Inform_DE: Die deutsche Inform-Library

Version: Freitag, 26. Juli 2002

Inhalt:

Inform_DE: Die deutsche Inform-Library 1
- Zur deutschen Library 2
  - Wie es dazu kam 2
  - Versionen 2
  - Hinweise, Vorschläge, Diskussionen 2
  - Rechtliches 2
- Dokumentation der deutschen Library 3
  - Aufbau einer deutschen Inform-Datei 3
  - GLULX-Kompatibilität 3
  - Parsing und LanguageToInformese 3
  - Deutsche Umlaute 4
  - Deklinationen für Objekte 4
  - Deklinationstabellen 5
  - Die „the“ und „a“ im Deutschen 6
  - Eigennamen 7
  - Objektspezifische indefinite Artikel 7
  - Funktionen für diverse Kurzformen 7
  - Grammatik 8
  - Tokens für die Grammatik 8
  - Neue und alte Rechschreibung und andere Konstanten 9
- LanguageToInformese: Anpassungsmöglichkeiten 9

Bekannte Fehler 10
- Grammatikalisch falsche Eingaben sind erlaubt 10
- Ungewolltes Abschneiden von Endungen 10
- Limitierung auf acht Buchstaben im Dictionary 11
- Referenzen auf Objekte mit Synonymen verschiedenen Geschlechts 11

ToDos und offene Punkte 13
- LanguageToInformese aufteilen 13
- Kleinere ToDos 13
- Bekannte Fehler der Library nach Bugfixes 13

Zur deutschen Library

Wie es dazu kam

Inform von Graham Nelson ist die am weitesten verbreitete Programmiersprache für interaktive Fiktion auf der Welt. Die Inform Library ist die dazugehörige Programmbibliothek, welche die grundlegende Funktionalität die Entwicklung von Textadventures zur Verfügung stellt. 1996 übersetzte Tinic Urou diese englische Bibliothek ins Deutsche. Obwohl diese Übersetzung nur ein Versuch mit vielen Fehlern blieb, so bildete Tinics Version die Grundlage für zwei weitere Versionen: Toni Arnold und Ralf Herrmann veröffentlichten unabhängig voneinander jeweils eine deutsche Library. Beide waren ausgereift und voll funktionstüchtig. Leider gab es große Unterschiede zwischen beiden. Stärken und Schwächen waren in etwa gleich verteilt.

1999 begann ich mein Textadventure „Starrider“, welches schließlich das erste längere Abenteuer (wenn man von der „Adventure“-Übersetzung von Toni Arnold absieht) in deutscher Sprache werden sollte, welches auf Inform basierte. Ich benutzte Tonis Version, korrigierte und erweiterte sie jedoch an einigen Stellen. Später übernahm ich außerdem Teile aus Ralfs Bibliothek. Somit entstand eine „eigene“ Bibliothek, die Elemente sowohl von Ralf als auch Toni enthielt.

Diese Bibliothek wurde zum ersten mal 2001 im „Deutschen Jump-Start-Kit“ veröffentlicht, einem Versuch, deutsches Inform Autoren leicher zugänglich zu machen. Auf Betreiben der deutschen IF-Community wurde diese Version der Library noch verbessert und zur „offiziellen deutschen Version“ gekürt. Seitdem haben viele an dieser Programmbibliothek mitgewirkt, Ideen, Verbesserungen und Vorschläge mit eingebracht, um Inform auf Deutsch zu einem möglichst ausgereiften System für Textadventures werden zu lassen.

Die offizielle Seite der Bibliothek ist http://www.textfire.de/archiv/

Viel Spaß mit der „offiziellen deutsche Version“,
Max Kalus.

Versionen

Es gibt zwei Arten von Versionen der Bibliothek: Eine stabile Version kann von allen verwendet werden, wärend eine Testversion für diejenigen gedacht ist, welche die letzten Bugfixes und Veränderungen testen wollen.

Versionsnummern sind fortlaufend und beginnen mit 01. Testversionen werden nach jeder veröffentlichten Änderung um eins hochgezählt. Stabile Versionen erhalten die Versionsnummer überarbeiteten Testversion.

Hinweise, Vorschläge, Diskussionen

Fehler und Hinweise können an archiv@textfire.de geschickt werden.

