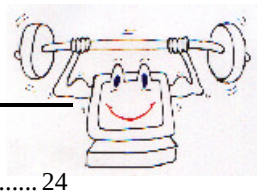
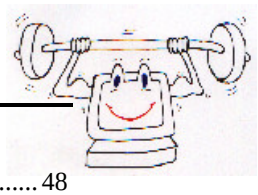
Pflichtenheft

Projektmitglieder
Erhard Dinhobl
Martin Reiterer

Autoren: Martin Reiterer
Erhard Dinhobl
Update: 10.05.2004
Version: 2.0
Gedruckt: 12.05.2004
Dateiname: Pflichtenheft Version 2.0.doc

Index





1 Zielbestimmungen

Im nachfolgenden Abschnitt werden die Ziele, die durch den Einsatz des ProgrammierTrainers erreicht werden sollen, näher beschrieben.

1.1 Musskriterien

Grundsätzlich besteht der Programmiertrainer aus zwei großen Sparten. Einerseits soll der ProgrammierTrainer dem Benutzer ein Interface zum Erstellen von Algorithmen bieten. Andererseits soll dem Lehrer ein Interface zur Verfügung stehen, in welchem dieser seine Aufgabenstellungen definieren kann.
Im nachfolgenden Abschnitt werden die Hauptkriterien, die an das Programm gestellt werden beschrieben.

1.1.1 Aufgaben laden

Grundsätzlich werden alle Aufgabenstellungen in Dateien abgespeichert.
Der Benutzer soll die Möglichkeit haben, Problemstellungen bzw. Aufgaben aus einer Datei auszulesen und in den ProgrammierTrainer zu laden. Das laden von Aufgabenstellungen kann auf zwei Arten erfolgen. Es soll möglich sein sowohl lokale Dateien zu laden als auch Dateien von einem Server zu holen.

1.1.2 Einstellmöglichkeiten – Anpassbarkeit

Dem Benutzer soll es möglich sein die Programmeinstellungen jederzeit zu ändern. Die vorgenommen Änderungen sollen sofort gespeichert und übernommen werden. Außerdem sollen die Fensterpositionen der einzelnen Fenster gespeichert werden, damit bei jedem Programmstart diese vom Benutzer nicht neu positioniert werden müssen.

Die zu ändernden Optionen sind:

1. Aufgabenpfad für lokal gespeicherte Aufgabenstellungen
2. Name des Servers, zu dem sich das Programm verbindet um Aufgaben zu suchen
3. Externer – Texteditor:
   Hier kann der Benutzer angeben, welchen externen Editor er für das Programmieren in Python verwenden möchte.
4. Timeout:
   Es wird festgelegt, wie lange ein Programm maximal laufen darf
5. Geschwindigkeit für Hamsterbeispiele soll änderbar sein.

1.1.3 ProgrammierTrainer

Dies ist eine der Hauptfunktionen des ProgrammierTrainers.
In diesem Interface wird vom Benutzer ein Algorithmus zum Lösen der Aufgabenstellung definiert.
Der Benutzer hat zwei Listenfelder vor sich.
Das eine Listenfeld ist die Auswahlliste, das andere die Algorithmusliste. Die Auswahlliste enthält die nötigen Anweisungen, die zum Erstellen des Algorithmus notwendig sind. Zur Verwirrung werden außerdem noch einige Anweisungen, die nicht benötigt werden, in die Auswahlliste eingefügt.

Mittels Drag & Drop werden die Anweisungen von der Auswahlliste in die Algorithmusliste verschoben. Zusätzlich können die in der Algorithmusliste stehenden Anweisungen durch Einrückung zu einem Block zusammengefasst werden.
In der Algorithmusliste steht also der vom Benutzer erstellte Algorithmus.

1.1.4 Python Editor

Im Python-Editor soll es dem Benutzer möglich sein, Algorithmen direkt in Python-Code zu formulieren. Dazu öffnet sich ein eigener Texteditor. Nach Fertigstellung des Algorithmus wird der vom Benutzer erstellte Algorithmus in den ProgrammierTrainer geladen.
Außerdem soll es dem Benutzer möglich sein, einen externen Texteditor zum Erstellen des Pythoncodes zu verwenden.

1.1.5 Test

In diesem Modul kann der Ersteller einer Aufgabe einige Standard-Test definieren. Diese Tests, bestehend aus Anfangswerte, Endwerten, Ein- und Ausgaben, laufen beim Benutzer später automatisiert ab. Hiermit soll dem Benutzer ein Modul zur Durchführung von automatisierten Tests bereitgestellt werden.

1.1.6 Erstellen von Aufgaben

Dem Lehrer soll es möglich sein neue Aufgabestellungen zu formulieren.
Zu diesem Zweck wird ein Aufgaben-Editor benötigt, in welchem der Lehrer seine Aufgabendateien erstellen kann. Es soll dem Lehrer möglich sein:

- Eine vollständige Musterlösung der Aufgabe zu erstellen
- Die Verfügbarkeit von Hinweisen und Musterlösung zu definieren
- Die Aufgaben-stellung textuell zu erfassen (soll intern in HTML abgespeichert werden)
- Weiters soll dem Lehrer bei der Erstellung von Hamster-Aufgaben ein grafisches Tool bereitgestellt werden, in dem er die Hamster-Welt definieren kann.

Hinweis zum Erstellen einer Hamster-Welt:
Es soll möglich sein eine neue Hamster-Welt, durch die Angabe der Welt-Dimensionen zu definieren. Weiters soll es auch möglich sein eine bereits bestehende Welt zu editieren. Es soll mittels Drag & Drop möglich sein die Hamster-Elemente in die Hamsterwelt zu verschieben. Hamster-Elemente sind Mauerelemente, Körner und der Hamster selbst.
Da der Hamster jedoch auch auf einem Korn platziert werden kann, darf die Speicherung der Hamster-Position nicht direkt in der Hamster-Welt erfolgen.
Die erstellte Hamster-Welt soll mit der Aufgabenstellung in einer gemeinsamen Datei gespeichert werden.
Idealerweise soll für den Hamster ein eigenes Panel erstellt werden, da der Hamster an vielen unterschiedlichen Stellen im Programm benötigt wird.

1.1.7 Datensicherheit – Verschlüsselung

Um zu Gewährleisten, dass die Benutzer (Schüler) die Musterlösung nicht aus der Aufgabendatei auslesen können, müssen diese Dateien mittels einem Verschlüsselungsverfahren gesichert werden können. Es soll jedoch durchaus auch möglich sein Dateien auch unverschlüsselt abzuspeichern.

Hinweis zum Verschlüsselungsverfahren: Zur Verschlüsselung der Aufgabenstellung soll ein Verschlüsselungsverfahren gewählt werden, mit dem eine unbefugte Entschlüsselung innerhalb einer vernünftigen Zeitspanne ausgeschlossen werden kann. Weiters soll die Funktionalität der Ent- und Verschlüsselung als Modul gehalten werden, um einen Austausch dieser Funktionalität zu erleichtern.

1.1.8 Systemunabhängigkeit

Es soll eine Systemunabhängigkeit gewährleistet werden. Deshalb wird zur Implementierung des ProgrammierTrainers die Programmiersprache Java herangezogen. Weiters soll zur initernen Handhabung der Algorithmen die Programmiersprache Python verwendet werden.

1.2 Wunschkriterien

In den folgenden Unterkapiteln werden die an den ProgrammierTrainer gestellten Wunschkriterien festgehalten.

1.2.1 Testumgebung

Später soll es auch möglich sein, mit dem ProgrammierTrainer Tests zu veranstalten. Diese Tests sollen zentral (vom Lehrer-PC aus) koordiniert werden können.

Der Lehrer soll vom Lehrer-PC aus den Fortschritt der Schüler sehen können. Außerdem muss zu diesem Zweck eine Zeitbegrenzung für das Erstellen eines Algorithmus eingebaut werden.

1.2.2 Speicherung von Bildern

Weiters soll es auch jederzeit möglich sein, den ProgrammierTrainer so zu erweitern, dass in der Aufgabenstellung Bilder angezeigt werden können.

Dies soll folgendermaßen geschehen:
Bild und Aufgabenstellung sollen in einer einzigen Datei gehalten werden. Beim Öffnen dieser Datei wird der Inhalt in die eigentliche Aufgabenstellung und die Bilder gesplittet. Die Bilder werden dann auf der Festplatte in einen Temporären Ordner abgelegt.

1.3 Abgrenzungskriterien

Die Aufstellung eines Webserver und die Installation des ProgrammierTrainers gehören nicht zu den Aufgaben der Autoren des ProgrammierTrainers.

1.4 Auslieferung

Die Auslieferung des ProgrammierTrainers erfolgt mittels einer auf einer CD gespeicherten Installationsroutine. Auf dieser CD befinden sich neben dem ProgrammierTrainer auch noch eine Runtine Engine für Java, ein Python-Interpreter (Jython) und eine Python-Documentation.

