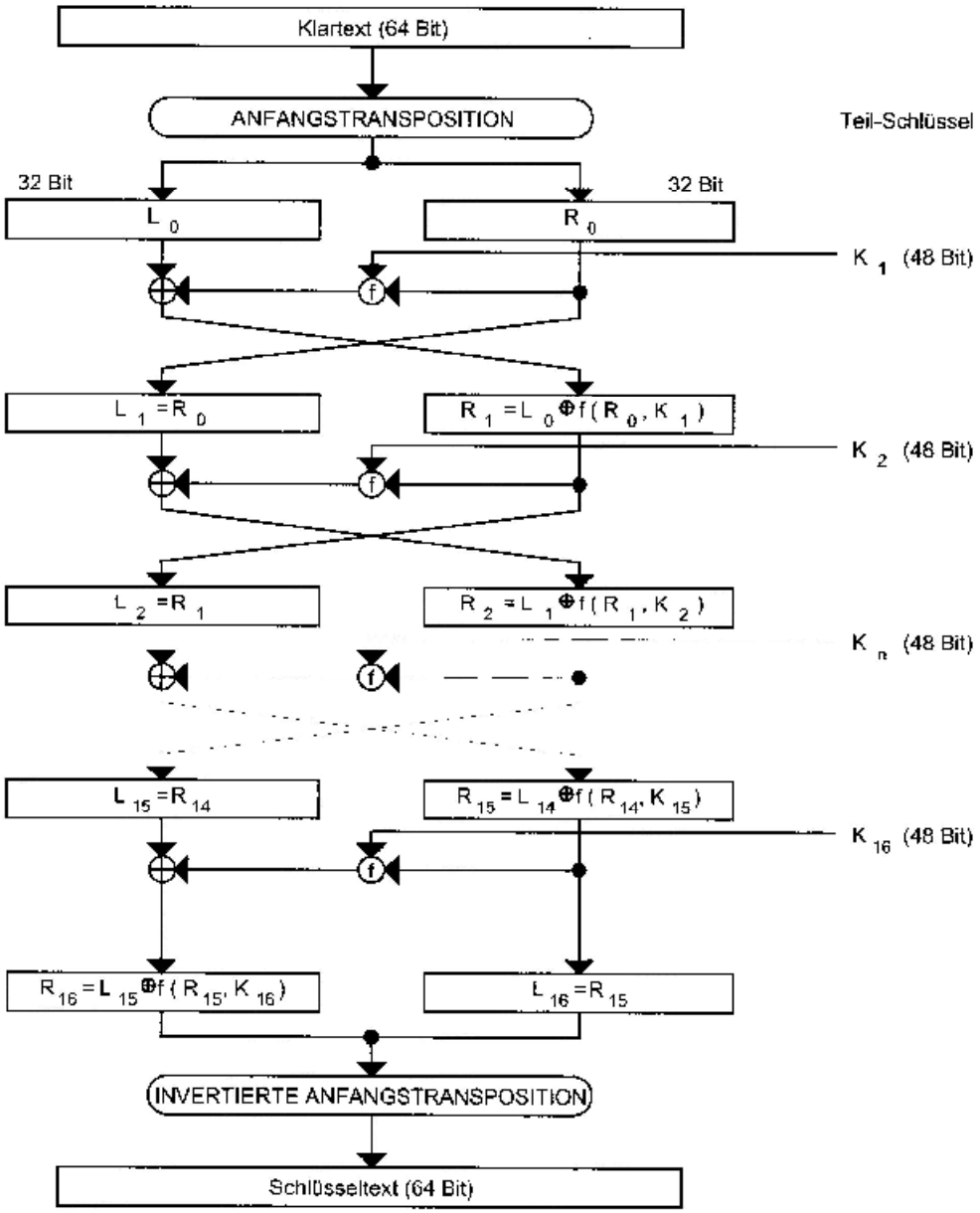
10.1.5 Wie sicher ist DES?

Auf diese Frage kann wohl niemand eine erschöpfende, offiziell bekannte Antwort geben. Es sind nur drei Angriffsmöglichkeiten gegen DES bekannt:

- Brute Force
- Differenzielle und
- Lineare Kryptanalyse

10.1.6 Grafische Darstellung der Ver- und Entschlüsselung:

Folgende Abbildung soll den Ver- und Entschlüsselungsvorgang grafisch veranschaulichen.



10.1.7 Beispiele zu DES

In diesem Abschnitt wird der Verschlüsselungsvorgang anhand eines Beispieles veranschaulicht.

Entschlüsselte Aufgaben-Datei:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<aufgabe name="Logik-Labyrinth Level 2">
   <Beschreibung><html>
  <head>

  </head>
  <body>
    Finden Sie das Korn.<br></br>Verwenden Sie f&#252;r jeden Aktionsbereich eine eigene
    Methode.<br></br>Hinweis: Halten Sie sich Rechts
  </body>
</html>
</Beschreibung>
   <Welt>
MMMMMMMMMMMMMMMMMMM
                   M
MM MMMMMM MMMMMMMM M
M     M            M
MMM MMM MMM MMM MMMM
M M M M M M M      M
M M M M M M MMMMMM M
M M       M M    M M
M MMM MMM M M M MM M
M   M M   M M M    M
MMM M M MMM M M MM M
      M M   M M MM M
MMMMMMM MMM M M    M
M           MMM MMMM
MMM MM MMMM M      M
M    M       MMMMMMMM
M MM M MMMMMM M  M M
M MM M M    M M  M M
M  M M             1
MM M MMMMMMMMMMMMMMM
</Welt>
   <HamsterInfo>
      <Koerner>0</Koerner>
      <Position>0,1</Position>
      <Richtung>0</Richtung>
   </HamsterInfo>
   <Programm>
       def sucheKorn(): # Funktion suchekorn (Parameter: )
          istatus = 0 # setze istatus auf 0
          while not kornDa(): # Solange nicht Korn ist da mache
             vorwaerts() # vorwaerts()
             if istatus < 2: # Wenn istatus  kleiner  2 dann
                rechts() # rechts()
             else: # sonst:
                linksUm() # Um 90 Grad nach links drehen
             istatus += 1 # istatus  um ( 1) erhoehen
             if istatus == 4: # Wenn istatus  gleich  4 dann
                istatus = 0 # istatus  auf  0 setzen
       def vorwaerts():  # Funktion vorwaerts (Parameter: )
          while vorneFrei():  # Solange vorne frei mache
             vor()  # Geh einen Schritt nach vor
       def rechts():  # Funktion rechts (Parameter: )
          i = 0  # i  auf  0 setzen
          while i < 3: # Solange i  kleiner  3 mache
             linksUm() # Um 90 Grad nach links drehen
             i+=1 # i um (1) erhoehen
       sucheKorn() # sucheKorn()
   </Programm>
   <Optionen>
      <Musterloesung>Nein</Musterloesung>
      <Zeit>Nein</Zeit>
   </Optionen>
</aufgabe>
```

Dieselbe Datei Verschlüsselt:

130-105-64-169-121-160-219-251-138-187-236-31-104-14-232-104-24-212-228-156-227-40-175-8-127-28-11-
209-237-142-64-60-206-105-24-116-163-220-233-209-136-42-149-65-0-161-99-214-235-249-123-242-104-125-
111-214-175-103-53-207-183-52-246-255-105-145-134-133-114-88-33-38-180-77-111-42-190-171-255-29-145-
37-197-184-61-71-190-15-111-230-241-25-207-249-118-149-248-16-21-234-249-246-181-105-15-176-60-136-
70-57-209-24-233-172-73-152-243-16-32-251-106-191-133-186-150-240-145-6-5-23-12-204-19-39-83-228-
124-225-66-169-191-208-34-136-14-106-172-170-231-180-113-33-143-173-133-235-7-200-154-83-194-20-243-
223-18-199-122-156-23-5-149-55-232-188-149-34-123-186-218-199-72-118-226-23-25-102-50-71-189-157-3-
254-128-159-247-177-161-55-207-67-110-234-231-114-152-4-4-1-214-36-88-242-73-165-163-240-141-1-39-
134-167-132-211-132-170-106-70-127-74-208-163-208-102-31-221-113-23-156-160-157-219-174-165-207-212-
26-171-99-142-150-218-32-125-112-118-75-160-123-2-188-74-136-121-10-103-18-133-117-24-87-172-85-238-
131-205-185-129-68-237-199-2-246-183-224-178-175-33-10-25-227-1-197-173-246-128-91-157-0-102-7-215-
233-173-52-27-199-130-5-16-92-197-222-59-185-186-195-184-139-113-16-154-248-76-206-98-247-239-148-
164-191-132-228-150-176-144-240-41-203-201-21-33-169-211-245-179-120-86-20-108-254-233-101-213-193-
213-203-91-95-96-46-243-164-37-33-91-105-192-203-243-164-37-33-91-105-192-203-64-255-98-212-192-11-
211-200-16-132-45-38-190-52-121-231-110-12-241-138-204-29-106-176-207-31-7-194-166-178-97-214-81-65-
151-75-88-226-24-214-176-97-162-254-169-49-64-191-42-105-49-187-165-145-19-195-16-132-45-38-190-52-
121-231-29-103-227-18-36-82-218-250-139-45-188-183-158-35-172-37-113-149-51-129-35-65-239-69-19-225-
38-17-63-15-253-207-220-111-250-65-174-110-212-188-110-12-241-138-204-29-106-176-220-111-250-65-174-
110-212-188-32-225-181-54-84-234-62-135-176-97-162-254-169-49-64-191-155-241-141-98-146-46-179-98-
53-220-77-77-133-39-187-228-190-204-230-34-100-231-3-96-77-100-12-149-86-251-192-1-232-138-83-81-
134-171-0-139-60-12-111-132-82-155-151-214-173-28-97-133-97-196-248-139-207-252-9-138-186-92-128-
202-117-98-124-217-126-23-75-176-117-98-124-217-126-23-75-176-107-232-47-15-33-253-194-15-164-16-64-
63-39-18-28-62-122-29-62-183-136-231-223-222-182-102-147-82-87-105-109-244-139-45-188-183-158-35-
172-37-53-220-77-77-133-39-187-228-94-5-16-130-55-58-130-239-125-74-155-2-84-82-164-167-75-29-53-53-
227-83-208-114-79-115-14-85-47-219-133-172-172-22-231-166-20-34-76-212-142-112-158-192-243-116-121-
150-90-237-222-113-11-176-232-54-109-192-105-224-31-214-85-128-190-23-188-195-206-218-189-118-183-
97-0-216-149-252-21-182-122-205-65-114-151-224-96-31-53-167-115-240-66-209-247-91-154-164-21-92-143-
7-225-164-226-10-15-86-64-211-247-56-103-51-149-123-151-204-163-82-16-132-45-38-190-52-121-231-122-
57-168-195-143-73-108-116-183-97-0-216-149-252-21-182-243-164-37-33-91-105-192-203-187-138-238-167-
68-221-205-136-246-202-83-194-26-242-167-24-139-92-213-184-107-137-211-34-242-196-33-212-117-160-
194-76-152-104-235-113-158-43-157-205-71-42-192-38-87-59-239-249-241-105-115-148-4-216-41-114-228-
114-197-65-19-2-250-115-44-209-180-234-89-128-41-137-130-119-134-123-69-95-81-109-159-130-189-133-
59-185-242-250-228-114-197-65-19-2-250-115-129-121-182-252-226-125-210-246-90-212-227-208-191-153-
60-218-104-196-9-55-32-167-227-17-24-244-218-244-61-42-163-137-127-119-41-192-8-143-28-185-101-170-
78-225-173-35-32-252-46-122-185-99-164-61-62-180-170-195-245-123-81-71-83-167-168-157-198-154-152-
84-203-186-116-23-175-18-50-188-182-162-226-70-253-177-95-50-19-209-97-42-217-7-176-200-226-217-44-
131-40-64-162-219-7-163-19-42-18-43-94-195-9-125-143-253-191-77-245-24-241-126-19-120-92-5-32-207-
184-99-190-133-123-192-230-233-242-37-29-5-239-46-133-230-188-102-39-195-121-167-221-158-218-43-75-
63-27-126-198-104-178-164-71-76-188-41-110-24-202-155-217-130-114-209-34-254-242-165-68-44-117-170-
51-39-204-238-225-222-4-172-175-155-106-184-66-160-133-167-254-184-247-105-32-77-203-120-213-150-
223-245-64-255-98-212-192-11-211-200-223-109-212-114-202-196-117-250-251-95-228-143-54-202-197-218-
59-74-45-25-223-18-196-185-226-147-138-124-224-127-20-176-24-236-20-99-67-201-10-7-0-135-207-253-
137-218-217-152-44-176-177-126-99-88-139-222-124-213-3-124-25-84-231-87-231-147-106-26-33-165-153-
225-254-249-91-72-25-204-199-28-64-255-98-212-192-11-211-200-85-230-56-137-169-40-7-229-95-29-209-
51-51-20-167-243-100-133-42-18-96-178-58-49-208-14-215-245-228-174-169-219-37-180-182-103-197-120-
99-103-7-101-112-161-252-103-213-97-44-111-104-94-57-76-120-247-77-193-10-165-149-70-152-253-75-149-
217-82-114-191-124-162-45-97-48-155-151-217-199-130-88-10-254-102-196-5-128-150-193-44-21-87-54-75-
115-89-216-204-226-193-221-15-220-84-45-60-242-120-122-59-171-249-40-49-135-231-25-249-174-92-133-
144-49-92-112-169-16-216-62-55-46-216-58-142-160-188-39-141-166-233-192-53-218-17-216-204-226-193-
221-15-220-84-192-170-56-25-190-66-194-213-254-189-133-21-134-228-32-231-214-47-25-102-153-55-52-
220-129-159-87-204-170-154-60-210-198-51-54-30-168-232-54-124-109-178-22-2-109-42-120-36-16-132-45-
38-190-52-121-231-29-5-239-46-133-230-188-102-99-83-242-131-63-233-116-239-129-159-87-204-170-154-
60-210-161-113-51-177-255-232-86-68-206-31-194-74-73-89-227-247-168-110-236-5-160-137-66-207-62-154-