Bei Vorschlägen, die Wording oder Erweiterungen der Library betreffen, bietet es sich an, diese erst in der if-de-Mailingliste zu besprechen. Mehr Informationen zur Mailingliste unter folgender URL:

http://www.if-de.de/liste.html

Rechtliches

Die deutsche Inform Library unterliegt den selben Bestimmungen und rechtlichen Grundlagen wie die englische Library. Es wird keinerlei Haftung bei Schaden übernommen, sprich: Benutzung auf eigene Gefahr.

Dokumentation der deutschen Library

Da die deutsche Library auf der englischen basiert, können alle Regeln für die englische Library angewendet werden. Folgende Ergänzungen und Ausnahmen gelten jedoch (Teile der Dokumentation habe ich Tonis entnommen):

Aufbau einer deutschen Inform-Datei

Im Großen und Ganzen ist der Aufbau der deutschen Inform-Datei mit der Englischen identisch, bis auf einige kleine Ausnahmen:

- Statt `Include "Grammar";` muss `Include "GermanG";` eingebunden werden.
- Am Anfang sollte man festlegen, ob man alte oder neue Rechtschreibung verwenden will (s.u.).

Mehr Informationen (eine Beispiel-Datei) findet man im Inform-Jump-Start-Kit auf dieser Seite:

http://www.beimax.de/if/jumpstart.html

GLULX-Kompatibilität

Die deutsche Library ist GLULX-Kompatibel! Folgendes ist zu beachten: Wer ein GLK-Datei erstellen will, muss zusätzlich zur deutschen Library den Header `infglk.h` einbinden, sonst kommt es zu Fehlern. Diese Header-Datei befindet sich in der GLULX-Library, welche unter dieser Adresse zu finden ist:

ftp://ftp.ifarchive.org/if-archive/programming/glulx/compilers/inform/library/infglk-0.6.1.zip

Wer eine normale Inform-Datei erstellen will, sollte den Biplatform-Compiler verwenden. Falls der normale Standardcompiler verwendet wird, müssen folgende Zeilen am Anfang des Codes eingegeben werden, um Fehlermeldungen zu vermeiden:

```
!ZCode-Unterstützung
#ifndef TARGET_GLULX;
Constant TARGET_ZCODE;
Constant WORDSIZE 2;
#endif;
```

Diese Zeilen dürfen beim Biplatform-Compiler nicht eingegeben werden!

Parsing und LanguageToInformese

Inform bietet für die Spracheinstellungen einen Entry-Point LanguageToInformese an. Die deutsche Library verwendet diese Routine (in german.h), um einfache Änderungen vor dem Parsen vorzunehmen:

- Falls NO_PUNCTUATION definiert ist, werden die Satzzeichen gelöscht (s.u.).
- Bei Verben wird ein mögliches End-„e“ gelöscht.
- Umlaute werden in ihre Kreuzworträtsel-Äquivalente umgewandelt.
- Einige Kurzformen (im, am, zum, zur, ins, ans, vom) werden ausgeschrieben (z.B. „im“ in „in dem“).
- Deutsche Deklinationsendungen werden ggf. gelöscht (z.B. „Geistes“ in „Geist“)
- **Weiter unten kommen noch weitere Informationen zur Routine für fortgeschrittenere Problemstellungen**

Diese Änderungen sind in aller Regel sehr hilfreich, da man sich z.B. nicht um alle Deklinationsformen eines Wortes kümmern muss. In einigen Fällen kommt es jedoch zu Problemen, welche weiter unten z.T. erklärt werden. Einige Punkte, auf die man achten sollte, wenn man die Library verwendet:

- Alle neue Verben müssen ohne End-„e“ geschrieben werden. Ein Verb wie „jauchze“ wird als „jauchz“ festgelegt, bei der Eingabe wird dann nämlich sowohl „jauchze“ als auch „jauchz“ verstanden. Einige Seltsamkeiten ergeben sich hier und da daraus, z.B. dass „nimme“ als „nimm“ interpretiert wird. Bei 90% aller Verben ist das jedoch korrekt.
- Name-Properties immer ohne Umlaute schreiben (s.u.)!
- Es kommt in seltenen Fällen zu einem Fehler im Wortstamm, z.B. „Bau“ und „Bauer“. Darauf wird ebenfalls weiter unten eingegangen.