2 Produkteinsatz

Im folgenden Kapitel wird beschrieben, wo das zu realisierende Produkt eingesetzt werden kann, welche Umgebung gegeben sein muss um die Lauffähigkeit zu garantieren und für welche Personen das Programm eine Erleichterung darstellt.

2.1 Anwendungsbereiche

Durch das Arbeiten mit dem ProgrammierTrainer soll lediglich eine Grundlage für das programmiertechnische Denken geliefert werden. Es soll ein Gefühl für den Umgang mit den Elementen wie Variablen oder Funktionen der Programmiersprachen wie C oder Visual Basic vermittelt werden. Es kann sowohl im Heimbereich als auch für Lernzwecke eingesetzt werden.
Durch das Tool „ExampleBuilder“ kann eine Aufgabe für den ProgrammierTrainer erstellt werden. Dieses Tool sollte dort eingesetzt werden, wo allgemeine Beispiele oder gezielte Aufgaben für die Benutzer des ProgrammierTrainers erstellt werden soll.

2.2 Zielgruppen

Der ProgrammierTrainer kann sowohl für schulische Zwecke als auch aufgrund des Interesses am Programmieren, eingesetzt werden. Der ExampleBuilder sollte von denjenigen eingesetzt werden, die Aufgaben für Benutzer des ProgrammierTrainers entwerfen und diese dann 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, beherrschen (wobei dies schon das Können von Programmieren und die Erfahrung am Computer voraussetzt).

2.3 Betriebsbedingungen

Der ProgrammierTrainer sowie die Tools sind in Java implementiert und können somit auf jeder Plattform laufen, die eine JVM (Java Virtual Machine) besitzt. Die vorauszusetzende Hardware ist ebenfalls nicht relevant für den Betrieb des ProgrammierTrainers sowie den Tools. Die Betriebszeit des ProgrammierTrainers hängt davon ab, wie lange der Benutzer bereit ist seine Zeit zu opfern um mit dem ProgrammierTrainer zu arbeiten.
Der ProgrammierTrainer kann aber auch die von einer vorhandenen Python-Installtion vorliegenden Bibliotheken verwenden.

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.

Es wird dadurch jedoch nicht garantiert, dass der ProgrammierTrainer keinen einzigen Bug aufweist. Schließlich gibt es kein Programm, welches in der Lage ist dies zu garantieren.

3.4 Organisationsumgebung

Der ProgrammierTrainer soll im Programmierunterricht eingesetzt werden. Er wird für den Einstieg in das komplexe, ablauforientierte Denken verwendet.

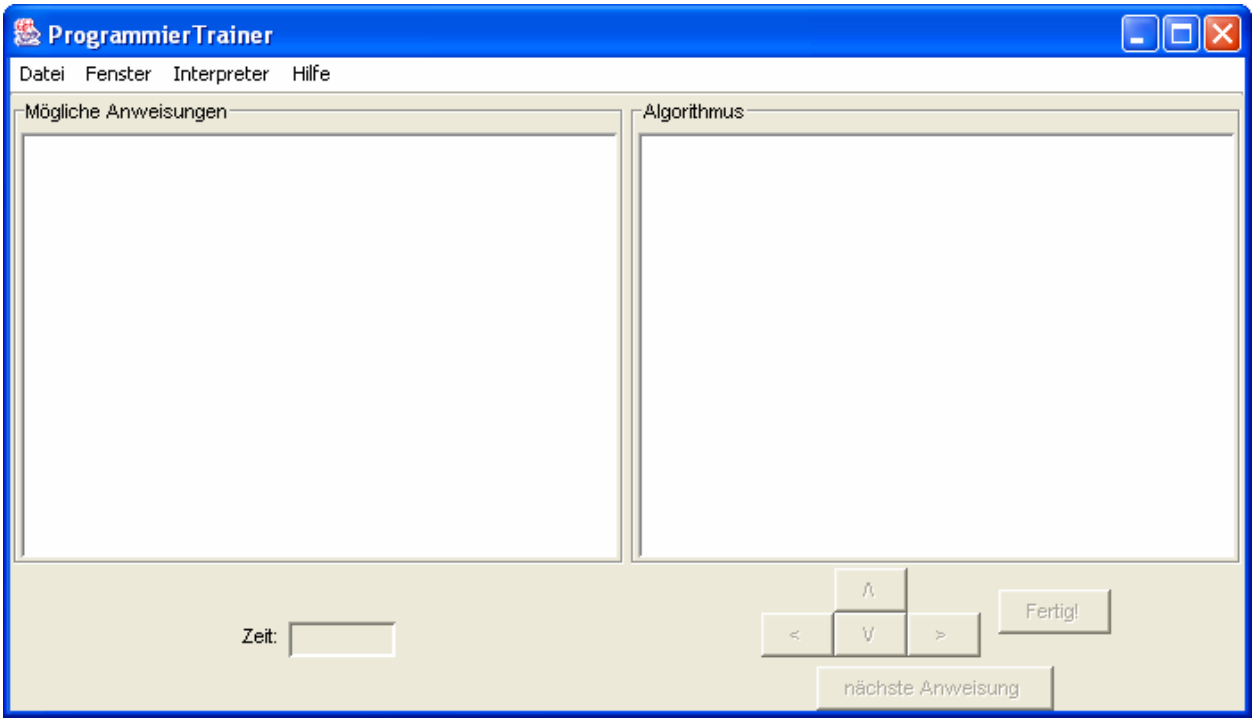
4 Produktfunktionen

Im folgenden Kapitel wird das Produkt aus Benutzersicht erklärt. Es werden alle Funktionen die die Software zur Verfügung stellt beschrieben.

4.1 ProgrammierTrainer (ED)

In den folgenden Unterkapiteln wird die Funktionsweise des ProgrammierTrainers beschrieben.

4.1.1 Hauptformular



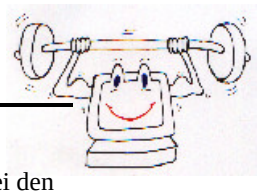
Das Hauptformular soll folgende, der Aufgabe entsprechende, Funktionen haben:

4.1.1.1 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 der Sammlung aller 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 der Sammlung der Anweisung, mittels Drag&Drop, verschoben werden. Hierbei wird die Anweisung analog zu vorhin von der Algorithmus-Listbox ausgeblendet und in die Listbox mit der Sammlung der Anweisungen verschoben.

4.1.1.2 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. Aufgrund dessen 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.

4.1.1.3 Shortcuts

Tasten	**Funktion**
Strg + O	Öffnet den Aufgabeöffnen-Dialog (siehe 4.1.5)
Strg + E	Öffnet den Editor (siehe 4.1.7)
Strg + Q	Beenden den ProgrammierTrainer
Strg + A	Öffnet den Aufgabendialog (siehe 4.1.3)
Strg + S	Öffnet den Optionendialog (siehe 4.1.6)
Strg + 5	Startet das Programm im Ausführen-Modus
Strg + 6	Startet das Programm im Debug-Modus
Strg + 7	Testet das Programm
Strg + X	Führt einen Anweisung während des Debug-Modus aus
Strg + N	Führt die nächste Anweisung im Debug-Modus aus
/\ (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

4.1.2 Menüpunkte

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

4.1.2.1 Datei

Menüeintrag Datei.

4.1.2.1.1 Aufgabe öffnen

Es wird der Aufgabe-Öffnendialog (beschrieben in 4.1.5) angezeigt und die dort ausgewählte Aufgabe geöffnet.

4.1.2.1.2 Python – Code

Es wird ein Editor geöffnet (siehe 4.1.7) 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 und die Listbox, welche die Sammlung aller Anweisungen enthält, geleert. Nun soll es soll nicht möglich sein Anweisungen zu verschieben.