51-130-204-50-204-118-214-92-126-167-90-199-30-193-220-122-54-88-79-200-99-18-68-98-44-150-34-19-41-
207-214-78-143-31-181-191-24-136-228-243-132-69-63-122-143-74-131-9-143-253-149-230-137-18-193-249-
220-80-71-233-127-219-219-3-235-250-59-14-159-5-10-219-230-182-235-136-216-158-206-197-84-47-38-164-
171-97-245-74-63-234-45-122-13-45-190-254-97-150-252-29-32-102-19-123-187-69-40-27-139-78-178-170-
159-164-239-254-192-28-246-130-32-93-108-173-23-169-217-13-136-111-251-228-54-195-51-87-159-201-171-
100-200-28-122-87-16-248-68-178-188-5-43-88-175-19-206-234-213-208-40-135-166-185-25-252-189-192-2-
151-46-60-98-92-109-182-24-80-19-141-61-242-125-242-238-18-123-135-190-245-166-211-102-255-216-160-
50-137-252-103-170-2-169-167-234-173-58-63-102-1-118-165-196-32-152-12-243-14-89-101-71-174-47-78-
110-243-170-10-76-28-106-117-33-207-24-15-38-219-86-151-222-82-225-222-112-86-135-133-164-179-42-38-
235-95-152-223-245-74-63-234-45-122-13-45-226-221-51-239-137-189-3-204-83-209-39-170-118-105-103-
179-81-55-134-34-68-123-162-200-67-11-71-156-0-120-109-176-100-104-225-97-196-200-176-175-158-89-
128-242-157-227-154-251-119-226-121-108-124-74-223-82-119-132-77-130-216-125-128-22-3-91-140-108-3-
179-118-114-116-224-75-37-86-82-77-210-227-190-131-19-162-144-141-254-149-33-118-174-236-35-207-209-
227-190-131-19-162-144-141-254-113-118-216-24-38-74-16-166-101-183-96-47-138-58-101-212-169-53-194-
39-81-181-75-74-95-232-196-25-187-12-106-1-232-223-227-246-22-52-79-224-112-125-206-9-8-199-80-85-
242-137-163-80-4-37-83-34-137-130-209-28-135-2-243-77-0-236-135-213-77-118-177-73-190-62-180-156-62-
213-95-130-64-255-98-212-192-11-211-200-219-128-218-82-210-115-153-124-218-197-4-205-130-141-33-9-
180-228-33-243-42-132-29-36-136-185-29-163-192-109-114-127-146-195-106-208-247-161-215-235-236-125-
115-126-188-113-171-54

10.2 Modulerklärung

Dieser Abschnitt des Dokuments befasst sich mit der Erklärung der programmtechnischen Lösung des Verschlüsselungsalgorithmus.

10.2.1 Konstruktor

Der Konstruktor dient der Initialisierung des Verschlüsselungsschlüssels. Der Verschlüsselungsschlüssel ist eine globale Variable namens **s** vom Typ **SecretKey.**

Basisschlüssel: „`programmiertrainer`"

Im Konstruktor muss deshalb die Zeichenkette des Basisschlüssels in eine Objektinstanz vom Typ SecretKey umgewandelt werden.

10.2.2 entschluessle

Diese Methode generiert unter Zuhilfenahme des Verschlüsselungsschlüssels aus dem übergebenen Schlüsseltext, den daraus resultierenden Klartext.

10.2.3 verschluessle

Diese Methode generiert unter Zuhilfenahme des Verschlüsselungsschlüssels aus dem übergebenen Klartext den daraus resultierenden Schlüsseltext.

10.3 Ersetzbarkeit

Eine der Anforderungen an die Verschlüsselung war dessen modulartige Einbindung in das Gesamtsystem. Es soll nämlich jederzeit die Möglichkeit bestehen, den aktuellen Verschlüsselungs-Algorithumus durch einen Neuen zu ersetzen.

Gründe dafür sind:

- Das Verschlüsselungsverfahren ist nicht mehr sicher genug.
- Ein anderes, wenn auch nicht so sicheres, Verfahren soll die Verschlüsselungsgeschwindigkeit erhöhen.
- Änderungen des Verschlüsselungsschlüssels sollen an einer einzigen Stelle (zentral) erfolgen.

Der Name des Verschlüsselungsmoduls ist **Krypt.class** und befindet sich im Package **programmiertrainer**. Um das Verschlüsselungsverfahren zu ändern, muss deshalb die Datei **Krypt.class** durch die neue Verschlüsselungsklasse ersetzt werden.

10.3.1 Aufbau der neuen Verschlüsselungsklasse

Grundsätzlich muss die Krypt-Klasse nur zwei Methoden zur Verfügung stellen:

Eine Methode zum Entschlüsseln

Prototyp:

```
String entschluessle (String sInhalt);
```

Diese Methode generiert aus dem übergebenen Schlüsseltext den daraus resultierenden Klartext.

Eine Methode zum Verschlüsseln

Prototyp:
```
String verschluessle (String sInhalt);
```

Diese Methode generiert aus dem übergebenen Klartext den daraus resultierenden Schlüsseltext.

11 Produktdaten (MR)

Die vom ProgrammierTrainer benötigten Daten sind auf vier verschiedene Dateiarten aufgeteilt.

Diese Dateien sind:

- Hamster-Aufgabendatei *
- E/A-Aufgabendatei *
- Optionen-Datei
- Zuordnungsdatei

Die mit * gekennzeichneten Datentypen können auch verschlüsselt werden. Dies ändert jedoch nichts an deren internen Aufbau.

11.1 Data Dictionary

Grundsätzlich werden alle vom ProgrammierTrainer benötigten Daten im XML-Format abgespeichert, um Dateioperationen einfacher handhaben zu können. Außerdem soll eine Verschlüsselung der Aufgabendateien möglich sein.
Die Aufzählungen in den unten beschriebenen Dateien stellen die einzelnen XML-Tags dar.

11.1.1 Hamster-Dateien

In dieser Datei wird ein Hamster-Beispiel gespeichert. Alle nötigen Informationen zu diesem Beispiel stammen von dieser Datei.

aufgabe **Attribut:** Name (Hier wird der Name der Aufgabe festgehalten)	
Beschreibung	Textuelle Beschreibung der Aufgabe. Hier sollen auch die Formatierungen im Text gespeichert werden. Deshalb wird die Aufgabenstellung im HTML-Format gespeichert.
Welt	Zeichenfolge, welche die Hamsterwelt festhält.
Hamster-Info	Zusätzliche Informationen zur Hamsterwelt
→ *Koerner*	Anzahl der Körner, die der Hamster zu Beginn im Maul hat.
→ *Position*	Position des Hamsters in der Hamsterwelt. Es wird ein genauer Punkt festgehalten. (zeile,spalte)
→ *Richtung*	Ist die Richtung, in welche der Hamster blickt. Mögliche Werte: 0,1,2,3 *0…rechts, 1…rauf, 2…links, 3…runter*
Programm	Hier werden die zur Verfügung stehenden Anweisungen gespeichert. Format: Python-Anweisung # Pseude-Code
Optionen	Hier werden die Einstellungen, welche die Aufgabenstellung benötigt, festgehalten
→ *Musterlösung*	Ja/Nein-Feld Hier wird festgehalten, ob es möglich sein soll, die Musterlösung einer Aufgabenstellung anzuzeigen.

→ *Zeit* [7]	Hier wird festgehalten, ob die Zeit, welche der Benutzer zum Erstellen eines Algorithmus benötigt, mitgestoppt bzw. eingeschränkt werden soll. Dieser Tag ist nur für zukünftige Erweiterungen vorhanden.

11.1.2 E/A-Dateien

In dieser Datei wird eine E/A-Aufgabe gespeichert. Außerdem wird die Musterlösung zur Aufgabenstellung mitgespeichert, um später das Ergebnis des Benutzers auf seine Korrektheit prüfen zu können.

aufgabe **Attribut:** Name (Hier wird der Name der Aufgabe festgehalten)	
Beschreibung	Textuelle Beschreibung der Aufgabe. Hier sollen auch die Formatierungen im Text gespeichert werden. Deshalb wird die Aufgabenstellung im HTML-Format gespeichert.
Programm	Hier werden die zur Verfügung stehenden Anweisungen gespeichert. Format: Python-Anweisung # Pseude-Code
Optionen	Hier werden die Einstellungen, welche die Aufgabenstellung benötigt, festgehalten
→ *Hinweis*	Ja/Nein-Feld Hier wird festgehalten, ob es möglich sein soll, dem Benutzer einen Hinweis beim Erstellen des Algorithmus auf die nächste Anweisung zu geben.
→ *Musterlösung*	Ja/Nein-Feld Hier wird festgehalten, ob es möglich sein soll, die Musterlösung der Aufgabenstellung anzuzeigen.
→ *Zeit 7*	Hier wird festgehalten, ob die Zeit, welche der Benutzer zum Erstellen eines Algorithmus benötigt, mitgestoppt bzw. eingeschränkt werden soll. Dieser Tag ist für spätere Erweiterungen vorhanden.
Tests	Hier werden die geplanten Standard-Tests festgehalten.
→ *Test**	Hier wird ein konkreter Test definiert.
→ *Attribut: id*	Identifikationsnummer des Tests.
→ *Variablen*	Hier werden Variablen mit deren Endwerten festgehalten.
→*Attribut: id*	Identifikationsnummer des Tests.
→ *Variable**	Hier wird die Initialisierung der Variable festgehalten.
→ *Attribut: name*	Enthält den Namen der Variable.
→ *Attribut: end*	Hier wird der Endwert dieser Variable festgehalten.
→ *Attribut: id*	Identifikationsnummer des Tests.

[7] Ist die Option „Zeit“ bereits in der Aufgabendatei festgehalten, so kann diese vom Benutzer nicht mehr geändert werden. Dies ist vor allem für die spätere Implementierung einer Testumgebung erforderlich.