Deutsche Umlaute

Inzwischen beherrscht der Compiler problemlos Umlaute und Sonderzeichen. Deutsche Umlaute können also bei der Eingabe von Namen, von Text, u.ä. ohne weiteres Eingegeben werden, wobei nicht alle Interpreter diese Zeichen korrekt darstellen können. Frotz z.B. beherrscht sowohl die Ein- als auch die Ausgabe von Umlauten.

Beispiel:
```
print "Der Löwe brüllt ängstlich.";
```

Es ist jedoch auch grundsätzlich korrekt, die ZSCII-Äquivalente für umlaute zu verwenden, wie im Designer's Manual angegeben. Wichtig hierbei ist, dass GLULX bis dato diese ZSCII-Umschreibungen jedoch nicht versteht.
Umlaute haben folgende Umschreibungen:

ä	@:a	Ä	@:A
ö	@:o	Ö	@:O
ü	@:u	Ü	@:U
ß	@ss		

Eine Ausnahme gibt es jedoch: Die Wörter im Dictionary von Inform dürfen keine Umlaute besitzen, da diese sonst nicht erkannt werden.

Beispiel:
```
Object -> Loewe
      with short_name "Löwe",
            name "loewe",
      has male;
```

Hier wird „loewe" im `name`-Array angegeben. Bei der Eingabe werden Umlaute automatisch in ihre Entsprechungen umgewandelt:

ä	ae
ö	oe
ü	ue
ß	ss

Deklinationen für Objekte

Die deutsche Sprache besitzt im Gegensatz zur Englischen Deklinationen. Zusätzlich werden Adjektive anders dekliniert als Substantive, so dass mehrere zusätzliche Properties nötig sind, um deutsche Objekte grammatikalisch korrekt ins Spiel einzubinden.

Im `short_name` sollte nur der Name des Objekts ohne Adjektive oder nachgeordnete Wortgruppen angegeben werden. Die Property `adj` kümmert sich um die Adjektive. Hier kann ein Array von Adjektiven für das Objekt angegeben werden. Ebenso die Property `post`, welche nachgeordnete Wortgruppen aufnehmen kann.

Wichtig ist auch, dass jedes Objekt im deutsche ein geschlechtsspezifisches Attribut bekommen muss!

Schließlich muss noch in der Poperty `dekl` angegeben werden, welche Deklination das Objekt besitzt (s.u.).

Beispiel:
```
Object -> "Buch"
      with dekl 4, adj "dick" "grün", post "des Zauberers",
            name "buch" "dick" "gruen" "zauberbuch",
      has neuter;
```

In diesem Beispiel wird angegeben, dass das Objekt Neutrum ist und nach der 4. Deklinationstabelle (s.u.) dekliniert werden soll. Das Buch wird im Spiel als „das dicke grüne Buch des Zauberers" bezeichnet. Wichtig hier ist, dass auch Adjektive in der name-Property aufgeführt werden sollten und dass Adjektive und Substantive in ihrer Grundform geschrieben werden müssen.

Deklinationstabellen

Hier die Deklinationen und die Nummern, welche in der `dekl`-Property angegeben werden:

Deklinationstyp 0

Nulldeklinationsklasse, d.h. das Nomen wird nicht dekliniert. Adjektive werden ganz normal dekliniert.

Deklinationstyp 1

		Maskulinum	Neutrum
Sing	Nom	der Tag	das Jahr
	Akk	den Tag	das Jahr
	Dat	dem Tag[-e]	dem Jahr[-e]
	Gen	des Tag-[e]s	des Jahr-[e]s
Plur	Nom	die Tag-e	die Jahr-e
	Akk	die Tag-e	die Jahr-e
	Dat	den Tag-en	den Jahr-en
	Gen	der Tag-e	der Jahr-e

Deklinationstyp 2

		Maskulinum	Neutrum
Sing	Nom	der Apfel	das Segel
	Akk	den Apfel	das Segel
	Dat	dem Apfel	dem Segel
	Gen	des Apfel-s	des Segel-s
Plur	Nom	die Äpfel	die Segel
	Akk	die Äpfel	die Segel
	Dat	den Äpfel-n	den Segel-n
	Gen	der Äpfel	der Segel