4.1.2.1.3 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
```

4.1.2.1.3.1 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

4.1.2.1.4 Beenden

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

4.1.2.2 Fenster

Menüeintrag Fenster.

4.1.2.2.1.1 Aufgabe

Es soll der Aufgabedialog (siehe 4.1.3) angezeigt werden.

4.1.2.2.1.2 Optionen

Es soll der Optionendialog (siehe 4.1.6) angezeigt werden.

4.1.2.2.1.3 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.

4.1.2.2.1.4 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.

4.1.2.3 Interpreter

Menüeintrag Interpreter.

4.1.2.3.1 Run

Das entworfenen Programm wird ausgeführt unter der Berücksichtigung des Timeouts (siehe 4.1.6 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.

4.1.2.3.2 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.

4.1.2.3.3 Abbruch

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

4.1.2.3.4 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 liefern und das Programm unter Umständen einen bestimmten Output liefern, was somit überprüft werden kann. (kein deutscher Satz)

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. Als Endwert kann alles eingegeben werden, was in einem If-Statement als Vergleich zu einer bestimmten Variable in Python zulässig ist.

  Die Ergebnisse der Testläufe werden dann in einem Fenster angezeigt.

4.1.2.3.5 letzte Testergebnisse

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

4.1.2.3.6 Eval

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

4.1.2.3.7 nächste Anweisung

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

4.1.2.4 Hilfe

Menüeintrag Hilfe.

4.1.2.4.1 Hilfe zu ProgrammierTrainer

Es wird der Hilfedialog (siehe 4.1.8) mit einer HTML-Seite, deren Inhalt die Hilfe zum ProgrammierTrainer beinhaltet, angezeigt.

4.1.2.4.2 Info

Es wird der Infodialog (siehe 4.1.9) angezeigt.

4.1.2.4.3 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.

4.1.2.4.4 Lösung

Falls in dieser Aufgabe die Musterlösung (so wie der Aufgabenersteller meint es gelöst zu haben) zur Verfügung steht (wird bei der Aufgabenerstellung im ExampleBuilder durch den Ersteller festgelegt ob diese für den Benutzer verfügbar ist oder nicht), soll beim Ausführen dieses Menüpunktes das Ausgabefenster (siehe 4.1.4) geöffnet werden und dort in der vorhandenen Listbox die Musterlösung angezeigt werden.

4.1.3 Aufgabendialog

Der Aufgabendialog soll folgende Funktionen der Aufgabe betreffend haben:

4.1.3.1 Eingabe/Ausgabe-Aufgabe

Es wird lediglich die Beschreibung der zu lösenden Aufgabe dargestellt.

4.1.3.2 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.

4.1.4 Ausgabefenster

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

[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.

4.1.4.1 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.

4.1.4.2 Hamster-Aufgabe

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

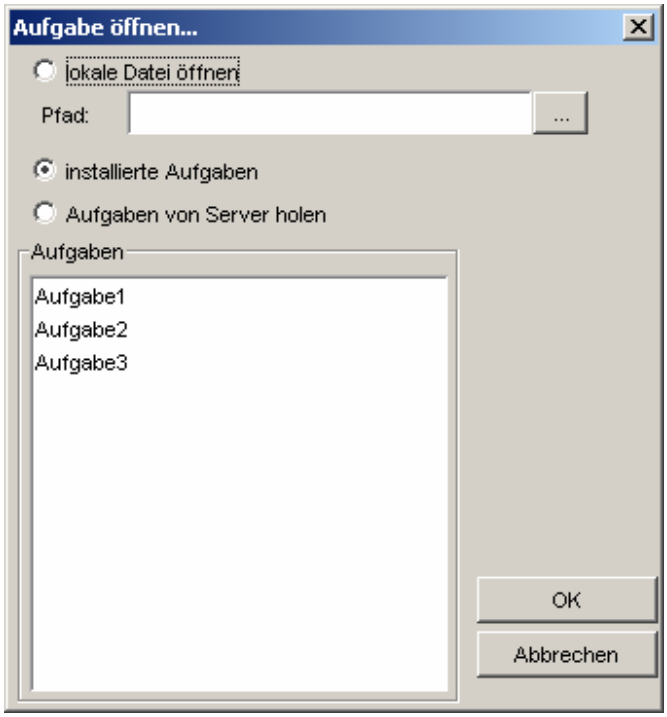
- Debug-Modus

  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 demonstriert bekommt.

- Ausführen

  Beim Ausführen wird das Hamster-Programm ausgeführt. Jede Anweisung wird nach der Anzahl der Millisekunden, die im Optionendialog (siehe 4.1.6) angegeben wurden, ausgeführt. Es werden Ausgaben die das Programm produziert in die Listbox des Ausgabefensters aufgenommen.

4.1.5 Aufgabe-Öffnendialog



Der Aufgabe-Öffnendialog soll folgenden Funktionen bieten:

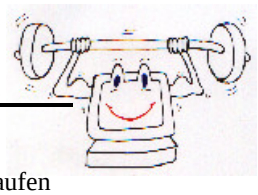
- Es soll möglich sein eine lokale Aufgaben-Datei zu öffnen
- Es soll möglich sein alle Aufgaben-Dateien, die bereits installiert sind[2], aufzulisten. Hierbei wird der jeweilige Titel der Aufgabe und in Klammer der Dateiname in der Listbox angezeigt. Davon kann man eine Aufgabe auswählen, die geöffnet 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.

4.1.6 Optionendialog

Folgende Optionen sollen festgelegt werden können, welche dann in die optionen.xml-Datei übernommen werden:

- Es soll festgelegt werden können, welches lokale Verzeichnis (bezieht sich auf 4.1.5) das Programm nach Aufgaben durchsucht.
- Es soll festgelegt werden können, zu welchen Server (bezieht sich auf 4.1.5) sich das Programm verbindet um eine Datei[4] auszulesen, welche die Informationen zu den am Server liegenden Aufgaben enthält.
- Es soll möglich sein einen externen Programm-Editor anzuführen, um statt des internen Editors (siehe 4.1.7), 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, dass in der optionen.xml-Datei definiert ist.
[3] in der optionen.xml-Datei defniert
[4] Diese Datei muss den Name „pt_aufgaben.con“ besitzen und in dem Verzeichnis liegen, welches als Server angegeben ist.



- Es soll festgelegt werden können, 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.
- Es soll festgelegt werden können 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.

4.1.7 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 soll noch nachgefragt werden, ob die Änderungen übernommen werden sollen.

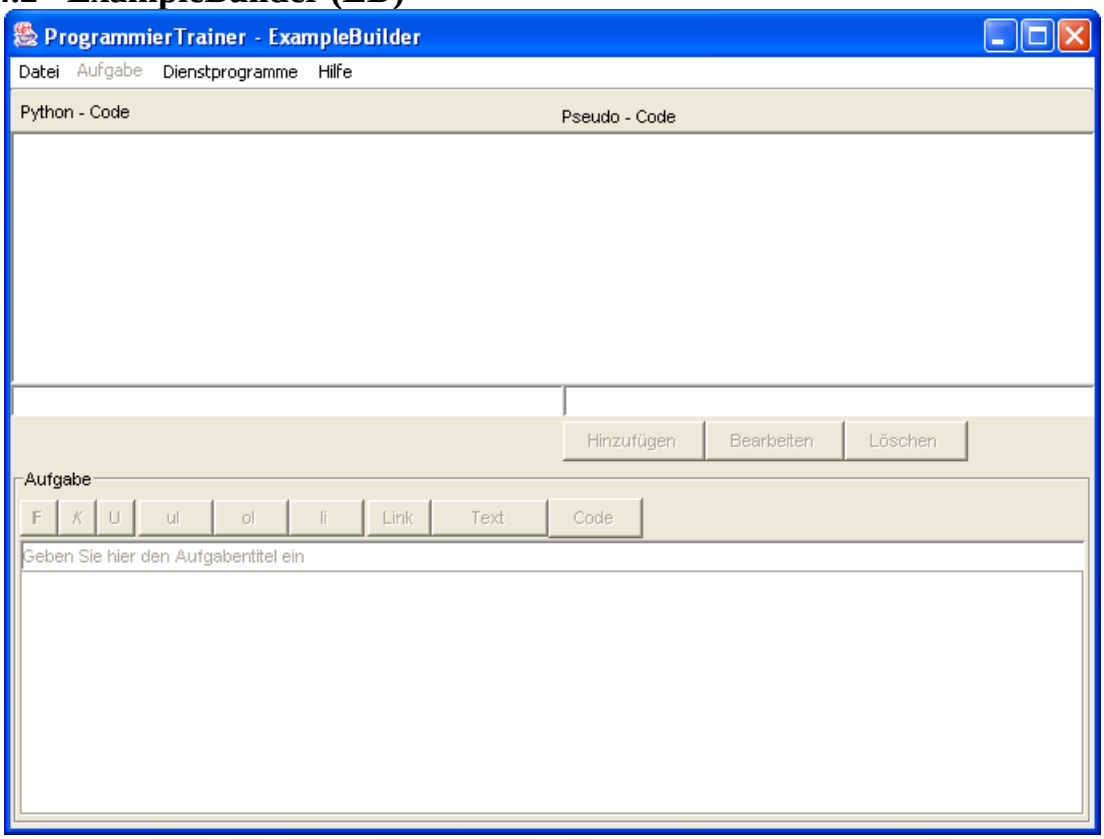
4.1.8 Hilfedialog

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

4.1.9 Infodialog

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

4.2 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 4.1) beschrieben.

4.2.1 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