→ *Ausgaben*	Hier werden alle Ausgaben für einen Test definiert.
→*Attribut: id*	Identifikationsnummer des Tests.
→ *Ausgabe**	Hier wird ein Ausgabewert definiert.
→*Attribut: id*	Identifikationsnummer des Tests.
→ *Eingaben*	Hier werden alle Eingaben für einen Test definiert.
→ *Attribut: id*	Identifikationsnummer des Tests.
→ *Eingabe**	Hier wird ein Eingabenwert für einen Test definiert.
→ *id*	Identifikationsnummer des Tests.

*) können mehrmals vorkommen

11.1.3 Optionen-Datei

In dieser Datei werden alle nötigen Einstellungen zum ProgrammierTrainer gespeichert.
Ort dieser Datei: `Home-Verzeichnis / programmiertrainer`

Optionen - Datei	
serverurl	URL des Aufgabenservers
aufgabenpfad	Hier wird festgehalten, wo die Aufgaben am Server liegen.
extedit	Wenn man zum Erstellen des Python-Codes einen externen Editor verwenden möchte, so kann man diesen hier eintragen.
zeit [7]	Hier wird festgehalten, ob die Zeit, welche der Benutzer zum Erstellen eines Algorithmus benötigt, mitgestoppt werden soll.
fensterpositionen(der Aufbau der einzelnen Child-Elemente wird im Punkt [8] *des Seitenkommentars beschreiben)*	Hier wird die Anordnung der einzelnen Fenster festgehalten. Pro Fenster werden die X ,Y-Werte und die Werte für Höhe und Breite festgehalten.
→ *ausgabe*	Werte für das Ausgabe-Fenster
→ *aufgabe*	Werte für den Aufgaben-Dialog
→ *aufgabeoeffnen*	Werte für den Aufgabe-Öffnen-Dialog
→ *einstellungen*	Werte für den Einstellungen-Dialog
→ *editor*	Werte für den Editor
→ *pthauptformular*	Werte für das Hauptformular des Programmiertrainers
→ *ebhauptformular*	Werte für das Hauptformular des Examplebuilders
→ *hilfe*	Werte für den Hilfe-Dialog
→ *ergebnis-ueberpruefung*	Werte für den Ergebnisüberprüfungsdialog
timeout	Hier wird festgehalten wie lange ein Programm in Sekunden laufen darf, bevor es abgebrochen wird. (siehe 7.6)
delay	Hier wird festgehalten wie viele Millisekunden zwischen zwei Ausführungen von Hamsteranweisungen vergehen sollen. (siehe 7.6)

[8] Diese einzelnen Elemente haben folgende Attribute:
h … die Fensterhöhe, w … die Fensterbreite,
x … X-Koordinate der Fensterposition, y … Y-Koordinate der Fensterposition

11.1.4 Zuordnungsdatei

In dieser Datei werden alle nötigen Einstellungen zum ProgrammierTrainer gespeichert.
Ort dieser Datei: `ProgrammierTrainer – Verzeichnis / zuordnungen`

Zuordnungsdatei	
Zuordnungen	Root-Element
→ *Zuordnung*	Hier werden die einzelnen Zuordnungen festgelegt.
→ *Ausdruck*	Ausgangsausdruck im regulären Ausdruck
→ *Umsetzung*	Übersetzter Ausdruck im regulären Ausdruck Alle im Ausgangsausdruck verwendeten „*“-Zeichen werden durchnummeriert und sind in der Umsetzung mittels „{laufende-Nummer}“ ansprechbar.

12 Benutzungskonzept (MR)

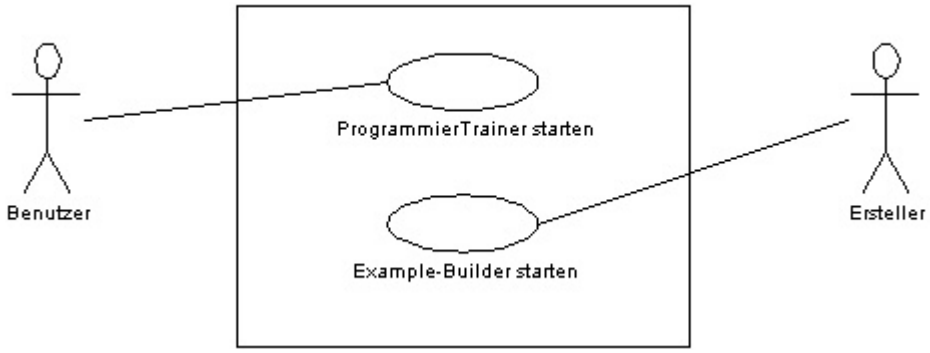
Im folgenden Kapitel möchte ich näher auf die Bedienung des ProgrammierTrainers und ExampleBuilders eingehen.

12.1 Use-Case-Diagramme

Zur übersichtlicheren Darstellung der Programmabläufe verwenden wir im folgenden Abschnitt USE-CASE-Diagramme (auch Anwendungs-Fall-Diagramme genannt).

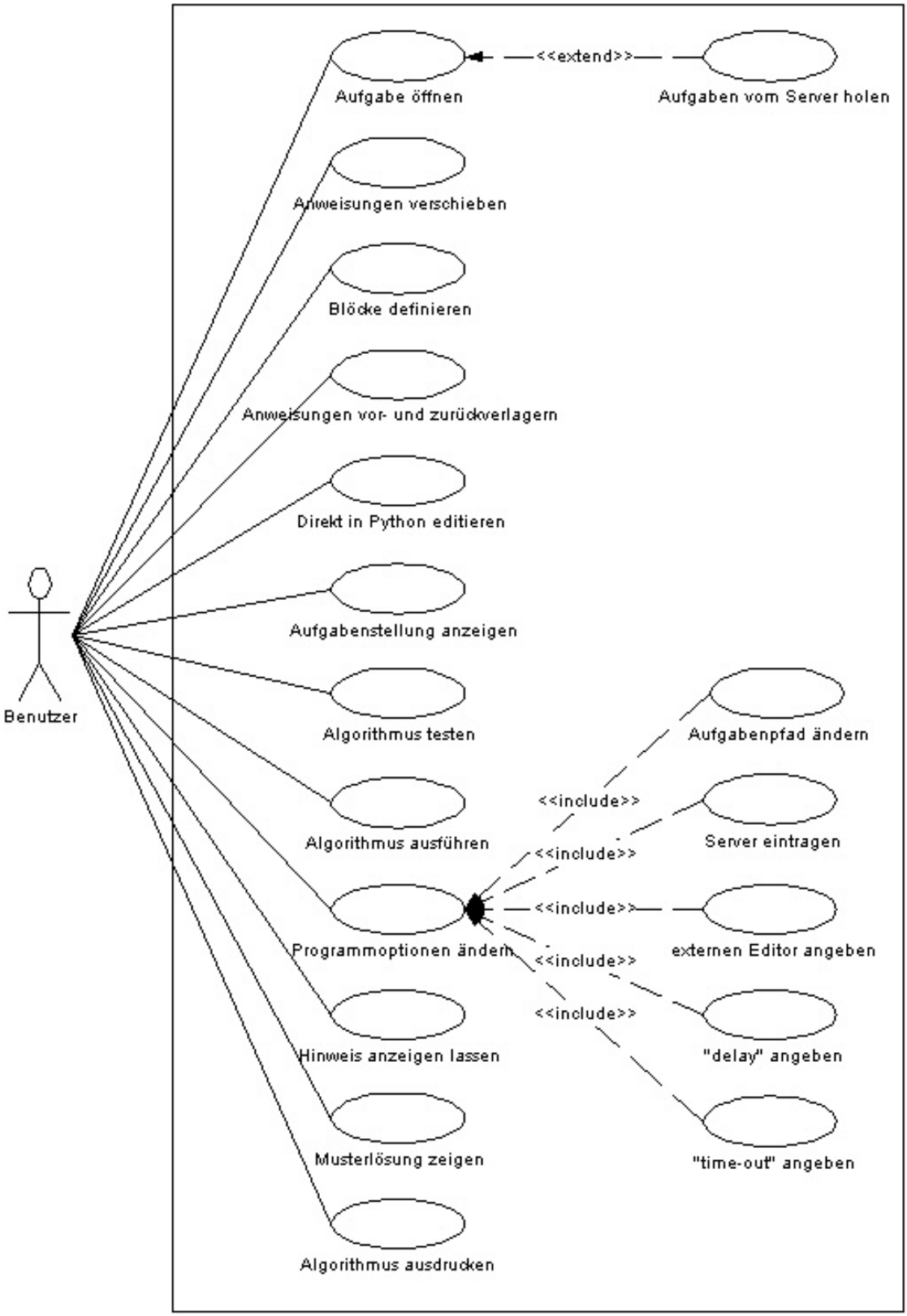
12.1.1 Übersicht aller Teilprogramme

Grobstruktur des ProgrammierTrainers. Es gibt zwei große Teilbereiche. Diese sind der ProgrammierTrainer selbst und der Example-Builder zum Erstellen von neuen Aufgabestellungen.



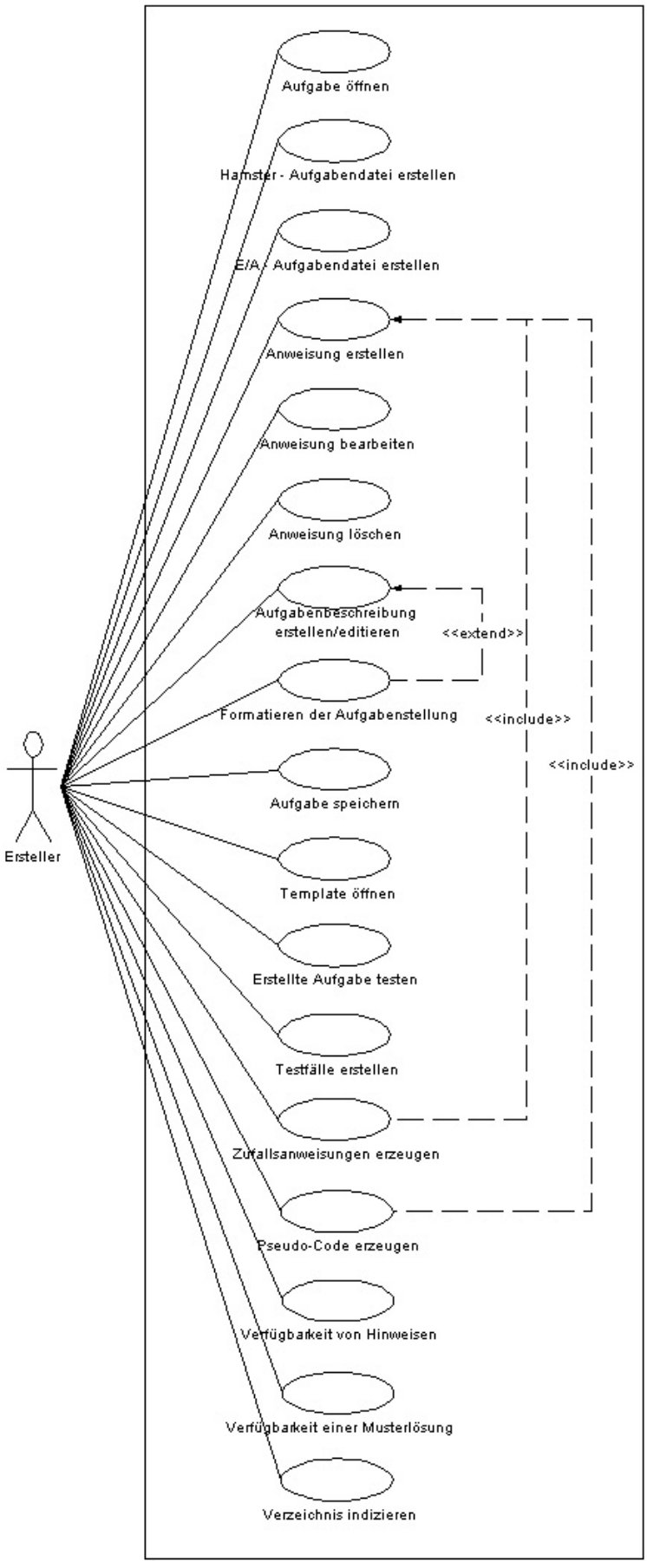
12.1.2 Hauptprogramm – ProgrammierTrainer

In der folgenden Abbildung sehen sie das USE-CASE-Diagramm des ProgrammierTrainers.