Deklinationstyp 3

		Maskulinum	Neutrum
Sing	Nom	der Staat	das Auge
	Akk	den Staat	das Auge
	Dat	dem Staat[-e]	dem Auge
	Gen	des Staat-[e]s	des Auge-s
Plur	Nom	die Staat-en	die Auge-n
	Akk	die Staate-en	die Auge-n
	Dat	den Staat-en	den Auge-n
	Gen	der Staat-en	die Auge-n

Deklinationstyp 4

		Maskulinum	Neutrum
Sing	Nom	der Wald	das Bild
	Akk	den Wald	das Bild
	Dat	dem Wald[-e]	dem Bild[-e]
	Gen	des Wald-[e]s	des Bild-[e]s
Plur	Nom	die Wäld-er	die Bild-er
	Akk	die Wäld-er	die Bild-er
	Dat	den Wäld-ern	den Bild-ern
	Gen	der Wäld-er	der Bild-er

Deklinationstyp 5

		Maskulinum	Neutrum
Sing	Nom	der Opa	das Deck
	Akk	den Opa	das Deck
	Dat	dem Opa	dem Deck
	Gen	des Opa-s	des Deck-s
Plur	Nom	die Opa-s	die Deck-s
	Akk	die Opa-s	die Deck-s
	Dat	den Opa-s	den Deck-s
	Gen	der Opa-s	der Deck-s

Deklinationstyp 6

		Maskulinum
Sing	Nom	der Mensch
	Akk	den Mensch-en
	Dat	dem Mensch-en
	Gen	des Mensch-en
Plur	Nom	die Mensch-en
	Akk	die Mensch-en
	Dat	den Mensch-en
	Gen	der Mensch-en

Deklinationstyp 7

		Femininum
Sing	Nom	die Kraft
	Akk	die Kraft
	Dat	der Kraft
	Gen	der Kraft
Plur	Nom	die Kräft-e
	Akk	die Kräft-e
	Dat	den Kräft-en
	Gen	der Kräft-e

Deklinationstyp 8

		Femininum
Sing	Nom	die Mutter
	Akk	die Mutter
	Dat	der Mutter
	Gen	der Mutter
Plur	Nom	die Mütter
	Akk	die Mütter
	Dat	den Mütter-n
	Gen	der Mütter

Deklinationstyp 9

		Femininum
Sing	Nom	die Frau
	Akk	die Frau
	Dat	der Frau
	Gen	der Frau
Plur	Nom	die Frau-en
	Akk	die Frau-en
	Dat	den Frau-en
	Gen	der Frau-en

Deklinationstyp 10

		Femininum
Sing	Nom	die Oma
	Akk	die Oma
	Dat	der Oma
	Gen	der Oma
Plur	Nom	die Oma-s
	Akk	die Oma-s
	Dat	den Oma-s
	Gen	der Oma-s

Die „the“ und „a“ im Deutschen

Im Englischen übernehmen zwei Funktionen, „the“ und „a“ viele der automatischen Antworten. Durch die Deklinationen im Deutschen gibt es hier mehr Funktionen:

Definit:

Nom	Akk	Dat	Gen
(der)	(den)	(dem)	(des)
(GDer)	(GDen)	(GDem)	(GDes)

Indefinit:

Nom	Akk	Dat	Gen
(ein)	(einen)	(einem)	(eines)
(GEin)	(GEinen)	(GEinem)	(GEines)

"Kein...":

Nom	Akk	Dat	Gen
(kein)	(keinen)	(keinem)	(keines)
(GKein)	(GKeinen)	(GKeinem)	(GKeines)

Kein Artikel (nach Präpositionen):

Nom	Akk	Dat	Gen
(_er)	(_en)	(_em)	(_es)

Die Artikel werden jeweils nur in der maskulinen Form angegeben da diese eindeutig ist. Fehlende Artikel nach Präpositionen sollen über die `(_e*)`-Routinen explizite als fehlend gesetzt werden, sonst werden das Nomen und die Adkjektive nicht dekliniert.