4.2.2 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 Fakeanweisung sein. D.h. sie ist nur zur Verwirrung des Benutzers, der die Aufgabe löst, vorhanden. Eine Fakeanweisung 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 Fakeanweisung.
Im unteren Teil des Formulars kann man den Text zur Aufgabe erstellen, der dann später im Aufgabendialog (siehe 4.1.3) 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.

4.2.3 Menüpunkte

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

4.2.3.1 Datei

Menüeintrag Datei.

4.2.3.1.1 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.

- 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.

4.2.3.1.2 Aufgabe öffnen…

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

4.2.3.1.3 Aufgabe speichern…

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

4.2.3.1.4 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.

4.2.3.1.5 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 4.1.2.3 „Test“ beschrieben ist.

4.2.3.1.6 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.

4.2.3.2 Aufgabe

Menüeintrag Aufgabe.

4.2.3.2.1 Ergebnisüberprüfung

Es wird der Ergebnisüberprüfungsdialog (siehe 4.2.4) angezeigt.

4.2.3.2.2 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 Pyton-Code in verschiedene Pseudo-Codeformen zu übersetzen, wie formale Sprache oder auch eine Programmiersprache als Pseudo-Code.

4.2.3.2.3 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.

4.2.3.2.4 Hinweise verfügbar

Mit dieser Option kann festgelegt werden, ob der Benutzer beim Lösen der Aufgabe Hinweise (siehe 4.1.2.4) bekommen kann.

4.2.3.2.5 Musterlösung

Mit dieser Option kann festgelegt werden, ob der Benutzer die Musterlösung (siehe 4.1.2.4) einsehen darf oder nicht.

4.2.3.3 Dienstprogramme

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

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

4.2.3.3.1 Verzeichnis Indizieren

Mit diesem Dienstprogramm ist es möglich, automatisch eine Datei (siehe 4.1.5) 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 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 Datei 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.

4.2.3.4 Hilfe

Menüeintrag Hilfe.

4.2.3.4.1 Hilfe zu ExampleBuilder

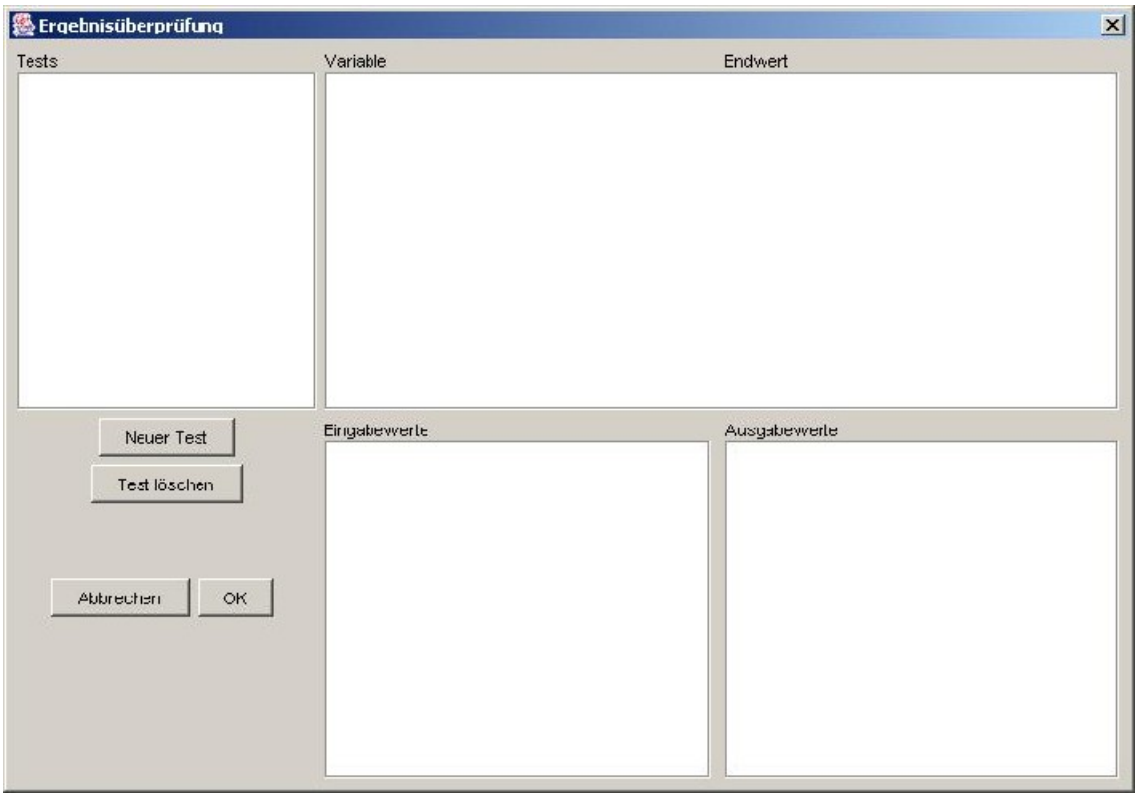
Es wird der Hilfedialog (siehe 4.1.8) mit einer HTML-Seite, deren Inhalt die Hilfe zum ExampleBuilder beinhaltet, angezeigt.

4.2.3.4.2 Info

Es wird der Infodialog (siehe 4.2.5) angezeigt.

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

4.2.4 Ergebnissüberprüfungsdialog



4.2.4.1 Shortcuts

Tasten	Funktion
Entf	löscht die markierte Zeile aus der Eingabe-, Ausgabe- und Variablenendwertetabelle

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 4.1.2.3 „Test“) Es können z.B. für ein Programm mit folgendem Pseudo-Code:

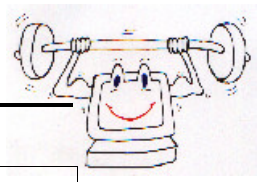
```
Variable x vom Benutzer einlesen
multiplizieren der Variable x mit 2 und in y speichern
y ausgeben
```

…Testfälle erstellt werden, wo anstatt vom Benutzer einen Wert zu verlangen, jener aus der Liste „Eingabewerte“ genommen wird. Somit ist x mit einem Wert belegt. Nach Ablauf des entworfenen Programms wird y ausgegeben, wobei dieser Ausgabewert doppelt so groß wie x sein muss. Die Testfälle könnten somit folgendermaßen aussehen:

Testfall Nr. 1:

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

Testfall Nr. 2:



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

Die Testfälle werden links in einer Listbox angezeigt. Beim Markieren eines Testfalles werden rechts in den Listboxen alle für diesen Testfall eingetragenen Variablen angezeigt. Via Doppelklick in eine Listbox kann der Zelleneintrag modifiziert werden. Mithilfe der Entfernen-Taste kann die Zeile einer betreffenden Listbox gelöscht werden.

4.2.5 Infodialog

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

5 Zusatzmodul – Hamster-Dialog (MR)

Dieser Teil der Diplomarbeit entstand erst währen der Implementierungsphase. Zu dieser Zeit wurde uns nämlich das volle Ausmaß der Vorteile eines Hamster-Panels klar. Deshalb äußerte unser betreuender Lehrer, Herr DI 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 anderen Java-Programmen.
- Der Hamster-Dialog kann aus JYTHON heraus direkt angesprochen werden.
- Dem Programmieranfänger werden die Auswirkungen seiner Programmiertätigkeiten sofort angezeigt.

5.1 Einbindung in Jyhton

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

Folgende Schritte sind notwendig:

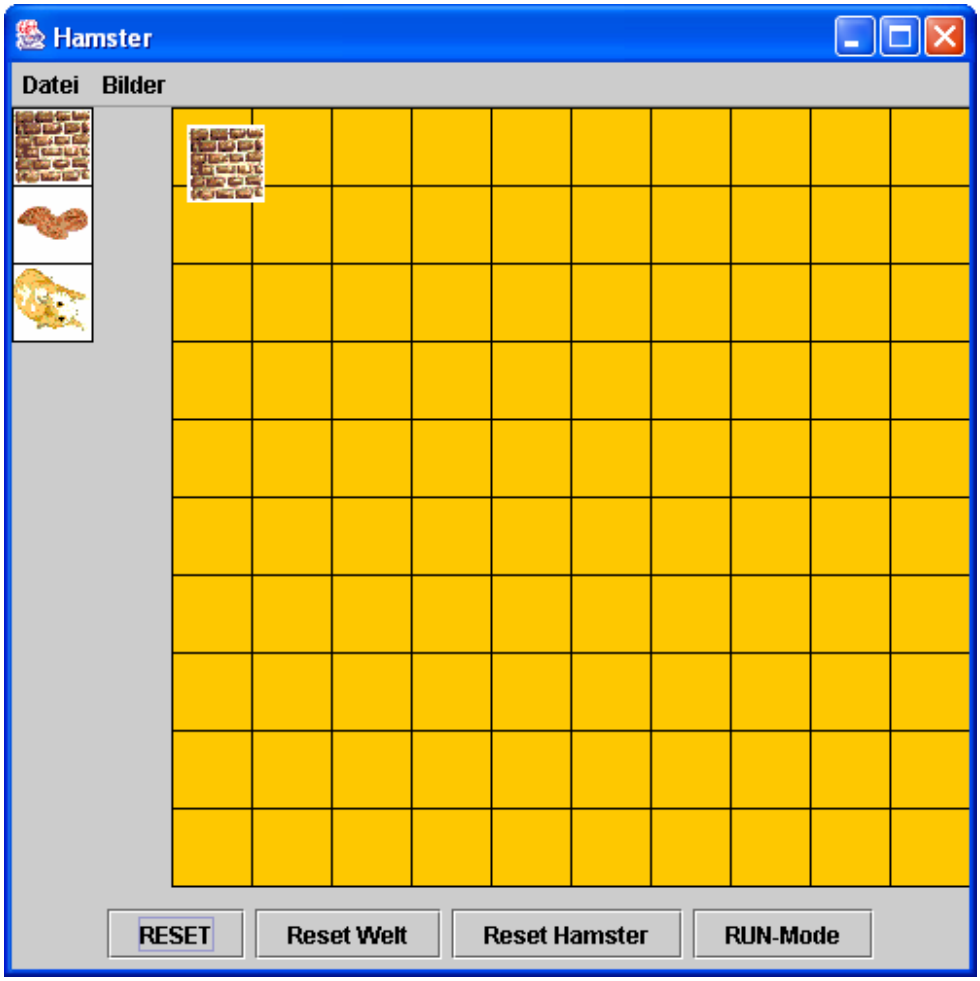
1. Die Umgebungsvariable CLASSPATH muss um das Java-Archiv des Hamster-Dialoges erweitert werden.
2. Im nächsten Schritt muss die Jython-Konsole gestartet werden.
3. Ist die Jython-Konsole gestartet, so muss jetzt nur noch die Hamster-Dialog-Klasse importiert werden.

   Dies geschieht mit dem folgenden Kommando:

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

5.2 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.



5.2.1 Menüpunkte

Die Menüpunkte sind folgendermaßen gegliedert:

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

→ Bilder
  → laden

5.2.2 Aufgabendatei öffnen

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

5.2.3 Bilder laden