12.1.3 Example-Builder

In der folgenden Abbildung sehen sie das USE-CASE-Diagramm des Example-Builder.

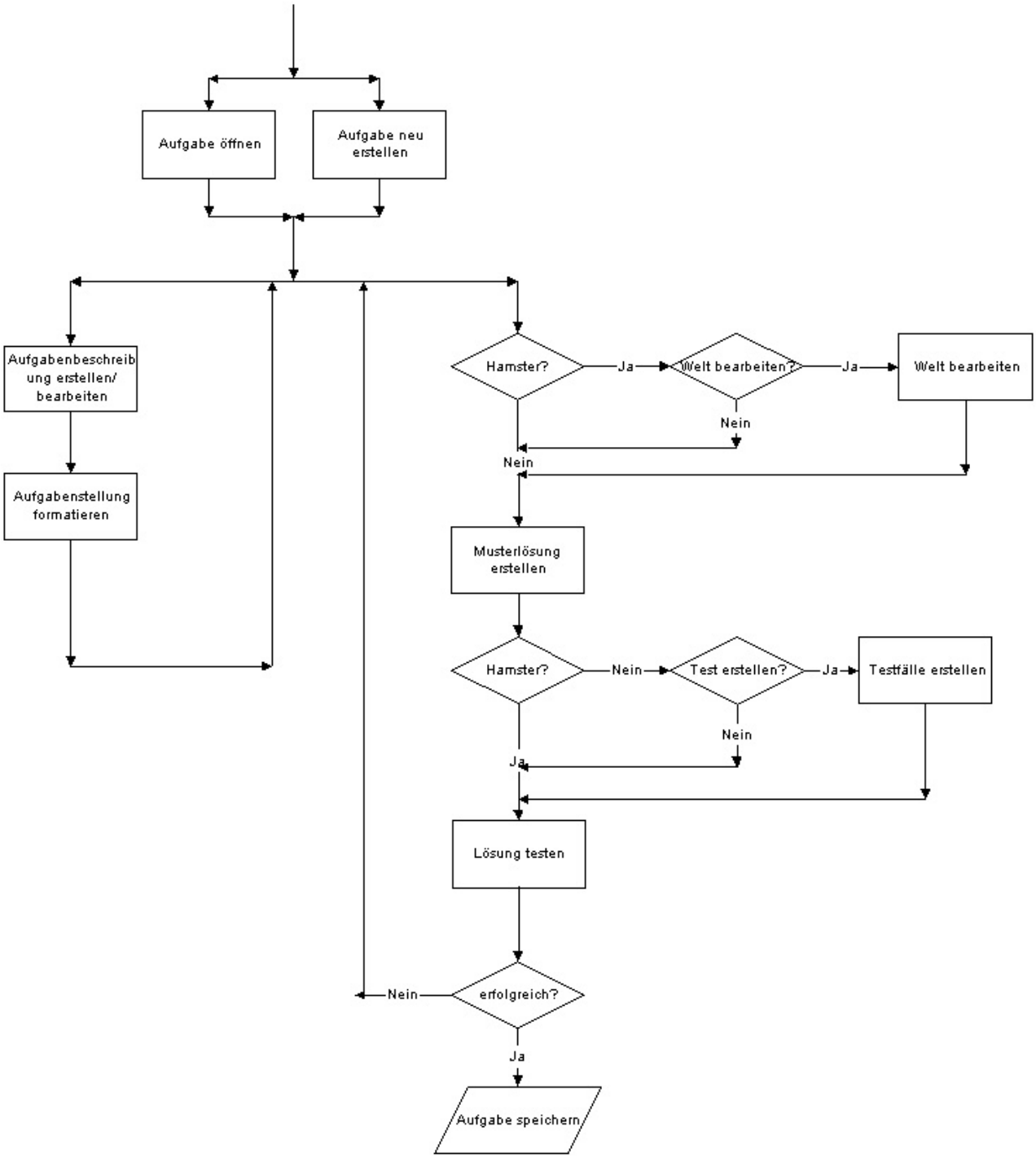


12.2 Ablaufdiagramme

Die folgenden Ablaufdiagramme sollen die Grundfunktionen aus der Sicht des Benutzers veranschaulichen.

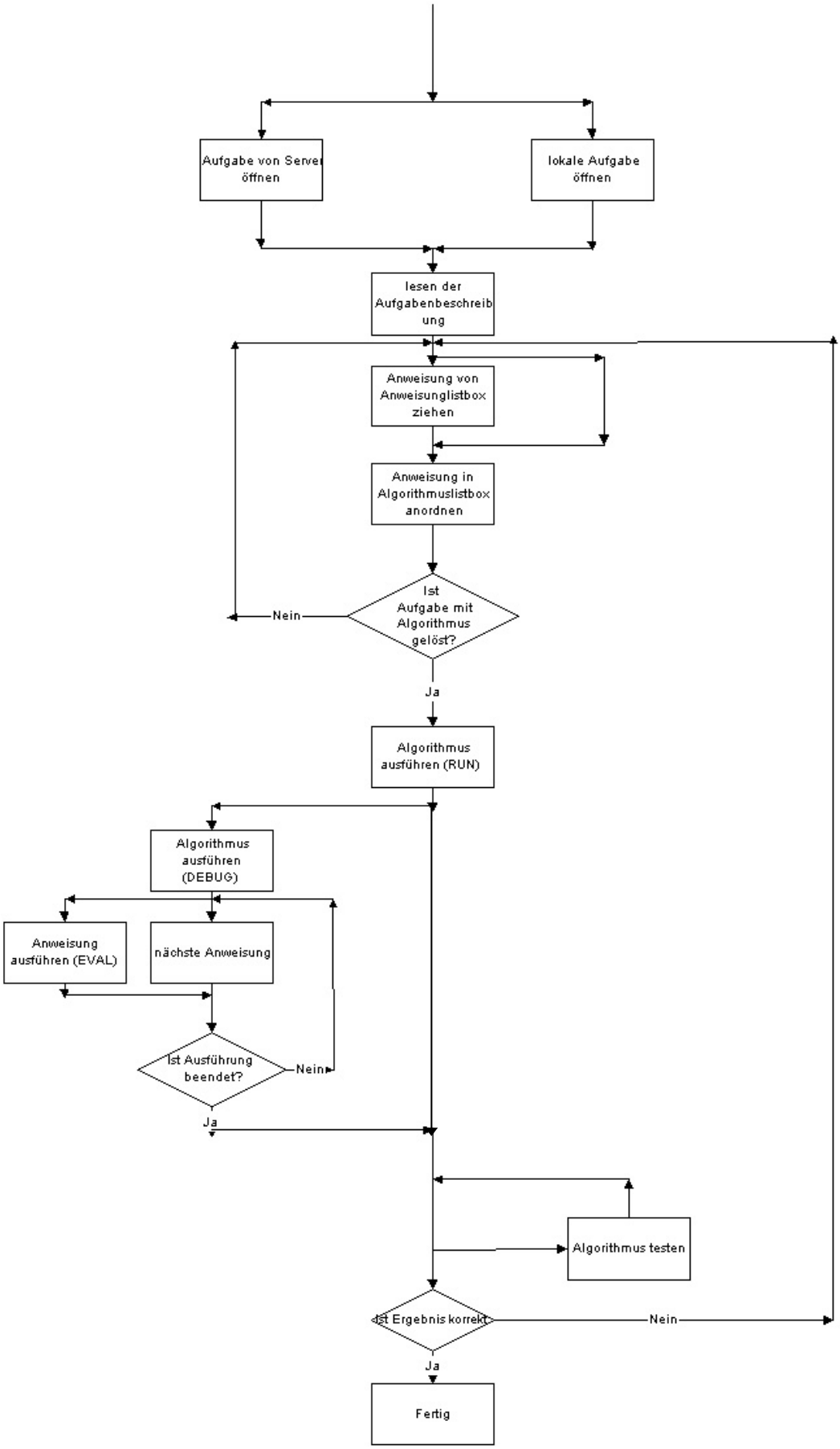
12.2.1 Erstellen einer Aufgabe

Dieser Punkt entspricht praktisch den Grundfunktionen des ExampleBuilders. Es soll veranschaulicht werden, wie der Benutzer beim Erstellen einer Aufgabe vorzugehen hat.



12.2.2 Lösen einer Aufgabenstellung

Dieser Punkt entspricht praktisch den Grundfunktionen des ProgrammierTrainers. Es soll veranschaulicht werden, wie der Benutzer beim Lösen einer Aufgabe vorzugehen hat.



12.3 Beschreibung der USE-CASE-Diagramme

In den folgenden Unterkapiteln werden die einzelnen USE-CASES näher beschrieben.

12.3.1 ProgrammierTrainer

Hier finden sie nähere Informationen zu den einzelnen Use-Cases des ProgrammierTrainers.

Aufgabe öffnen

Name:	Aufgabe öffnen
Zweck:	Zum Öffnen einer beliebigen Aufgabenstellung.
Beschreibung:	Der Benutzer öffnet eine von ihm gewünschte Aufgabenstellung. Diese kann entweder eine Hamster- oder E/A-Aufgabe sein. Außerdem hat der Benutzer noch die Wahl, ob er eine lokal gespeicherte Aufgabenstellung oder eine auf einem Server liegende Aufgabenstellung laden möchte. Hat der Benutzer eine Aufgabenstellung ausgewählt, so wird diese in den ProgrammierTrainer geladen. Ist bereits eine Aufgabenstellung geladen, so wird rückgefragt, ob diese gespeichert werden soll, bevor sie durch die neue Aufgabenstellung ersetzt wird.
Vorbedingungen:	Es gibt zwei Möglichkeiten: 1. Es ist bereits eine Aufgabenstellung geladen 2. Es wurde noch keine Aufgabenstellung geladen
Nachbedingungen:	Die vom Benutzer ausgewählte Aufgabenstellung wird in den ProgrammierTrainer geladen und es öffnet sich ein zweites Fenster mit der Aufgabenbeschreibung.
Alternative Abläufe:	Abbruch des Vorgangs durch den Benutzer: In diesem Fall werden im ProgrammierTrainer keine Änderungen vorgenommen.

Aufgaben vom Server holen

Name:	Aufgaben vom Server holen
Zweck:	Zum laden von zentral gespeicherten Aufgabenstellungen.
Beschreibung:	Der Benutzer hat die Möglichkeit Aufgabenstellungen von einem Server zu beziehen. Hierzu verbindet sich der ProgrammierTrainer mit einem Server und sucht nach einer Indexdatei. In dieser Indexdatei sind alle Aufgabenstellungen angeführt, welche auf diesem Server liegen. Diese Aufgabenstellungen hat der Benutzer dann zur Auswahl.
Vorbedingungen:	Der Benutzer ist gerade beim Laden einer Aufgabe.
Nachbedingungen:	Der Benutzer hat die Möglichkeit, die vom Server geholten Aufgaben in den ProgrammierTrainer zu laden.
Alternative Abläufe:	Abbruch des Vorganges durch den Benutzer.

Anweisungen verschieben

Name:	Anweisungen verschieben
Zweck:	Durch das Verschieben der Anweisungen von der Liste mit den möglichen Anweisungen (links), in die Algorithmus-Liste(rechts) hat der Benutzer die Möglichkeit einen Algorithmus zu formulieren.