Beispiel zur Anwendung:

```
print "Du gibst ", (dem) noun, " ", (den) second, ".";
```

Pronomen werden ähnlich definiert wie die Artikel:

Pronomen:

Nom	Akk	Dat	Gen
(er)	(ihn)	(ihm)	(seiner)
(GEr)	(GIhn)	(GIhm)	(GSeiner)

Demonstrativ:

Nom	Akk	Dat	Gen
(dieser)	(diesen)	(diesem)	(dieses)
(GDieser)	(GDiesen)	(GDiesem)	(GDieses)

Eigennamen

Eigennamen können in Inform mit der property `proper` als solche deklariert werden. Dies hat zur Folge, dass der Artikel weggelassen wird. Sollte eigentlich ein indefiniter Artikel gedruckt werden, wird dieser durch den Definiten ersetzt.

Beispiel:

```
Du siehst hier einen Zauberer.
```
aber mit Eigennamen:
```
Du siehst hier Zauberer Runzelfus.
```
aber mit Adjektiven:
```
Du siehst hier den netten Zauberer Runzelfus.
```

Objektspezifische indefinite Artikel

In Inform können fixe indefinite „Artikel“ mit der Property `article` gesetzt werden. Dies wird benötigt einerseits für nicht zählbare Objekte im Singular ohne oder mit einem speziellen Artikel und für bestimmte Objekte, die nie mit indefinitem Artikel vorkommen. (Bei `pluralname`-Objekten, die immer im Plural stehen, wird der indefinite Artikel automatisch weggelassen). Wird mit `article` ein String deklariert, wird dieser zwar gedruckt, aber nicht dekliniert. Gleiches gilt für den Fall einer Routine. Die `definit`-Konstante ist speziell definiert, um definite Artikel zu erzwingen. Hat das (in diesem Fall "widersprüchliche" Objekt) zusätzlich das `proper`-Attribut, wird `article` wie in der englischen Version ignoriert.

Drei Beispiele:

```
Du siehst hier Blut.
```
mit `article ""`

```
Du siehst hier viel böses Blut.
```
mit `article "viel"` und `adj "b@:os"`

```
Du siehst hier die andere Seite des H@:ugels.
```
mit `article definit` - im Unterschied zur englischen Version eine spezielle Konstante

Die `print (name) obj;`-Konstruktion für „kein Artikel drucken in jedem Fall“ wird von der deutschen Übersetzung nicht unterstützt, ist doch die Adjektivdeklination auch dann unklar, wenn vier Konstrukte für vier Kasus definiert würden.

Funktionen für diverse Kurzformen

Die deutsche Library enthält eine Reihe von Funktionen, um das Leben des Programmierers zu vereinfachen. Im folgenden sind diese aufgeführt.

GEristSiesind

Erwartet einen Parameter - gibt je nach Genus und Anzahl der Objekte „Er/Sie/Es ist“ oder „Sie sind“ aus.

Beispiel:

```
print GEristSiesind (x1), " zu beschäftigt, um dich wahrzunehmen.";
```

ist

Alias zu isorare - gibt „ist“ oder „sind“ aus, je nach Anzahl der Objekte.

Beispiel:

```
print (Gder) obj, " ", (ist) obj, " zu beschäftigt, um dich wahrzunehmen.";
```

hat

Gibt „hat“ oder „haben“ aus.

wird

Gibt „wird“ oder „werden“ aus.

endT

Kann als Endung für Verben verwendet werden. Gibt „t“ bei Singular aus oder „en“ bei Plural.

Beispiel:

```
print (Gder) obj, " spring", (endT) obj, " über die Brücke.";
```

endEt

Kann als Endung für Verben verwendet werden. Gibt „et“ bei Singular aus oder „en“ bei Plural.

Beispiel:

```
print (Gder) obj, " schneid", (endEt) obj, " die Karotten.";
```

singplur

Gibt Singular und Plural bei unregelmäßigen Verben aus.

Beispiel:

```
print (Gder) obj, " ";
singplur(obj, "spricht", "sprechen");
print " über deinen Auftrag.";
```

Grammatik

Aufbau und Struktur der Grammatik unterscheiden sich natürlich vom Englischen. Die Verbliste ist demnach völlig anders aufgebaut. Debug-Verben sind jedoch gleich geblieben!

Um einen Einblick in die Verben zu bekommen, bitte die Header-Datei `germang.h` zu Gemüte führen!

Tokens für die Grammatik

Auch für die Grammatik gibt es einfache Funktionen, welche spezielle deutsche Ausdrücke parsen helfen.

<hinein>

Überliest „hinein“ und „rein“.