Mit diesem Menüpunkt ist es dem Benutzer möglich die Standard-Bilder des Hamster-Panels 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.

5.3 Schaltflächen

Der Hamster-Dialog stellt folgende Menüpunkte bereit:

RESET
Reset Welt
Reset Hamster
Run-/Edit-Mode

5.3.1 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 5.4.1 und 5.4.2.

5.3.2 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 5.4.1 und 5.4.2.

5.3.3 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 5.4.1 und 5.4.2.

5.3.4 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 5.4.1 und 5.4.2.

5.4 Bedienung des Hamster-Dialogs

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

- Edit-Mode
- Run-Mode

5.4.1 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 die 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 nötig.
Weiters kann der Benutzer auch Korn- und Mauerelemente in die Hamster-Welt verschieben.

Achtung: Es können maximal 9 Körner auf demselben Platz liegen!

5.4.2 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.

5.5 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 „true“ zurück. Schlägt das Laden fehl, so liefert diese Methode „false“ 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°	keine
`vorneFrei()`	`int`	Überprüft, ob ein weiterer Schritt vorwärts möglich ist	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.

5.6 Exceptions

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

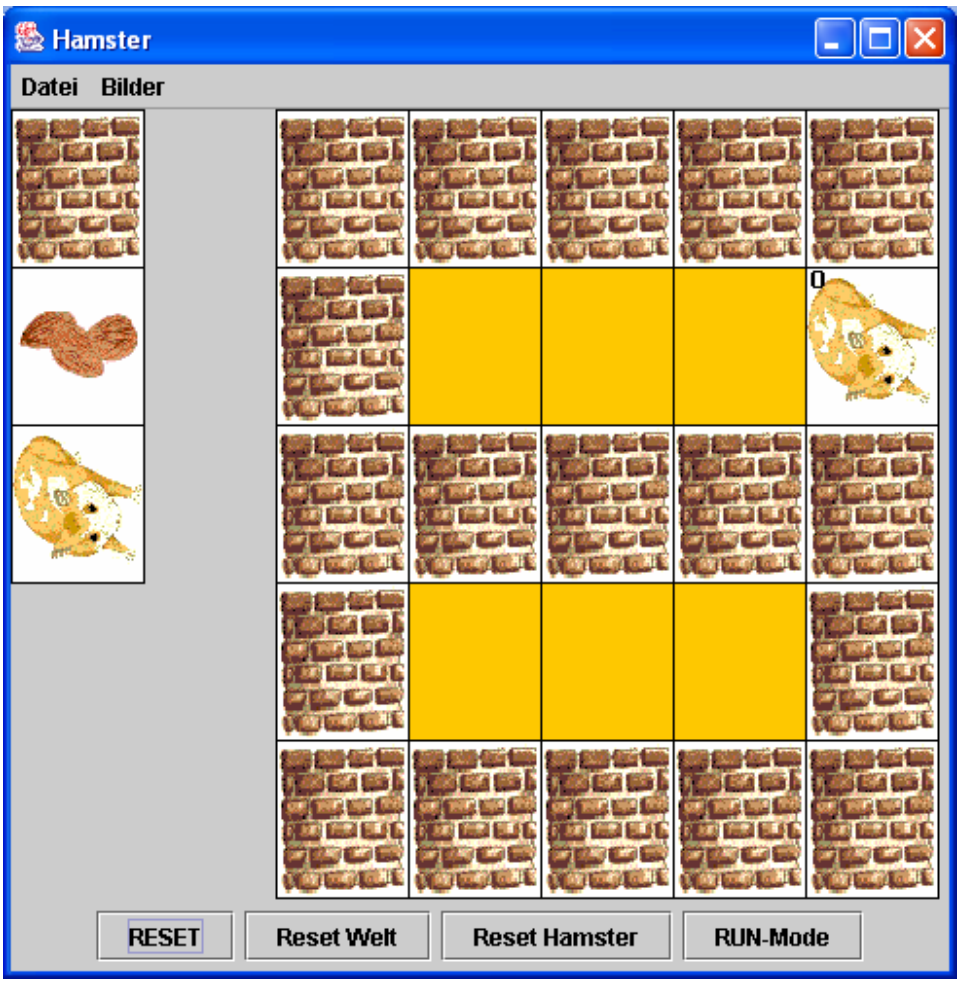
5.6.1 MauerException

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

5.6.1.1 Beispiel

Folgendes Beispiel soll diese Situation veranschaulichen.

5.6.1.1.1 Ausgangslage:



5.6.1.1.2 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° dreht und erst dann nach vor bewegt.

5.6.1.1.3 Mögliche Vorgehensweise

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

Programmausschnitt:

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

5.6.2 KornException

Eine KornException wird ausgeworfen, wenn es Probleme mit den Körnern gibt. Es können verschiedene Fehler auftreten. Einerseits tritt eine KornException auf, wenn der Benutzer versucht ein Korn von einem Feld aufzunehmen, auf dem sich in Wirklichkeit jedoch Keines befindet.
Andererseits kann es zu einer KornException kommen, wenn der Benutzer versucht mehr als neun Körner auf einem Feld zu positionieren.

5.6.2.1 Beispiel

Folgendes Beispiel soll diese Situation veranschaulichen.

5.6.2.1.1 Ausgangssituation:

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

5.6.2.1.2 Aktion

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

5.6.2.1.3 Mögliche Vorgehensweise

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

Programmausschnitt:

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

6 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.

6.1 Data Dictionary

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

6.1.1 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
→ *Körner*	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 beim Start 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.

Data Dictionary - Hamster – Aufgabendatei 1

[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.

6.1.2 E/A-Dateien

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

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 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 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 werden alle Informationen betreffend einer Variable festgehalten
→ *Attribut: name*	Enthält den Namen der Variable
→ *Attribut: end*	Hier wird der Endwert dieser Variable festgehalten.
→ *Attribut: id*	Identifikationsnummer des Tests
→ *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

Data Dictionary - E/A-Aufgabendatei 1

*) können mehrmals vorkommen

6.1.3 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	Hier wird die Anordnung der einzelnen Fenster festgehalten. Pro Fenster werden die X und Y-Werte 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 4.1.6)
delay	Hier wird festgehalten wie viele Millisekunden zwischen zwei Ausführungen von Hamsternaweisungen vergehen. (siehe 4.1.6)

Data Dictionary - Optionen Datei 1

6.1.4 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.

Data Dictionary - Zuordnungsdatei

7 Benutzungskonzept

Hier wird für den ProgrammierTrainer das Benutzungskonzept aus Benutzersicht näher beschrieben.

7.1 Interfaces

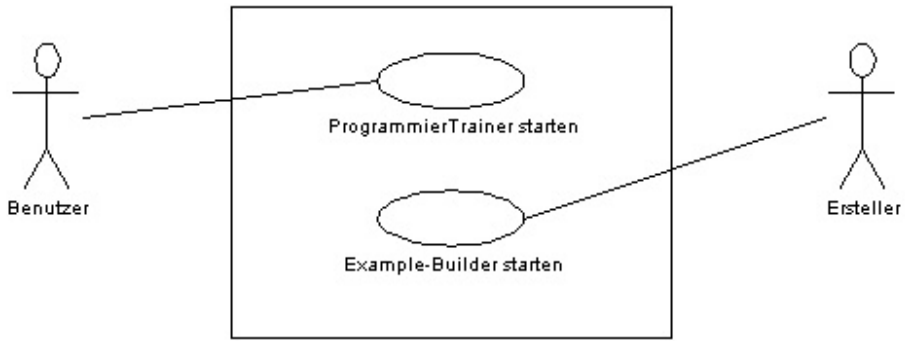
siehe Prototyp

7.2 Use-Case-Diagramm

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

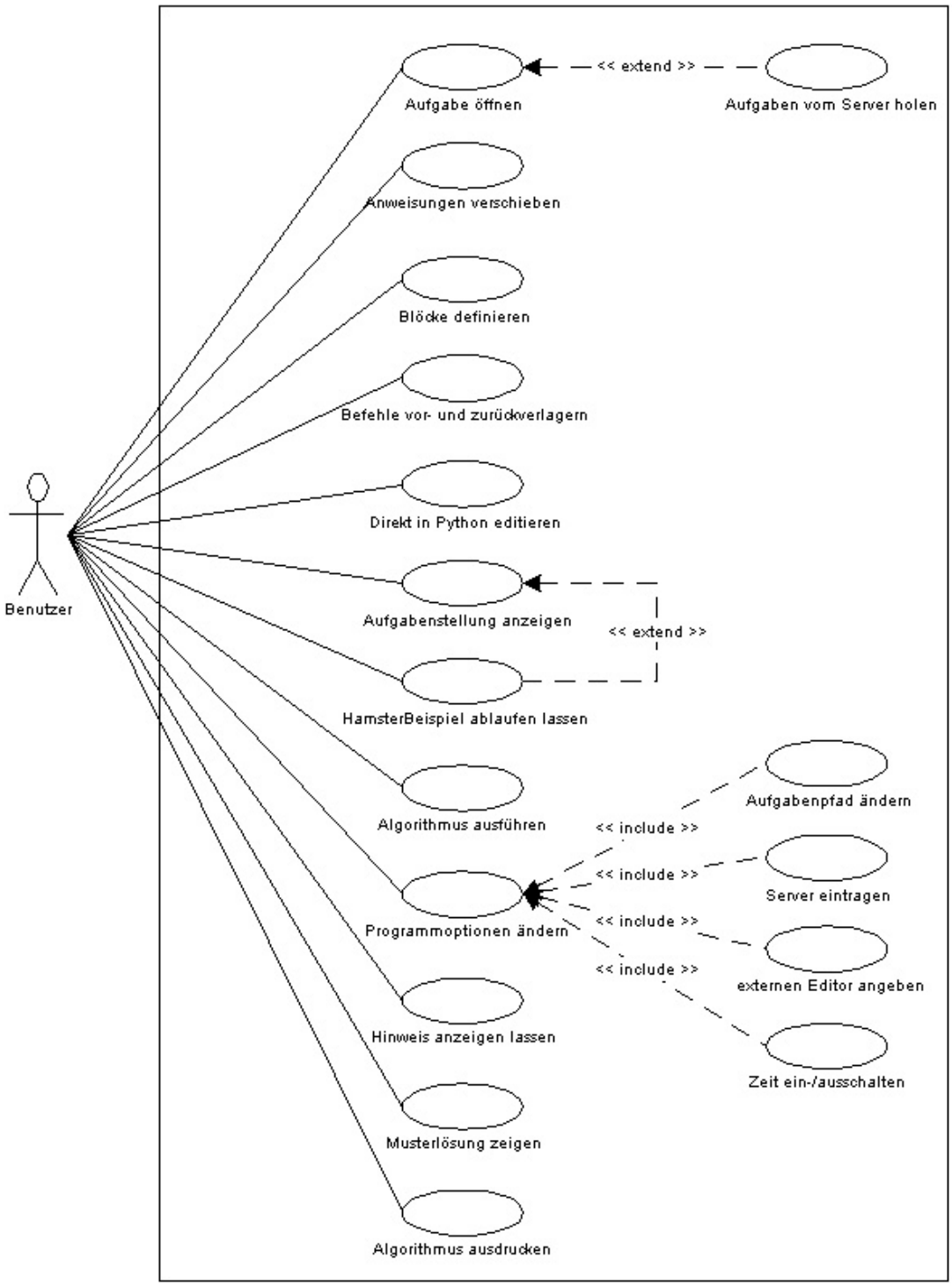
7.2.1 Übersicht aller Teilprogramme

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



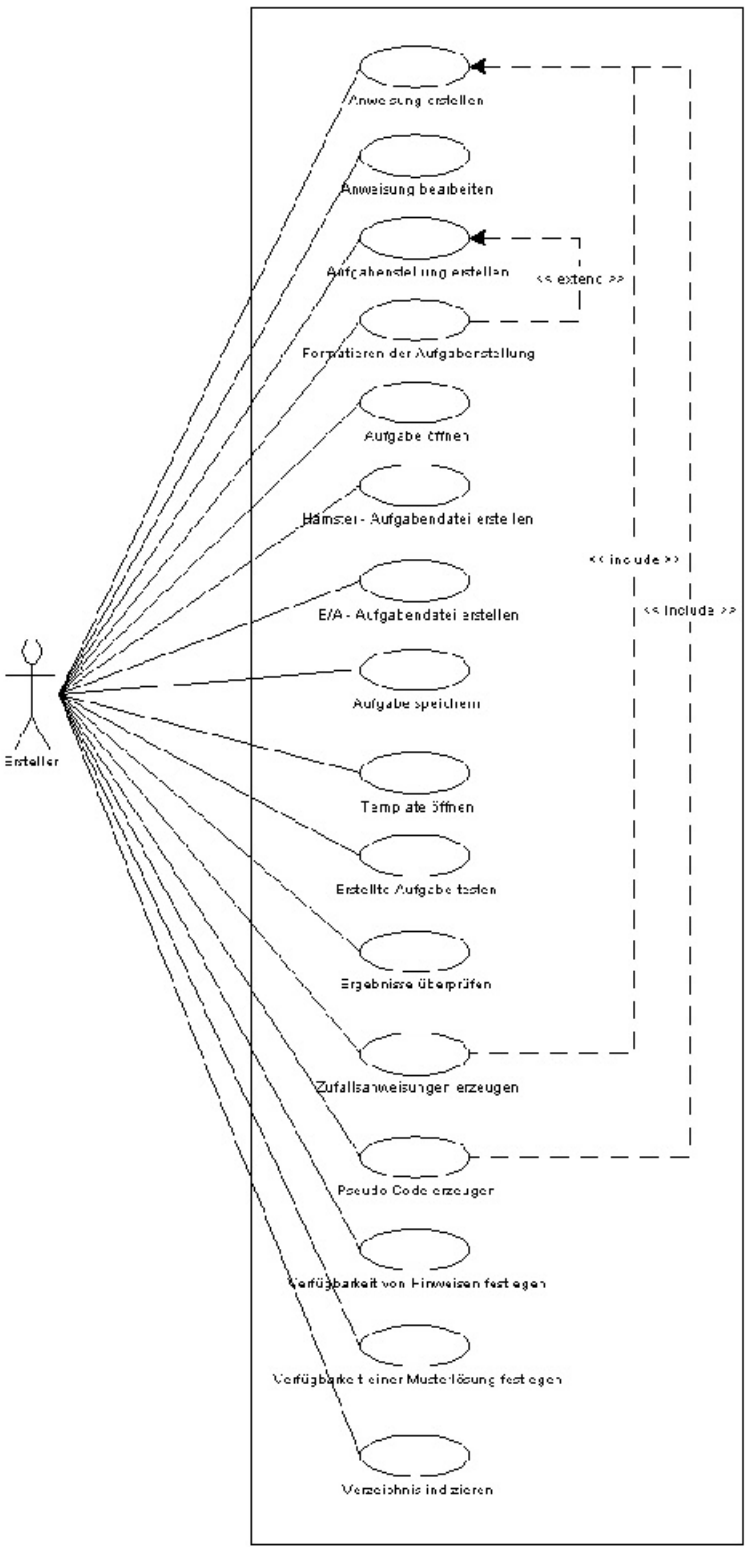
7.2.2 Hauptprogramm – ProgrammierTrainer

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



7.2.3 Example-Builder

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



7.3 Beschreibung

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

7.3.1 ProgrammierTrainer

Nähere Informationen zu den USE-CASES des ProgrammierTrainers

7.3.1.1 Aufgabe öffnen

Name:	Aufgabe öffnen
Zweck:	Zum öffnen einer beliebigen Aufgabenstellung.
Beschreibung:	Der Benutzer öffnet eine von ihm gewünschte Aufgabenstellung. Dies 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 in 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.