Beschreibung:	Mittels Drag&Drop hat der Benutzer die Möglichkeit Anweisungen zu Verschieben. Mittels dieses Vorgangs werden vom Benutzer jene Anweisungen in die rechte Liste verschoben, welche er zum Erstellen eines Algorithmus benötigt. Weiters können die Anweisungen, welche in die Algorithmus-Liste verschoben wurden, wieder in die Liste mit den möglichen Anweisungen zurück verschoben werden.
Vorbedingungen:	Der Benutzer hat eine Aufgabenstellung in den ProgrammierTrainer geladen.
Nachbedingungen:	Es gibt zwei, den Aktionen entsprechende, Nachbedingungen: 1. Der Benutzer hat die, seines Erachtens, benötigte Anweisung in die rechte Liste verschoben. 2. Der Benutzer hat die, seines Erachtens, nicht benötigte Anweisung zurück in die linke Liste verschoben.
Alternative Abläufe:	Es gibt zwei Möglichkeiten: 1. Der Benutzer verschiebt eine Anweisung innerhalb der rechten Liste: Dieser Vorgang hat keine Auswirkungen. 2. Der Benutzer verschiebt eine Anweisung innerhalb der linken Liste: Dieser Vorgang hat ebenfalls keine Auswirkungen.

Blöcke definieren

Name:	Blöcke definieren
Zweck:	Der Benutzer hat die Möglichkeit Anweisungsblöcke zu definieren.
Beschreibung:	Mithilfe der Pfeiltasten im ProgrammierTrainer hat der Benutzer die Möglichkeit, Einrückungen innerhalb der Algorithmus-Liste vorzunehmen. Mithilfe von Einrückungen werden im ProgrammierTrainer Blöcke deklariert. Diese Blöcke sind zum Beispiel: Schleifen, IF-Anweisungen, Prozeduren, usw… Der Benutzer hat jedoch auch jederzeit die Möglichkeit, die von ihm vorgenommene Blockdefinition rückgängig zu machen.
Vorbedingungen:	Es befinden sich bereits eine oder mehrere Anweisungen in der Algorithmus-Liste.
Nachbedingungen:	Es wurde ein Block definiert.
Alternative Abläufe:	Eine vorgenommene Blockdefinition wurde rückgängig gemacht.

Anweisungen vor- und zurückverlagern

Name:	Anweisungen vor- und zurückverlagern
Zweck:	Dient zur Änderung der Reihenfolge der Anweisungen.
Beschreibung:	Der Benutzer ändert die Reihenfolge der in der Algorithmus-Liste stehenden Anweisungen. Dies kann auf verschiedene Arten geschehen: 1. Pfeil-Schaltflächen im ProgrammierTrainer-Interface 2. Tastenkürzel (Pfeiltasten)
Vorbedingungen:	Es befinden sich mindestens zwei Anweisungen in der Algorithmus-Liste.
Nachbedingungen:	Die Reihenfolge der Anweisungen wurde verändert.
Alternative Abläufe:	Es gibt zwei alternative Abläufe: 1. Die markierte Anweisung kann nicht nach oben verschoben werden, da sie die erste Anweisung ist. 2. Die markierte Anweisung kann nicht nach unten verschoben werden, da sie die letzte Anweisung ist.

Direkt in Python editieren

Name:	Direkt in Python editieren
Zweck:	Mithilfe eines Editors, soll es dem Benutzer möglich sein, seine Algorithmen direkt in Python zu formulieren.
Beschreibung:	Durch den Aufruf des Python-Editors soll es dem Benutzer möglich sein, seine Algorithmen in Python zu formulieren. Hierzu wird entweder der in den ProgrammierTrainer eingebaute Python-Editor oder der vom Benutzer angegebene externe Editor verwendet. Hat der Benutzer seine Anweisungen erstellt, so ist es ihm möglich, diese in den ProgrammierTrainer zu übernehmen, um seinen Algorithmus zu testen.
Vorbedingungen:	Eine Aufgabe wurde geöffnet.
Nachbedingungen:	Der Benutzer hat den von ihm erstellten Algorithmus in den ProgrammierTrainer geladen.
Alternative Abläufe:	Der Benutzer hat den von ihm erstellten Algorithmus nicht in den ProgrammierTrainer geladen: Dies hat ebenfalls keine Auswirkungen auf den ProgrammierTrainer.

Aufgabenstellung anzeigen

Name:	Aufgabenstellung anzeigen
Zweck:	Dem Benutzer wird die Definition der von ihm geöffneten Aufgabenstellung angezeigt.
Beschreibung:	Es öffnet sich ein eigenes Fenster mit der Beschreibung der Aufgabenstellung. Bei Hamster-Aufgaben beinhaltet dieser Aufgabendialog zusätzlich noch ein Hamster-Panel, in welchem die geforderte Lösung zur Aufgabenstellung mithilfe der Musterlösung vorgeführt wird.
Vorbedingungen:	Eine Aufgabe wurde geladen.
Nachbedingungen:	Dem Benutzer wird die Definition der Aufgabenstellung gezeigt.
Alternative Abläufe:	-

Algorithmus testen

Name:	Algorithmus testen
Zweck:	Der Benutzer hat die Möglichkeit, seinen Algorithmus anhand von vordefinierten Testfällen zu testen.
Beschreibung:	Hierzu werden die vom Ersteller der Aufgabenstellung definierten Testfälle auf den Benutzer-Algorithmus angewandt. Der Test kann entweder erfolgreich sein, oder fehlschlagen.
Vorbedingungen:	Der Benutzer hat einen Algorithmus erstellt.
Nachbedingungen:	Der Algorithmus hat alle Testfälle bestanden.
Alternative Abläufe:	Der Algorithmus hat mindestens einen Testfall nicht bestanden.

Algorithmus ausführen

Name:	Algorithmus ausführen
Zweck:	Der Benutzer hat die Möglichkeit, die von ihm erstellten Anweisungen auszuführen.
Beschreibung:	Der vom Benutzer erstellte Algorithmus wird vom ProgrammierTrainer interpretiert und ausgeführt. Der Benutzer hat die Möglichkeit seinen Algorithmus einerseits im RUN- und andererseits im DEBUG-Modus auszuführen. RUN: Die Anweisungen werden nacheinander mit einem vom Benutzer definierten „delay“ ausgeführt. DEBUG: Der Benutzer hat die Möglichkeit, die Anweisungen seines Algorithmus einzeln auszuführen. Die durch den Algorithmus erzeugten Ausgaben werden in einer eigenen Liste angezeigt.
Vorbedingungen:	Es befindet sich mindestens eine Anweisung in der Algorithmus-Liste.
Nachbedingungen:	Der Benutzer sieht die Ergebnisse seines Algorithmus.
Alternative Abläufe:	Es gibt zwei alternative Abläufe: 1. Der Benutzer hat einen syntaktischen Fehler in seinem Algorithmus. In diesem Fall ist es dem Benutzer nicht möglich den Algorithmus auszuführen. 2. Der Benutzer erkennt einen logischen Fehler in seinem Algorithmus. In diesem Fall liefert der vom Benutzer erstellte Algorithmus das falsche Ergebnis.

Optionen ändern

Name:	Optionen ändern
Zweck:	Der Benutzer hat die Möglichkeit, den ProgrammierTrainer seinen Ansprüchen gemäß zu konfigurieren.
Beschreibung:	Der Benutzer kann folgende Einstellungen ändern: 1. Aufgabenpfad 2. Server-URL 3. externer Editor 4. „delay“: Ist die Verzögerung, mit welcher sich der Hamster durch die Hamster-Welt bewegt. 5. „time-out“: Ist die Zeitspanne, nach welcher der Algorithmus beim Ausführen abgebrochen wird. Eingabeaufforderungen unterbrechen jedoch diese Zeitspanne. Die Einstellungen, welche geändert werden können, werden in den folgenden Use-Cases beschreiben.
Vorbedingungen:	ProgrammierTrainer gestartet
Nachbedingungen:	Einstellungen wurden geändert.
Alternative Abläufe:	Der Benutzer hat den Vorgang abgebrochen.

Aufgabenpfad ändern

Name:	Aufgabenpfad ändern
Zweck:	Der Aufgabenpfad wird vom Benutzer geändert.
Beschreibung:	Mit dem Aufgabenpfad wird angegeben, wo der ProgrammierTrainer nach Aufgabenstellungen suchen soll.
Vorbedingungen:	Der Optionen-Dialog des ProgrammierTrainers wurde geöffnet.
Nachbedingungen:	Der Aufgabenpfad wurde geändert.
Alternative Abläufe:	Der Aufgabenpfad wurde nicht geändert.

Server eintragen

Name:	Server eintragen
Zweck:	Zum Ändern der URL des Aufgaben-Servers.
Beschreibung:	Mit der Server-Adresse wird der Server definiert, auf welchem der ProgrammierTrainer nach Aufgabenstellungen suchen soll.
Vorbedingungen:	Der Optionen-Dialog des ProgrammierTrainers wurde geöffnet.
Nachbedingungen:	Die Server-Adresse wurde eingetragen/geändert.
Alternative Abläufe:	Die Server-Adresse wurde nicht eingetragen/geändert.

Externen Editor angeben

Name:	Externen Editor angeben
Zweck:	Hier wird ein externer Editor angegeben, welcher zum Editieren verwendet werden soll.
Beschreibung:	Der hier angegebene Editor wird gestartet, wenn der Benutzer seinen Algorithmus direkt in Python formulieren möchte. Wird hier kein externer Editor angegeben, so wird der in den ProgrammierTrainer integrierte Python-Editor verwendet.
Vorbedingungen:	Der Optionen-Dialog des ProgrammierTrainers wurde geöffnet.
Nachbedingungen:	Editor-Dateipfad wurde eingetragen/geändert.
Alternative Abläufe:	Editor-Dateipfad wurde nicht eingetragen/geändert.

„`delay`“ angeben

Name:	„`delay`“ angeben
Zweck:	Der Benutzer kann hier die Verzögerung angeben, mit welcher sich der Hamster durch die Hamster-Welt bewegt.
Beschreibung:	Das hier angegebene „`delay`“ wird nur bei Hamster-Aufgaben verwendet.
Vorbedingungen:	Der Optionen-Dialog des ProgrammierTrainers wurde geöffnet.
Nachbedingungen:	Die Angaben für die Verzögerung wurden geändert.
Alternative Abläufe:	Die Angaben für die Verzögerung wurden nicht geändert.

„`time-out`“ angeben

Name:	„`time-out`“ angeben
Zweck:	Der Benutzer kann hier die Verzögerung angeben, mit welcher sich der Hamster durch die Hamster-Welt bewegt.
Beschreibung:	Das hier angegebene „`delay`“ wird nur bei Hamster-Aufgaben verwendet.
Vorbedingungen:	Der Optionen-Dialog des ProgrammierTrainers wurde geöffnet.
Nachbedingungen:	Die Angaben für das „`time-out`“ wurden geändert.
Alternative Abläufe:	Die Angaben für das „`time-out`“ wurden nicht geändert.

Hinweis anzeigen lassen

Name:	Hinweis anzeigen lassen
Zweck:	Dem Benutzer wird ein Tipp gegeben.
Beschreibung:	Falls der Ersteller der Aufgabenstellung dem Benutzer das Recht eingeräumt hat, einen Hinweis anzufordern, so hat der Benutzer die Möglichkeit sich die nächste in Frage kommende Anweisung markieren zu lassen. Weiters werden auch die Anweisungen markiert, bei der die Einrückung nicht korrekt ist.
Vorbedingungen:	Eine Aufgabe wurde geöffnet.
Nachbedingungen:	Dem Benutzer wird ein Hinweis gegeben.
Alternative Abläufe:	Es gibt zwei alternative Abläufe: 1. Der Algorithmus ist korrekt: In diesem Fall wird keine Anweisung markiert. 2. Der Menüpunkt „`Hinweis`“ ist für die aktuelle Aufgabe nicht aktiv.

Musterlösung zeigen

Name:	Musterlösung zeigen
Zweck:	Dem Benutzer wird die Musterlösung angezeigt.