Beispiel:

```
Verb 'geh' * hinein     -> GoIn;
```

<heraus>

Überliest „heraus“ und „herunter“ und „raus“.

Beispiel:

```
Verb 'geh' * heraus     -> Exit;
```

<xhinweg>

Bildet „hinweg“, „hinfort“, „weg“ und „fort“ auf qwe (also das Objekt, das von diesem Pronomen bezeichnet wird) ab.

Beispiel:

```
Verb 'wirf' * multiheld xhinweg     -> Drop;
```

<xhinein>

Bildet „hinein“ und „rein“ auf qwe (also das Objekt, das von diesem Pronomen bezeichnet wird) ab.

Beispiel:

```
Verb 'steck' * multiexcept xhinein  -> Insert;
```

<xheraus>

Bildet „heraus“ und „herunter“ und „raus“ auf qwe (also das Objekt, das von diesem Pronomen bezeichnet wird) ab.

Beispiel:

```
Verb 'nimm' * multiinside xheraus   -> Remove;
```

<xdamit>

Bildet „damit“ auf qwe (also das Objekt, das von diesem Pronomen bezeichnet wird) ab.

Beispiel:

```
Verb 'fuetter' * creature xdamit    -> Give;
```

Neue und alte Rechschreibung und andere Konstanten

Die Library versteht sowohl alte als auch neue Rechtschreibung. Voreinstellung ist (aus Kompatibilitätsgründen) die alte Rechtschreibung, welche mit der Konstante R_ALT gesetzt wird. Falls man die neue Rechtschreibung benutzen will, setzt man am Anfang seines Codes folgende Zeile:

```
Constant R_NEU;
```

Damit wird die neue Rechtschreibung aktiviert.
In Zusammenhang damit steht die Konstante C_DASS, welche den String „daß“, bzw „dass“ enthält. Diese kann im Code verwendet werden, um die richtige Form zu verwenden.

Schließlich gibt es eine weitere Konstante, welche für den Entwickler interessant sein dürfte: `NO_PUNCTUATION` kann definiert werden, um den Parser anzuweisen, jegliche Satzzeichen zu ignorieren.

LanguageToInformese: Anpassungsmöglichkeiten

Wie oben schon erwähnt, ist die LanguageToInformese-Routine für eine Anpassung deutscher Eingaben zuständig. Seit Version 13 von Inform_de ist die Routine in mehrere Unterroutinen aufgeteilt worden, um eine Anpassung/Erweiterung zu vereinfachen:

```
[ LanguageToInformese;
      PreProcessGerman();
      ProcessGermanVerbs();
      ProcessGermanUmlauts();
      ProcessGermanShorties();
      ProcessGermanSuffixes();
#ifdef DEBUG;
      ShowGermanDebug();
#endif;
];
```

Am einfachsten ist demnach die Anpassung der Routine selbst, da sie an sich recht kurz ist (per `Replace LanguageToInformese` geht das ja einfach). Was machen die einzelnen Routinen? Hier eine kurze Erläuterung:

- **PreProcessGerman:** Macht die Drecksarbeit vor dem Rest. Hier steht nicht viel drin und die Routine eignet sich sehr gut, um eigene Wortregeln abzufangen.
- **ProcessGermanVerbs:** Verben werden um das ‚e' gekürzt, auch bei Befehlen (drache, speie feuer) muss dies funktionieren!
- **ProcessGermanUmlauts:** Relativ klar: Hier werden schlicht Umlaute in Kreuzworträtsel-Umlaute geändert.
- **ProcessGermanShorties:** Einige Kurzformen (im, am, zum, zur, ins, ans, vom) werden ausgeschrieben (z.B. „im" in „in dem").
- **ProcessGermanSuffixes:** Der kniffligste Teil: Die deutschen Endungen werden weggekürzt, damit der Parser die Grundform später auch versteht.
- **ShowGermanDebug:** Im Debug-Modus wird im Trace-Modus das Ergebnis von LanguageToInformese ausgegeben - praktisch, wenn man Fehler jagen will.

Bekannte Fehler

Inform ist zwar sehr mächtig, aber nicht perfekt. Einige Probleme und Fehler bestehen in der deutschen Library und manche wirken sich negativ auf das Deutsche aus.