Use-Case – Aufgabe öffnen

7.3.1.2 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.

Use-Case – Aufgaben vom Server holen

7.3.1.3 Anweisungen verschieben

Name:	Anweisungen verschieben
Zweck:	Durch das Verschieben der Anweisungen von der linken Listbox in die rechte Listbox hat der Benutzer die Möglichkeit einen Algorithmus zu formulieren.
Beschreibung:	Mittels Drag&Drop hat der Benutzer die Möglichkeit Anweisungen zu Verschieben. Mittels diesem Vorgang werden vom Benutzer jene Anweisungen in die rechte Listbox verschoben, welche er zum Erstellen eines Algorithmus benötigt.
Vorbedingungen:	Der Benutzer hat eine Aufgabenstellung in den ProgrammierTrainer geladen.
Nachbedingungen:	Der Benutzer hat die, Seinäsachtens benötigten Anweisungen in die Algorithmusliste verschoben.
Alternative Abläufe:	Es gibt zwei Möglichkeiten: 1. Der Benutzer Verschiebt eine Anweisung innerhalb der rechten Listbox: Dies wird zum Sortieren der Anweisungen benötigt. 2. Der Benutzer Verschiebt eine Anweisung innerhalb der linken Listbox: Dieser Vorgang hat keine Auswirkungen.

Use-Case – Anweisungen verschieben

7.3.1.4 Blöcke definieren

Name:	Blöcke definieren
Zweck:	Der Benutzer hat die Möglichkeit Anweisungsblöcke zu definieren
Beschreibung:	Mithilfe der Pfeiltasten im ProgrammierTrainer-Interface 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 ist bereits mindestens eine Anweisung in der Algorithmus-Liste.
Nachbedingungen:	Es wurde ein Block definiert.
Alternative Abläufe:	Eine vorgenommene Blockdefinition wurde rückgängig gemacht.

Use-Case – Blöcke definieren

7.3.1.5 Befehle vor- und zurückverlagern

Name:	Befehle vor- und zurückverlagern
Zweck:	Zum Ändern der Reihenfolge der Abarbeitung der Anweisungen.
Beschreibung:	Der Benutzer ändert die Reihenfolge der in der Algorithmus-Liste stehenden Anweisungen. Dies kann auf verschiedene Arten geschehen: 1. Pfeiltasten im Interface 2. Drag&Drop innerhalb der Algorithmus-Liste 3. Tastenkürzel
Vorbedingungen:	Mindestens zwei Anweisungen in der Algorithmus-Liste.
Nachbedingungen:	Die Reihenfolge der Anweisungen wurde verändert.
Alternative Abläufe:	-

Use-Case – Befehle vor- und zurückverlagern

7.3.1.6 Direkt in Python editieren

Name:	Direkt in Python editieren
Zweck:	Mithilfe eines in den ProgrammierTrainer eingebauten oder externen Editor 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 hat er die Möglichkeit diese in den ProgrammierTrainer zu übernehmen, um seinen Algorithmus testen zu können. Andererseits hat der Benutzer jedoch auch die Möglichkeit, den von ihm erstellten Algorithmus zu verwerfen um wieder in den ProgrammierTrainer zurückzukehren.
Vorbedingungen:	Eine Aufgabe wurde geöffnet.
Nachbedingungen:	Es gibt drei Möglichkeiten: 1. Der Benutzer hat keinen Algorithmus erstellt: Hat keine Auswirkungen. 2. Der Benutzer hat den von ihm erstellten Algorithmus in den ProgrammierTrainer geladen. 3. Der Benutzer hat den von ihm erstellten Algorithmus nicht in den ProgrammierTrainer geladen: Dies hat ebenfalls keine Auswirkungen auf den ProgrammierTrainer.
Alternative Abläufe:	Der Benutzer hat in seinem Algorithmus einen Syntax-Fehler. Reaktion: Er gelang zurück zum Editor.

Use-Case – Direkt in Python editieren

7.3.1.7 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 einer 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:	-

Use-Case – Aufgabenstellung anzeigen

7.3.1.8 Hamsterbeispiel ablaufen lassen

Name:	Hamsterbeispiel ablaufen lassen
Zweck:	Der Benutzer hat einerseits die Möglichkeit, die vom Ersteller kreierte Musterlösung in einem Hamster-Panel ablaufen zu lassen. Andererseits hat er auch die Möglichkeit seinen eigenen Algorithmus in diesem Panel ablaufen zu lassen.
Beschreibung:	Der Benutzer sieht wie von einem Hamster Aktionen ausgeführt werden. In diesem Hamster-Panel bewegt sich der Hamster in einem durch den Benutzer eingestellten delay (Verzögerung).
Vorbedingungen:	Hamsterbeispiel wurde geladen.
Nachbedingungen:	Dem Benutzer wurden die Auswirkungen grafisch dargestellt.
Alternative Abläufe:	-

Use-Case – Hamsterbeispiel ablaufen lassen

7.3.1.9 Algorithmus ausführen

Name:	Algorithmus ausführen
Zweck:	Der Benutzer hat die Möglichkeit, die von ihm erstellten Anweisungen auszuführen und das Ergebnis seiner Aktionen zu sehen.
Beschreibung:	Der vom Benutzer erstellte Algorithmus wird vom ProgrammierTrainer interpretiert und ausgeführt. Der Benutzer hat die Möglichkeit seinen Algorithmus einerseits im RUN- andererseits im DEBUG-Modus auszuführen. RUN: Die Anweisungen werden nacheinander in einem vom Benutzer definierten delay ausgeführt. DEBUG: Der Benutzer hat die Möglichkeit die Anweisungen seines Algorithmus einzeln durchzugehen. Die durch den Algorithmus erzeugten Ausgaben werden in einer eigenen Liste ausgegeben.
Vorbedingungen:	Der Benutzer hat mindestens eine Anweisung in der Algorithmus-Liste.
Nachbedingungen:	Der Benutzer sieht die Ergebnisse seines Algorithmus.
Alternative Abläufe:	Der Benutzer hat einen Fehler in seinem Algorithmus. In diesem Fall ist es dem Benutzer nicht möglich den Algorithmus auszuführen.

Use-Case – Algorithmus ausführen

7.3.1.10 Optionen ändern

Name:	Optionen ändern
Zweck:	Der Benutzer soll die Möglichkeit haben den ProgrammierTrainer seinen Ansprüchen gemäß zu konfigurieren.