Beschreibung:	Falls der Ersteller der Aufgabenstellung dem Benutzer das Recht eingeräumt hat, die Musterlösung anzufordern, so hat der Benutzer die Möglichkeit sich diese Anzeigen zu lassen.
Vorbedingungen:	Eine Aufgabe wurde geöffnet.
Nachbedingungen:	Die Musterlösung wird angezeigt.
Alternative Abläufe:	Der Menüpunkt „`Musterlösung`“ ist für die aktuelle Aufgabe nicht aktiv.

Algorithmus ausdrucken

Name:	Algorithmus ausdrucken
Zweck:	Dem Benutzer soll es möglich sein, den von ihm erstellten Algorithmus auszudrucken.
Beschreibung:	Der vom Benutzer erstellte Algorithmus und weitere Informationen werden in Druckformat gebracht und an den ausgewählten Drucker geschickt.
Vorbedingungen:	Eine Aufgabe wurde geöffnet.
Nachbedingungen:	Der vom Benutzer erstellte Algorithmus wird dem gewählten Format (siehe 7.2.1) entsprechend ausgedruckt.
Alternative Abläufe:	Es gibt zwei alternative Abläufe: 1. Fehler beim Ausdrucken 2. Abbruch durch den Benutzer In beiden Fällen gibt es keine Ausgabe auf dem Drucker.

12.3.2 Example-Builder

Hier finden sie nähere Informationen zu den einzelnen Use-Cases des ExampleBuilders.

Aufgabe öffnen

Name:	Aufgabe öffnen
Zweck:	Der Ersteller möchte Änderungen an einer bereits erstellten Aufgabe vornehmen.
Beschreibung:	Der Ersteller öffnet eine Aufgabendatei. Diese Aufgabendatei wird dann in den Example-Bilder geladen. Ist diese Aufgabe geladen, so kann der Ersteller jederzeit Änderungen vornehmen.
Vorbedingungen:	Example-Builder wurde gestartet.
Nachbedingungen:	Eine Aufgabendatei wurde in den Example-Builder geladen
Alternative Abläufe:	Abbruch durch den Ersteller: Es wird keine Aufgabendatei in den Example-Builder geladen.

Hamster Aufgabendatei erstellen

Name:	Hamster Aufgabendatei erstellen
Zweck:	Der Ersteller möchte eine neue Hamster-Aufgabenstellung entwerfen.
Beschreibung:	Der Ersteller legt eine neue Hamster-Aufgabe an.
Vorbedingungen:	Der Example-Builder wurde gestartet.
Nachbedingungen:	Der Ersteller kann nun eine Hamster-Aufgabe entwerfen.
Alternative Abläufe:	Im Example-Builder befindet sich bereits eine geladene Aufgabe. Deshalb gibt es zwei Alternativen: 1. Soll die geladene Aufgabe, bevor sie geschlossen wird, gespeichert werden? 2. Soll die geladene Aufgabe, ohne sie zu speichern, geschlossen werden werden?

E/A-Aufgabendatei erstellen

Name:	E/A-Aufgabendatei erstellen
Zweck:	Der Ersteller möchte eine neue E/A-Aufgabenstellung entwerfen.
Beschreibung:	Der Ersteller legt eine neue E/A-Aufgabe an.
Vorbedingungen:	Der Example-Builder wurde gestartet.
Nachbedingungen:	Der Ersteller kann nun eine E/A-Aufgabe entwerfen.
Alternative Abläufe:	Im Example-Builder befindet sich bereits eine geladene Aufgabe. Deshalb gibt es zwei Alternativen: 1. Soll die geladene Aufgabe, bevor sie geschlossen wird, gespeichert werden? 2. Soll die geladene Aufgabe, ohne sie zu speichern, geschlossen werden werden?

Anweisung erstellen

Name:	Anweisung erstellen
Zweck:	Der Ersteller formuliert die Anweisungen der Musterlösung.
Beschreibung:	Im Example-Builder muss es dem Ersteller möglich sein, eine Musterlösung zu erstellen. Zu diesem Zweck kann der Ersteller einzelne Anweisungen formulieren, diese zu Blöcken zusammenfassen und sortieren. Die einzelnen Anweisungen werden vom Ersteller in Python implementiert. Optional kann er auch noch einen Pseudo-Code zuordnen, wenn dies nicht automatisch geschehen soll.
Vorbedingungen:	Eine Aufgabendatei wurde geöffnet bzw. eine neue erstellt.
Nachbedingungen:	Mindestens eine Anweisung wurde zur Musterlösung hinzugefügt.
Alternative Abläufe:	Es wurden keine Anweisungen in die Musterlösung hinzugefügt.

Anweisung bearbeiten

Name:	Anweisung bearbeiten
Zweck:	Zum Abändern einer bereits erstellten Anweisung.
Beschreibung:	Der Ersteller möchte eine bereits bestehende Anweisung abändern. Bei diesem Arbeitsschritt kann er sowohl den Python-Code als auch den zugewiesenen Pseudo-Code ändern. Weiters kann auch die Einrückung dieser Anweisung geändert werden.
Vorbedingungen:	Es ist mindestens eine Anweisung in der Musterlösung vorhanden.
Nachbedingungen:	Es wurden Änderungen in der Anweisung vorgenommen.
Alternative Abläufe:	Es wurden keine Änderungen in der Anweisung vorgenommen.

Anweisung löschen

Name:	Anweisung löschen
Zweck:	Zum Löschen einer in der Anweisungsliste der Musterlösung stehenden Anweisung.
Beschreibung:	Der Ersteller möchte in diesem Arbeitsschritt eine bereits bestehende Anweisung aus der Musterlösung herauslöschen.
Vorbedingungen:	Es ist mindestens eine Anweisung in der Musterlösung vorhanden.
Nachbedingungen:	Eine Anweisung wurde aus der Musterlösung herausgelöscht.
Alternative Abläufe:	Der Benutzer entscheidet sich anders: Es werden keine Änderungen in der Musterlösung vorgenommen.

Aufgabenbeschreibung erstellen/editieren

Name:	Aufgabenbeschreibung editieren
Zweck:	Um den Benutzer eine Problembeschreibung für die zu lösende Aufgabe bieten zu können.
Beschreibung:	Der Ersteller schreibt im Example-Builder eine Aufgabenbeschreibung, welche die für den Benutzer zu lösende Problemstellung definiert.
Vorbedingungen:	Eine Aufgabendatei wurde geöffnet bzw. eine Neue angelegt.
Nachbedingungen:	Der Ersteller hat eine Aufgabenbeschreibung für die zu lösende Problemstellung in der Aufgabendatei abgespeichert.
Alternative Abläufe:	Der Benutzer hat keine Änderungen an der Aufgabenbeschreibung vorgenommen.

Formatieren der Aufgabenbeschreibung

Name:	Formatieren der Aufgabenbeschreibung
Zweck:	Der Ersteller kann Formatierungen innerhalb der textuellen Aufgabenbeschreibung vornehmen.
Beschreibung:	Der Ersteller nimmt Formatierungen innerhalb der Aufgabenbeschreibung vor. Es gibt fünf verschiedene Formatierungsarten: 1. Fett 2. Kursiv 3. Unterstrichen 4. Aufzählung 5. Link
Vorbedingungen:	Aufgabenbeschreibung wird erstellt/editiert.
Nachbedingungen:	Es wurden Formatierungen in die Aufgabenbeschreibung eingefügt.
Alternative Abläufe:	Der Ersteller nimmt keine Formatierungen vor.

Aufgabe speichern

Name:	Aufgabe speichern
Zweck:	Der Benutzer möchte die Änderungen an einer geöffneten Aufgabenstellung speichern.
Beschreibung:	Der Benutzer speichert die Aufgabenstellung. Weiters kann diese Aufgabenstellung auch verschlüsselt abgespeichert werden.
Vorbedingungen:	Es gibt zwei Möglichkeiten: 1. Es wurde eine Aufgabenstellung geöffnet. 2. Es wurde eine neue Aufgabenstellung erstellt.
Nachbedingungen:	Aufgrund der Vorbedingungen gibt es auch zwei verschiedene Möglichkeiten bei den Nachbedingungen: 1. Die Änderungen wurden in die Aufgabendatei übernommen. 2. Die Aufgabendatei wurde angelegt.
Alternative Abläufe:	Abbruch durch den Benutzer.

Template öffnen

Name:	Template öffnen
Zweck:	Der Ersteller erspart sich dadurch einen großen Zeitaufwand beim Schreiben von Standardanweisungen.
Beschreibung:	Es wird eine Vorlage für eine Aufgabenstellung geöffnet. Diese Vorlage beinhaltet Standardanweisungen, welche für diese Art von Aufgabenstellung notwendig sind.
Vorbedingungen:	Der Example-Builder wurde gestartet.
Nachbedingungen:	Eine Vorlage für die zu erstellende Aufgabenstellung wurde in den Example-Builder geladen.
Alternative Abläufe:	Im Example-Builder befindet sich bereits eine geladene Aufgabenstellung. Deshalb gibt es zwei Alternativen: 1. Soll die geladene Aufgabe, bevor sie geschlossen wird, gespeichert werden? 2. Soll die geladene Aufgabe, ohne sie zu speichern, geschlossen werden?

Erstellte Aufgabe testen

Name:	Erstellte Aufgabe testen
Zweck:	Zum Testen der vom Ersteller programmierten Musterlösung.
Beschreibung:	Die Musterlösung wird in einem eigenen Fenster geöffnet und ausgeführt. Sind Benutzereingaben erforderlich, so öffnen sich Pop-up-Fenster, welche vom Ersteller die gewünschten Eingaben fordern.
Vorbedingungen:	Der Ersteller hat eine Musterlösung entwickelt.
Nachbedingungen:	Die Musterlösung wurde getestet.
Alternative Abläufe:	Es gibt zwei alternative Abläufe: 1. Der Ersteller hat einen syntaktischen Fehler in seiner Musterlösung. In diesem Fall ist es ihm nicht möglich den Algorithmus auszuführen. 2. Der Ersteller erkennt einen logischen Fehler in seinem Algorithmus. In diesem Fall liefert die von ihm erstellte Musterlösung das falsche Ergebnis.

Testfälle erstellen

Name:	Testfälle erstellen
Zweck:	Damit der Ersteller Testfälle für seine Aufgabe erzeugen kann.
Beschreibung:	Für diesen Punkt öffnet sich ein eigenes Fenster, in welchem der Ersteller, Aus- und Eingabe-Werte für sein Programm definieren kann. Zusätzlich kann er Anfangsinitialisierungen und End-Werte für die Variablen definieren. Anhand von diesen Daten kann eine mögliche Korrektheit des Programms garantiert werden.
Vorbedingungen:	Eine Aufgabe ist geöffnet.
Nachbedingungen:	Der Ersteller hat Testfälle für seine Aufgabenstellung festgelegt.
Alternative Abläufe:	Abbruch durch den Ersteller.

Zufallsanweisungen erzeugen

Name:	Zufallsanweisungen erzeugen
Zweck:	Zum automatischen Erstellen von Zufallsanweisungen.
Beschreibung:	Zufallsanweisungen sind zur Verwirrung der Benutzer nötig. Die Zufallsanweisungen werden aus der vom Benutzer erstellten Musterlösung generiert.
Vorbedingungen:	Es steht mindestens eine Anweisung in der Anweisungs-Liste.
Nachbedingungen:	Zufallsanweisungen wurden in die Aufgabendatei hinzugefügt.
Alternative Abläufe:	Aus dem vom Benutzer erzeugten Algorithmus konnten keine Zufallsanweisungen erzeugt werden.

Pseudo-Code erzeugen

Name:	Pseudo-Code erzeugen
Zweck:	Zum automatischen Generieren von Pseudo-Code-Zuweisungen.
Beschreibung:	Zu den vom Ersteller erzeugten Python-Code-Anweisungen werden automatisch Pseudo-Code-Anweisungen generiert.
Vorbedingungen:	Um diese Aktion erfolgreich durchführen zu können muss unbedingt eine Zuordnungsdatei vorhanden sein. Außerdem muss mindestens eine Anweisung in der Anweisungs-Liste stehen.