Grammatikalisch falsche Eingaben sind erlaubt

Auch die deutsche Library bleibt dem Motto treu: Wenn es irgendwie Sinn macht, dann versuche den Befehl zu verstehen. Andere deutsche Autorensysteme sind hier etwas pingeliger (z.B. „schau schwertlilie" wird nicht akzeptiert, sondern nur „schau schwertlilie an"). Da Inform nur die einzelnen Wörter im Dictionary abgleicht und durch die LanguageToInformese-Routine nur die Grundformen gelangen, kann der Benutzer jeden grammatikalischen Unsinn eingeben. Sätze wie „geh die tür" oder „nimm dem weins" werden ohne Kommentar akzeptiert. Um Andrew Plotkin zu zitieren: „So what?".

Ungewolltes Abschneiden von Endungen

In seltenen Fällen kann es vorkommen, dass Endungen abgeschnitten werden, die noch zum Wortstamm gehören. Dies ist dann der Fall, wenn diese Wörter nicht im Dictionary sind, das Abgeschnittene Wort jedoch schon (anderes Wort mit kürzerem Stamm). Beispiel:

```
Object -> "Bau"
      with dekl 3,
            name 'bau',
      has male static;

Object -> "Bauer"
      with dekl 6,
            name 'bauer',
      has male animate;
```

Beispielausgabe:

```
> betrachte bau
Du entdeckst an dem Bau nichts Spezielles.

> betrachte bauer
Du entdeckst an dem Bauern nichts Spezielles.

>betrachte bauern
Du entdeckst an dem Bau nichts Spezielles.
```

Die letzte Zeile erkennt die Eingabe falsch. Die einzige Möglichkeit in diesem Fall ist, ‚bauern', usw. direkt in die name-Property mitaufzunehmen, was freilich sehr unschön ist...

Limitierung auf acht Buchstaben im Dictionary

Das Deutsche hat viele lange Wörter. Die Beschränkung des Dictionary auf 8 Buchstaben pro Begriff erscheint deshalb häufig als enger Zwang. Glücklicherweise gibt es eine Möglichkeit, diese Beschränkung zu umgehen.

Beispiel:

In „Starrider“ gibt es einen Sicherungsschlüssel und einen Sicherungsschalter. Folgende Routine ermöglicht die Überprüfung langer Wörter. Sie macht sich einige Opcodes und die parse_name-Methode zunutze. Ähnliche Effekte kann man auch mit der string.h-Library erreichen, welche im if-Archiv zu finden ist:

```
Array aux -> 64;

[ istLangesWort wn s wl wa i;
    ! Diese Routine prüft, ob das Wort wn im Textbuffer mit
    ! dem String s übereinstimmt. Der String sollte weder
    ! Großbuchstaben noch Umlaute haben.
    ! Dank an Martin Oehm

    @output_stream 3 aux;
    print (string) s;
    @output_stream -3;
    wl = WordLength(wn);
    wa = WordAddress(wn);
    if (wl ~= aux-->0) return false;
    for (i=0: i<wl: i++, wa++)
        if (aux->(i+2) ~= wa->0) return false;
    return true;
];

[ IsAWordIn w obj prop k l m;
      ! Überprüft, ob das Wort w in der Name-Property von obj vorkommt
      k=obj.&prop; l=(obj.#prop)/2;
      for (m=0:m<l:m++)
      if (w==k-->m) rtrue;
      rfalse;
];

Object -> Hauptsicherung "Sicherungsschalter"
    with dekl 2, post "des Schiffs", article "der",
    name "hauptsicherung" "sicherung" "schalter" "notstrom",
      parse_name [n word;
            word = NextWordStopped();
            while (IsAWordIn(word,self,name))
            {

                  n++;
                  ! Sicherungsschalter muss eingegeben werden - "Sicherung"
                  ! zählt nicht...
                  if ((word == 'sicherung') &&
                        (istLangesWort(wn-1, "sicherungsschalter")
                              == false)) n--;
                  word = NextWordStopped();
            }
            return n;
    ],
    has male static;
```

Referenzen auf Objekte mit Synonymen verschiedenen Geschlechts

Folgendes Beispiel demonstriert das Problem:

```
Du siehst hier auch ein großes Tor.
```

```
> betrete tür

Es ist nicht offen.
```

Das Tor ist als Objekt als Neutrum definiert, weshalb die automatische Antwort ein „es“ referenziert. Da der Spieler jedoch ein Synonym verwendet hat, nämlich das weibliche Wort Tür, kommt es zu einer grammatikalisch falschen Antwort.