Beschreibung:	Der Benutzer kann folgende Einstellungen ändern: 1. Aufgabenpfad 2. Server 3. externer Editor 4. Zeit ein-/ausschalten
Vorbedingungen:	ProgrammierTrainer gestartet
Nachbedingungen:	Einstellungen wurden geändert.
Alternative Abläufe:	Einstellungen wurden nicht geändert.

Use-Case – Optionen ändern

7.3.1.11 Aufgabenpfad ändern

Name:	Aufgabenpfad ändern
Zweck:	Der Benutzer hat die Möglichkeit den Aufgabenpfad zu ändern
Beschreibung:	Mit dem Aufgabenpfad wird angegeben, wo der ProgrammierTrainer nach Aufgabenstellungen suchen soll.
Vorbedingungen:	ProgrammierTrainer wurde gestartet und Optionen-Dialog aufgerufen.
Nachbedingungen:	Der Aufgabenpfad wurde geändert.
Alternative Abläufe:	Der Aufgabenpfad wurde nicht geändert.

Use-Case – Aufgabenpfad ändern

7.3.1.12 Server eintragen

Name:	Server eintragen
Zweck:	Der Benutzer hat die Möglichkeit die Adresse eines Aufgaben-Servers einzutragen.
Beschreibung:	Mit der Server-Adresse wird vom Benutzer der Server definiert, auf welchem der ProgrammierTrainer nach Aufgabenstellungen suche soll.
Vorbedingungen:	ProgrammierTrainer wurde gestartet und Optionen-Dialog aufgerufen.
Nachbedingungen:	Die Server-Adresse wurde eingetragen/geändert.
Alternative Abläufe:	Die Server-Adresse wurde nicht eingetragen/geändert.

Use-Case – Server eintragen

7.3.1.13 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:	ProgrammierTrainer wurde gestartet und Optionen-Dialog aufgerufen.
Nachbedingungen:	Editor-Dateipfad wurde eingetragen/geändert.
Alternative Abläufe:	Editor-Dateipfad wurde nicht eingetragen/geändert.

Use-Case – Externen Editor angeben

7.3.1.14 Zeit ein-/ausschalten

Name:	Zeit ein-/ausschalten
Zweck:	Der Benutzer hat die Möglichkeit, mitzustoppen, wie lange er benötigt um die Aufgabe zu lösen.
Beschreibung:	Wird die Zeit eingeschaltet, so wird die Zeit wie bei einer Stoppuhr im Hauptformular angezeigt. Die Zeit kann jedoch nur ausgeschaltet werden, wenn dies vom Ersteller der Aufgabenstellung erlaubt wurde.
Vorbedingungen:	ProgrammierTrainer wurde gestartet und Optionen-Dialog aufgerufen.
Nachbedingungen:	Zeit wurde eingeschaltet.
Alternative Abläufe:	Zeit wurde ausgeschaltet.

Use-Case – Zeit ein-/ausschalten

7.3.1.15 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.
Vorbedingungen:	Eine Aufgabe wurde geöffnet.
Nachbedingungen:	Dem Benutzer wird die nächste Anweisung markiert, welche er verschieben muss.
Alternative Abläufe:	Der Algorithmus ist korrekt: In diesem Fall wird keine Anweisung markiert.

Use-Case – Hinweis anzeigen lassen

7.3.1.16 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:	-

Use-Case – Musterlösung zeigen

7.3.1.17 Algorithmus ausdrucken

Name:	Algorithmus ausdrucken
Zweck:	Dem Benutzer soll es möglich sein den von ihm erstellten Algorithmus auszudrucken.
Beschreibung:	Der von Benutzer erstellte Algorithmus und weitere Informationen werden in Druckformat gebracht und an den ausgewählten Drucker geschickt.
Vorbedingungen:	Eine Aufgabe wurde geöffnet.
Nachbedingungen:	Eine Aufgabe wurde 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.

Use-Case – Algorithmus ausdrucken

7.3.2 Example-Builder

Hier finden sie nähere Informationen zu den USE-CASES des Example-Builders

7.3.2.1 Anweisung erstellen

Name:	Anweisung erstellen
Zweck:	Der Ersteller erstellt 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 erstellen, 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:	Eine Anweisung wurde zur Musterlösung hinzugefügt.
Alternative Abläufe:	Abbruch durch den Benutzer.

Use-Case – Anweisung erstellen

7.3.2.2 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 Anweisungsliste vorhanden.
Nachbedingungen:	Es wurden Änderungen in der Anweisung vorgenommen.
Alternative Abläufe:	Es wurden keine Änderungen in der Anweisung vorgenommen.

Use-Case – Anweisung bearbeiten

7.3.2.3 Anweisung löschen

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

Use-Case – Anweisung löschen

7.3.2.4 Aufgabenbeschreibung erstellen

Name:	Aufgabenbeschreibung erstellen
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 die Aufgabenbeschreibung nicht abgespeichert.

Use-Case – Aufgabenbeschreibung erstellen

7.3.2.5 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 drei verschiedene Formatierungsarten: 1. Fett 2. Kursiv 3. Unterstrichen 4. Aufzählung 5. Link
Vorbedingungen:	Aufgabenbeschreibung wird erstellt.
Nachbedingungen:	Es wurden Formatierungen in die Aufgabenbeschreibung eingefügt.
Alternative Abläufe:	Der Ersteller nimmt keine Formatierungen vor.

Use-Case – Formatieren der Aufgabenbeschreibung

7.3.2.6 Aufgabe öffnen

Name:	Aufgabe öffnen
Zweck:	Der Ersteller möchte Änderungen an einer bereits erstellten Aufgabe vornehmen.
Beschreibung:	Der Ersteller öffnet entweder eine lokal bzw. auf einem Server liegende 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.

Use-Case – Aufgabe öffnen

7.3.2.7 Hamster Aufgabendatei erstellen

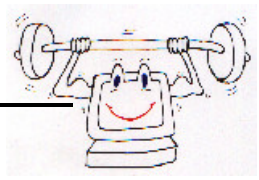
Name:	Hamster Aufgabendatei erstellen
Zweck:	Der Ersteller möchte eine neue Hamster-Aufgabenstellung entwerfen.
Beschreibung:	Der Ersteller legt eine neue Hamster-Aufgabe an.
Vorbedingungen:	Der Example-Builder wurde gestartet.
Nachbedingungen:	Der Ersteller kann nun eine Hamster-Aufgabe entwerfen.