Nachbedingungen:	Zu jeder Python-Code-Anweisung wurde eine Pseudo-Code-Anweisung erzeugt.
Alternative Abläufe:	In der Zuordnungsdatei wurde zu einer Python-Code-Anweisung keine Zuordnung gefunden. In diesem fall wird keine Pseudo-Code-Anweisung erzeugt.

Verfügbarkeit von Hinweisen festlegen

Name:	Verfügbarkeit von Hinweisen festlegen.
Zweck:	Der Ersteller kann kontrollieren, ob der Benutzer des ProgrammierTrainers Hinweise erhalten darf.
Beschreibung:	Der Ersteller wählt, ob für die von ihm geöffnete Aufgabe Hinweise zulässig sein sollen oder nicht.
Vorbedingungen:	Im Example-Builder befindet sich eine geöffnete Aufgabendatei.
Nachbedingungen:	Der Ersteller legt fest, dass Hinweise verwendet werden dürfen.
Alternative Abläufe:	Der Ersteller legt fest, dass keine Hinweise verwendet werden dürfen.

Verfügbarkeit der Musterlösung festlegen

Name:	Verfügbarkeit der Musterlösung festlegen
Zweck:	Der Ersteller kann kontrollieren, ob der Benutzer des ProgrammierTrainers die Musterlösung erhalten darf.
Beschreibung:	Er wählt, ob für die von ihm geöffnete Aufgabe, dem Benutzer des ProgrammierTrainers die Musterlösung angezeigt werden darf oder nicht.
Vorbedingungen:	Im Example-Builder befindet sich eine geöffnete Aufgabendatei.
Nachbedingungen:	Der Ersteller legt fest, dass der Benutzer des ProgrammierTrainers die Musterlösung verwenden darf.
Alternative Abläufe:	Der Ersteller legt fest, dass der Benutzer des ProgrammierTrainers die Musterlösung nicht verwenden darf.

Verzeichnis indizieren

Name:	Verzeichnis indizieren
Zweck:	Der Benutzer des ExampleBuilders kann ein Verzeichnis indizieren.
Beschreibung:	Der Ersteller kann ein Verzeichnis angeben, welches indiziert werden soll. (Der genaue Vorgang des Indizierens wird unter Punkt 8.2.3 beschrieben)
Vorbedingungen:	Der Example-Builder wurde gestartet.
Nachbedingungen:	Das Verzeichnis wurde indiziert.
Alternative Abläufe:	Abbruch durch den Benutzer.

13 Lösungsansatz (MR)

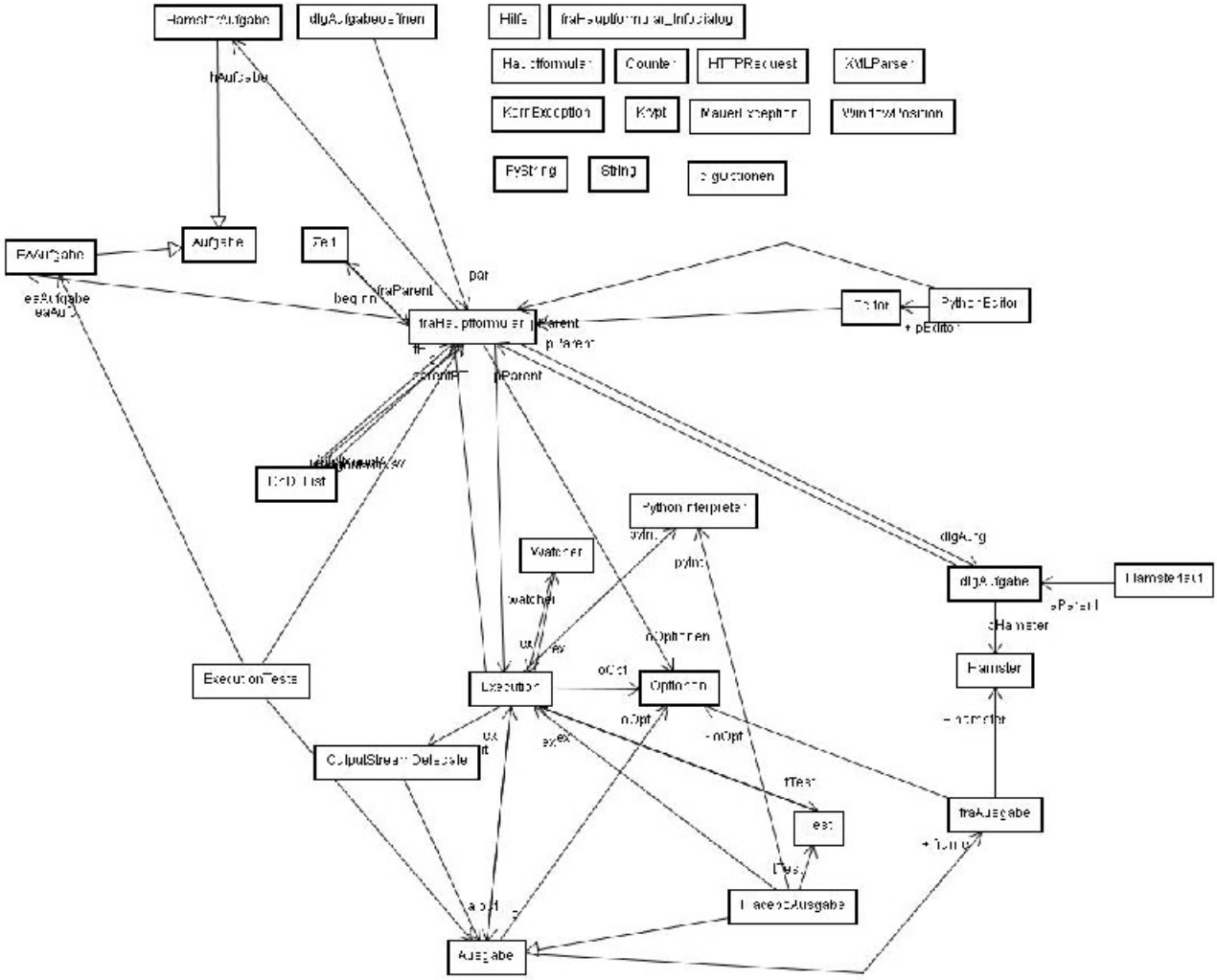
Im folgenden Kapitel werde ich einen groben Überblick über unseren Lösungsansatz für die zu realisierenden Systeme liefern.

13.1 Klassendiagramm

Hier wird die Klassenaufteilung der einzelnen Systeme grafisch beschrieben.

13.1.1 ProgrammierTrainer

Das Klassendiagramm des ProgrammierTrainers.

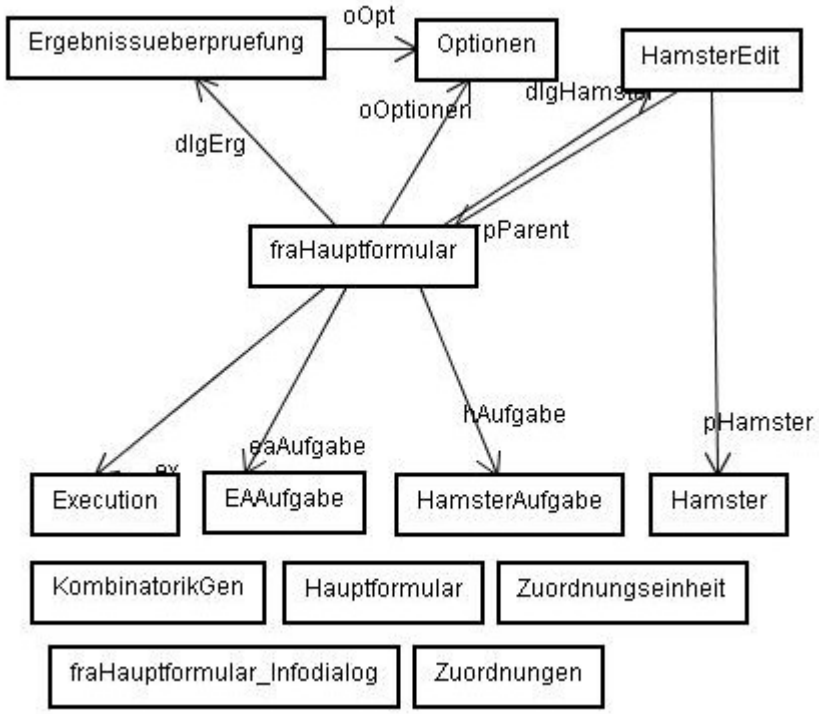


13.1.2 Attribute und Methoden der Klassen im ProgrammierTrainer

Die im obigen Diagramm gezeigten Klassen werden in der JavaDoc, welche sich auf der Installations-CD befindet, beschrieben. Dort sind alle Informationen zu den Methoden und Attributen der einzelnen Klassen ersichtlich.

13.1.3 ExampleBuilder

Das Klassendiagramm des ExampleBuilders.



13.1.4 Attribute und Methoden der Klassen im ExampleBuilder

Die im obigen Diagramm gezeigten Klassen werden in der JavaDoc, welche sich auf der Installations-CD befindet, beschrieben. Dort sind alle Informationen zu den Methoden und Attributen der einzelnen Klassen ersichtlich.

14 Ausblick (ED)

Bei der Erweiterung steht in Planung, eine Testumgebung zu implementieren. Dieser Punkt wurde bereits im Pflichtenheft angesprochen. Bei der Testumgebung soll es möglich sein, die auf Clients laufende ProgrammierTrainer-Applikationen von einem zentralen Server aus zu verwalten. D.h. am Server sollen Statistiken erhoben werden können, Nachrichten an bestimmte Clients geschickt werden und weitere Funktionen die als sinnvoll erachtet werden, durchgeführt werden können. Schließlich sollen die an den Clients erstellten Algorithmen abgegeben werden können. D.h. Sie sollen zum Server geschickt werden.

Ein weiterer Punkt bei der Erweiterung ist die Abspeicherung von Bildern in einer Aufgabenbeschreibung, was zurzeit nicht möglich ist.

15 Anhang A) komplette E/A-Aufgabe (MR)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<aufgabe name="Euro-Splitter">
   <Beschreibung><html>
  <head>

  </head>
  <body>
    <p>
      <b>Wieviel Scheine und M&#252;nzen sind das? </b>
    </p>
    <p>
      Es soll ein Programm entworfen werden, welches einen vom Benutzer
      eingegeben Betrag in die einzelen Scheine und M&#252;nzen aufteilt.
    </p>
    <p>
      Wider gr&#246;sseren Aufwandes wegen nehmen wir an, dass es 100, 50, 20 und
      10 Scheine und 2, 1, 0.5 und 0.1 Euro M&#252;nzen gibt.
    </p>
  </body>
</html>
</Beschreibung>
   <Programm>
def eurosplit(wert):#Funktion eurosplit(Parameter: wert)
   global eurobetrag#importiere globale Variable eurobetrag
   iAnz = 0#setze Variable iAnz auf 0
   while eurobetrag >= wert:#solange eurobetrag >= wert
      eurobetrag = round(eurobetrag - wert, 1)#vermindere eurobetrag um wert
      iAnz = iAnz + 1#erhöhe iAnz um 1
   return iAnz#eurosplit liefert iAnz zurück
eurobetrag = float(read("Der Euro-Betrag: "))#lese wert von Benutzer in eurobetrag ein
print str(eurobetrag) + " aufgesplittet"#gib eurobetrag und " aufgesplittet"
print "=================="#gib eine linie aus
print "100er: " + str(eurosplit(100))#gib "100er" und das ergebnis von eurosplit mit wert 100 aus
print "50er: " + str(eurosplit(50))#gib "50er" und das ergebnis von eurosplit mit wert 50 aus
print "20er: " + str(eurosplit(20))#gib "20er" und das ergebnis von eurosplit mit wert 20 aus
print "10er: " + str(eurosplit(10))#gib "10er" und das ergebnis von eurosplit mit wert 10 aus
print "2er: " + str(eurosplit(2))#gib "2er" und das ergebnis von eurosplit mit wert 2 aus
print "1er: " + str(eurosplit(1))#gib "1er" und das ergebnis von eurosplit mit wert 1 aus
print "50erl: " + str(eurosplit(0.5))#gib "50erl" und das ergebnis von eurosplit mit wert 0.5 aus
print "20erl: " + str(eurosplit(0.2))#gib "20erl" und das ergebnis von eurosplit mit wert 0.2 aus
print "10erl: " + str(eurosplit(0.1))#gib "10erl" und das ergebnis von eurosplit mit wert 0.1 aus
   </Programm>
   <Optionen>
      <Hinweis>Nein</Hinweis>
      <Musterloesung>Nein</Musterloesung>
      <Zeit>Nein</Zeit>
   </Optionen>
   <Tests>
      <Test id='1'>
         <Variablen id='1'>
           <Variable id='1' name='eurobetrag' end='0.0'></Variable>
         </Variablen>
         <Ausgaben id='1'>
           <Ausgabe id='1'>100.0 aufgesplittet</Ausgabe>
           <Ausgabe id='1'>*</Ausgabe>
           <Ausgabe id='1'>100er: 1</Ausgabe>
           <Ausgabe id='1'>*</Ausgabe>
           <Ausgabe id='1'>*</Ausgabe>
           <Ausgabe id='1'>*</Ausgabe>
           <Ausgabe id='1'>*</Ausgabe>
           <Ausgabe id='1'>*</Ausgabe>
           <Ausgabe id='1'>*</Ausgabe>
           <Ausgabe id='1'>*</Ausgabe>
           <Ausgabe id='1'>*</Ausgabe>
         </Ausgaben>
         <Eingaben id='1'>
           <Eingabe id='1'>100</Eingabe>
         </Eingaben>
      </Test>
      <Test id='2'>
         <Variablen id='2'>
           <Variable id='2' name='eurobetrag' end='0.02'></Variable>
         </Variablen>
```

```
      <Ausgaben id='2'>
        <Ausgabe id='2'>200.0 aufgesplittet</Ausgabe>
        <Ausgabe id='2'>*</Ausgabe>
        <Ausgabe id='2'>100er: 2</Ausgabe>
        <Ausgabe id='2'>*</Ausgabe>
        <Ausgabe id='2'>*</Ausgabe>
        <Ausgabe id='2'>*</Ausgabe>
        <Ausgabe id='2'>*</Ausgabe>
        <Ausgabe id='2'>*</Ausgabe>
        <Ausgabe id='2'>*</Ausgabe>
        <Ausgabe id='2'>*</Ausgabe>
        <Ausgabe id='2'>*</Ausgabe>
      </Ausgaben>
      <Eingaben id='2'>
        <Eingabe id='2'>200</Eingabe>
      </Eingaben>
    </Test>
    <Test id='3'>
      <Variablen id='3'>
      </Variablen>
      <Ausgaben id='3'>
        <Ausgabe id='3'>300.0 aufgesplittet</Ausgabe>
        <Ausgabe id='3'>*</Ausgabe>
        <Ausgabe id='3'>100er: 3</Ausgabe>
        <Ausgabe id='3'>*</Ausgabe>
        <Ausgabe id='3'>*</Ausgabe>
        <Ausgabe id='3'>*</Ausgabe>
        <Ausgabe id='3'>*</Ausgabe>
        <Ausgabe id='3'>*</Ausgabe>
        <Ausgabe id='3'>*</Ausgabe>
        <Ausgabe id='3'>*</Ausgabe>
        <Ausgabe id='3'>*</Ausgabe>
      </Ausgaben>
      <Eingaben id='3'>
        <Eingabe id='3'>300</Eingabe>
      </Eingaben>
    </Test>
    <Test id='4'>
      <Ausgaben id='4'>
        <Ausgabe id='4'>400.0 aufgesplittet</Ausgabe>
        <Ausgabe id='4'>*</Ausgabe>
        <Ausgabe id='4'>100er: 4</Ausgabe>
        <Ausgabe id='4'>*</Ausgabe>
        <Ausgabe id='4'>*</Ausgabe>
        <Ausgabe id='4'>*</Ausgabe>
        <Ausgabe id='4'>*</Ausgabe>
        <Ausgabe id='4'>*</Ausgabe>
        <Ausgabe id='4'>*</Ausgabe>
        <Ausgabe id='4'>*</Ausgabe>
      </Ausgaben>
      <Eingaben id='4'>
        <Eingabe id='4'>400</Eingabe>
      </Eingaben>
    </Test>
  </Tests>
</aufgabe>
```

16 Anhang B) komplette Hamster-Aufgabe (MR)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<aufgabe name="Logik-Labyrinth Level 2">
   <Beschreibung><html>
  <head>

  </head>
  <body>
        Finden Sie das Korn.<br></br>Verwenden Sie f&#252;r jeden Aktionsbereich eine eigene
Methode.<br></br>Hinweis: Halten Sie sich rechts.
  </body>
</html>
</Beschreibung>
   <Welt>
MMMMMMMMMMMMMMMMMM
                  M
MM MMMMMM MMMMMMMM M
M     M            M
MMM MMM MMM MMM MMMM
M M M M M M M      M
M M M M M M MMMMMM M
M M       M M    M M
M MMM MMM M M M MM M
M   M M   M M M    M
MMM M M MMM M M MM M
      M M   M M MM M
MMMMMMM MMM M M    M
M           MMM MMMM
MMM MM MMMM M      M
M    M      MMMMMMMM
M MM M MMMMMM M  M M
M MM M M    M M  M M
M  M M             1
MM M MMMMMMMMMMMMMMM
</Welt>
   <HamsterInfo>
      <Koerner>0</Koerner>
      <Position>0,1</Position>
      <Richtung>0</Richtung>
   </HamsterInfo>
   <Programm>
      def sucheKorn(): # Funktion suchekorn (Parameter: )
         istatus = 0 # setze istatus auf 0
         while not kornDa(): # Solange nicht Korn ist da mache
            vorwaerts() # vorwaerts()
            if istatus < 2: # Wenn istatus  kleiner  2 dann
               rechts() # rechts()
            else: # sonst:
               linksUm() # Um 90 Grad nach links drehen
            istatus += 1 # istatus  um ( 1) erhoehen
            if istatus == 4: # Wenn istatus  gleich  4 dann
               istatus = 0 # istatus  auf  0 setzen
      def vorwaerts():  # Funktion vorwaerts (Parameter: )
         while vorneFrei():  # Solange vorne frei mache
            vor()  # Geh einen Schritt nach vor
      def rechts():  # Funktion rechts (Parameter: )
         i = 0  # i  auf  0 setzen
         while i < 3: # Solange i  kleiner  3 mache
            linksUm() # Um 90 Grad nach links drehen
            i+=1 # i um (1) erhoehen
      sucheKorn() # sucheKorn()
         </Programm>
   <Optionen>
      <Musterloesung>Ja</Musterloesung>
      <Zeit>Nein</Zeit>
   </Optionen>
</aufgabe>
```

17 Danksagung

Zu erst möchten wir uns bei unserem betreuenden Lehrer **Herrn Dipl.-Ing. Harald Haberstroh** bedanken, unter seiner Leitung wir diese Diplomarbeit machen durften und der auch wirkliches Interesse an der Realisierung dieser Software gezeigt hat. Es waren für uns amüsante Besprechungsstunden an Freitagen, bei denen es nicht immer nur um die Diplomarbeit ging.

Weiters möchten wir uns noch bei

Herrn Dipl.-Ing. Reinhard Simon, für die Zeit die er geopfert hat, um uns eine Einführung in die Programmierung des GridBagLayouts für die GUI in Java zu geben

und bei **Herrn Dipl.-Ing. Johannes Binz** für den 3. Jahrgang des Faches Programmieren bei dem wir Java gelernt haben.

Außerdem möchten wir uns noch herzlich bei Herrn **Prof. Ditmar Lang** bedanken, welcher uns bei der Erstellung der englischen Version der Einleitung mit Rat und Tat beiseite stand.

Es bedanken sich
Martin Reiterer & Erhard Dinhobl

18 Literatur

Java für C-Programmierer
Skriptum
von Dipl.-Ing. Johannes Binz

Java – Programmieren der Grafischen Oberfläche
Skriptum
von Dipl.-Ing. Johannes Binz

Java 2 mit Methode
Auszug Kapitel 4: Drag and Drop
von Anja Austermann

Das Python-Tutorium Version 2.1
von Guido von Rossum, Fred L. Drake

Programmieren in Java
von Fritz Jobst

Python GE-PACKT
von Michael Weigend

Abenteuer Kryptologie (3. Auflage)
von Reinhard Wobst

Angewandte Datentechnik Version 8.0
Skriptum
von Dipl.-Ing. Dr. Kurt Hillebrand

19 Lebensläufe

Erhard Dinhobl
Ich wurde am 14.01.1985 in Niederösterreich, Neunkirchen geboren. Von 1991 bis 1995 ging ich vom 1. bis zum 4. Jahrgang in die Steinfeld Volksschule Neunkirchen zur Schule. Von 1995 bis 1997 besuchte ich den 1. und 2. Jahrgang des Bundesgymnasiums Neunkirchen und von 1997 bis 1999 den 3. und 4. Jahrgang des Bundesrealgymnasiums Neunkirchen. 1999 bis 2004 absolvierte ich den 1. bis 5. Jahrgang der Höheren Technischen Bundes- Lehr- und Versuchsanstalt Wr. Neustadt Abteilung Elektronische Datenverarbeitung und Organisation.

Meine Ferialpraktika: **August 2002**

K. Heyer GesmbH
Guntramserstrasse 7
2620 Natschbach – Loipersbach
Abteilung: EDV

Meine Tätigkeiten umfassten: PC-Installationen, Erstellen von Excel-Listen, Server-Installation, Lösung von Netzwerkproblemen, Arbeiten mit Proxyservern

Juli 2003

K. Heyer GesmbH
Abteilung: EDV

Meine Tätigkeiten umfassten: Behebung allgemeiner Netzwerkprobleme, Server-Installation, Ausarbeitung diverser Listen, Unterstützung des Netzwerk-Administrators

Martin Reiterer

Ich wurde am 18. November 1984 in Neukirchen geboren und bin ein sehr kommunikationsfreudiger, zielstrebiger Mensch, der immer offen für neue Sachen ist.

Meine Schulische Ausbildung umfasst:

- Volkschule Peisching 1991-1995
- Hauptschule Augasse Neunkirchen 1995-1999
- HTBLuVA Wiener Neustadt Abt. EDVO 1999-2004

Meine Ferialpraktika:

- August 2001: **Austrian Airlines Rechenzentrum** (Wien)
  Meine Tätigkeiten umfassten:
  Datenerhebung, Help-Desk-Tätigkeiten (weltweit), Datenverarbeitungstätigkeiten, PC-Installation und Reperatur

- August 2002: **Billa Dienstleistungs GesmbH** (Wr. Neudorf)
  Meine Tätigkeiten umfassten routinemäßige buchhalterische Aufgaben. In diesem Bereich war mein Hauptaufgabengebiet die buchmäßige Kontrolle der einzelnen Filialen der Bipa-Reihe.

- August 2003: **Maba Fertigteilindustrie GmbH** (Wöllersdorf)
  Meine Tätigkeiten umfassten:
  Datenerhebung, Systemverwaltung und Help-Desk-Tätigkeiten.

Hobbys:
Eines meiner Hobbys ist das Musizieren. In diesem Bereich bilde ich mich schon seit einigen Jahren fort. Mittlerweile besitze ich das goldene Jungmusiker Leistungsabzeichen auf der Tuba.
Außerdem gehe ich in meiner Freizeit gerne Segeln und Schwimmen. Weiters unternehme ich im Sommer meistens größere Radtouren.