Alternative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

Use-Case – Hamster Aufgabendatei erstellen

7.3.2.8 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 Aufgabenstellung. Deshalb gibt es zwei Alternativen: 1. Soll die geladene Aufgabe, bevor sie ersetzt wird, gespeichert werden? 2. Soll die geladene Aufgabe, ohne sie zu speichern, ersetzt werden?

Use-Case – E/A Aufgabendatei erstellen



7.3.2.9 Aufgabe speichern

Name:	Aufgabe speichern
Zweck:	Es gibt zwei Möglichkeiten: 1. Der Benutzer möchte entweder die Änderungen an einer geöffneten Aufgabenstellung speichern. 2. Ist die Aufgabendatei noch nicht angelegt worden, so muss diese erstellt.
Beschreibung:	Der Benutzer speichert die Aufgabenstellung. Der Benutzer hat die Wahl zwischen zwei verschiedenen Speicherorten: 1. Die Aufgabendatei soll auf dem lokalen Datenträger abgelegt werden. 2. Die Aufgabendatei soll auf einem zentral liegenden Server abgelegt 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:	Wenn beim Anlegen einer Datei bereits eine Datei mit dem selben Dateinamen vorhanden ist gibt es zwei alternative Abläufe: 1. Soll die vorhandene Datei überschrieben werden? 2. Soll ein anderer Dateiname angegeben werden?

Use-Case – Aufgabe speichern

7.3.2.10 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 ersetzt wird, gespeichert werden? 2. Soll die geladene Aufgabe, ohne sie zu speichern, ersetzt werden?

Use-Case – Template öffnen

7.3.2.11 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 Anfangsinitialisierungen für die Variablen nötig so öffnen sich Pop-up-Fenster, welche vom Ersteller die gewünschte Anfangsinitialisierung fordern.
Vorbedingungen:	Der Ersteller hat eine Musterlösung entwickelt.
Nachbedingungen:	Die Musterlösung wurde getestet.
Alternative Abläufe:	Der Musterlösung ist fehlerhaft. In diesem Fall muss der Ersteller, bevor er sein Programm testen kann, die Fehler korrigieren.

Use-Case – Erstellte Aufgabe testen

7.3.2.12 Ergebnisse überprüfen

Name:	Ergebnisse überprüfen
Zweck:	Zur Überprüfung der Endwerte der anfangsinitialisierten Variablen.
Beschreibung:	Für diesen Punkt öffnet sich ein eigenes Fenster, in welchem der Benutzer, Aus- und Eingabe-Werte für sein Programm definieren kann. Zusätzlich kann er End-Werte für Variablen definieren. Anhand von diesen Daten kann eine mögliche Korrektheit des Programm garantiert werden.
Vorbedingungen:	Eine Musterlösung wurde entwickelt.
Nachbedingungen:	Es werden die Endwerte der in die Überwachung eingetragenen Variablen angezeigt.
Alternative Abläufe:	Die Musterlösung ist fehlerhaft. In diesem Fall muss der Ersteller zuerst seine Fehler korrigieren.

Use-Case – Ergebnisse überprüfen

7.3.2.13 Zufallsanweisungen erzeugen

Name:	Erstellte Aufgabe testen
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:	-

Use-Case – Zufallsanweisungen erzeugen

7.3.2.14 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:	Es steht mindestens eine Anweisung in der Anweisungsliste.
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.

Use-Case – Pseudo-Code erzeugen

7.3.2.15 Verfügbarkeit von Hinweisen festlegen

Name:	Verfügbarkeit von Hinweisen festlegen.
Zweck:	Der Ersteller soll kontrollieren können, ob der Benutzer vom ProgrammierTrainer 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.

Use-Case – Verfügbarkeit von Hinweisen festlegen

7.3.2.16 Verfügbarkeit einer Musterlösung festlegen

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

Use-Case – Verfügbarkeit einer Musterlösung festlegen

7.3.2.17 Verzeichnis indizieren

Name:	Verzeichnis indizieren
Zweck:	Der Ersteller indiziert das Serververzeichnis.
Beschreibung:	Der Ersteller soll die Möglichkeit haben das auf dem Server liegende Aufgabenverzeichnis zu indizieren. Dies wird notwendig, wenn auf dem Server eine neue Datei angelegt bzw. eine bestehende Datei gelöscht wird. Beim Indizieren wird die auf dem Server liegende Indexdatei aktualisiert.
Vorbedingungen:	Der Example-Builder wurde gestartet.
Nachbedingungen:	Das Server-Verzeichnis wurde indiziert.
Alternative Abläufe:	Der Server ist nicht erreichbar: Das Serververzeichnis wurde nicht indiziert.

Use-Case – Verzeichnis indizieren

8 Qualitäts-Zielbestimmungen

Rasche Antwortzeiten in allen Interfaces. Auch wenn die Daten erst vom Server geholt werden, sollten diese in 90% aller Fälle innerhalb von 2 bis maximal 5 Sekunden verfügbar sein. Wir gehen hier von 90% aus, weil es verschiedene Einflussfaktoren gibt, wie z.B. Netzwerkauslastung, die wir nicht beeinflussen können.
Der Datenumfang ist variabel und auch irrelevant, da die Übungsaufgaben in einzelnen Dateien gespeichert werden.

8.1 Wichtige Qualitätsmerkmale:

- Geschwindigkeit
- Zuverlässigkeit
- Portabilität in andere Systeme
- Benutzerfreundlichkeit
- Datensicherheit

9 Entwicklungsumgebung

Für die Entwicklung des Programmiertrainers wird folgende Software eingesetzt:

Programmierumgebung:	Borland JBuilder 9.0 Professional
Projektplan:	Microsoft Project 98
UML-Tool:	Microsoft Visio 2000
Sun JDK:	Version 1.4

10 Systemweiterentwicklung

Im Weiteren soll das System so implementiert werden, dass es auch auf die Syntax einer anderen Programmiersprache umgestellt werden kann. Dazu ist es nötig, die im Python entfallenden End-Marken der einzelnen Blöcke mittels Kommentaren festzuhalten.