Es gibt leider wenig einfache Möglichkeiten, dies zu umgehen. In der Library wurde darauf geachtet, dass Antworten möglichst allgemein gehalten sind, z.B. statt „Er/Sie/Es ist nicht offen.“ kommt als Antwort „Die Tür ist nicht offen“.

Folgendes Beispiel zeigt eine Möglichkeit dieses Problem zu umgehen. Leider ist die Möglichkeit zu umständlich, um bei vielen Objekten genutzt zu werden.:

```
Object SimpleDoor "Tür"
with
    name 'tuer' 'tor',
    dekl 9,
    short_name [; if (self.short_name_index == 1) {
                              print "Tor"; rtrue;
                  } else {
                              print "Tür"; rtrue;
                  }
    ],
    short_name_index 1,
    description [; print (Gder) self, " erkennt, ob ", (er) self, " Tür
            oder Tor genannt wurde.";
    ],
    parse_name [n word;
            word = NextWordStopped();
            while (word ~= -1)
            {
                        if (word == 'tor')
                        {
                              give self ~female;
                              give self neuter;
                              self.short_name_index = 1;
                              n++;
                        }
                        else if (word == 'tuer')
                        {
                              give self ~neuter;
                              give self female;
                              self.short_name_index = 0;
                              n++;
                        }
                        else
                              return n;
                        word = NextWordStopped();
            }
            return n;
      ],
    found_in [;
            return (location == EastRoom)
                || (location ==  WestRoom);
    ],
    door_to [; if (self in EastRoom) return WestRoom; return EastRoom;
    ],
    door_dir [; if (self in EastRoom) return w_to; return e_to;
    ],
```

```
has static door openable ~open neuter;
```

ToDos und offene Punkte

LanguageToInformese aufteilen

Die Routine LanguageToInformese ist für die Bearbeitung der deutschen Grammatik verantwortlich und eine mächtige Funktion. Das macht sie leider auch sehr lang, so dass individuelle Anpassungen nur schwer möglich sind. Eine Aufteilung der Routine in mehrere Unterfunktionen ist eines der Anliegen, die noch zu erledigen sind. Entry Points sollen außerdem die Funktion flexibler machen.
Das ganze sollte außerdem zum Anlass genommen werden, die Funktion endlich zu dokumentieren.

Kleinere ToDos

- „nehme“ in „ich nehme“ oder „du nimmst“ umwandeln
- Articles-Property evt. beheben, damit ausdruck richtig gedruckt wird
- adj nicht nur als Array, sondern auch als Routine

und viele, viele andere...

Bekannte Fehler der Library nach Bugfixes

Auf der Inform-Patch-Seite finden sich aktuelle Patches zur Library. Die Patch-Seite findet man unter: http://www.inform-fiction.org/patches/library.html
Ich bemühe mich, die aktuellen Patches in der deutschen Bibliothek einzubauen. Manche sind jedoch nicht relevant oder nur schwer implementierbar.

Nicht behoben sind:

- Spaces inserted after apostrophes [trifft auf deutsche Lib nicht zu]
- Grammar property breaks in large games [wohl selten benötigt, der Biplattform-Code muss evt. angepasst werden]
- Colors incorrect after RESTORE/UNDO [nur in bestimmten Spielen relevant]
- Buggy message for boxes of scenery [selten benötigt]
- Infix ";" can take only one argument [wer’s braucht, kann einen Bugfix einreichen]
- Move count starts at 1 [Geschmackssache…]
- Inventory with a container [ändert Standardverhalten]
- "Can't go that way" message [Sollte dies geändert werden?]
- GoSub routine broken [Falsche Fehlermeldung?]
- Inform's standard screen effects don't work in V6 [Wer v6 braucht, kann einen Bugfix einreichen]
- short_name idiosyncracy [weniger ein Bug]
- DictionaryLookup returns wrong results [mein Fehler ist inzwischen fehlerhaft
- name property includes 'the' [kein Fehler]

Behoben wurden folgende Bugs:

- "IT" handled sub-optimally
- Buffers declared incorrectly
- Implementing a clock
- Plural containers listed wrong
- Strange spacing on inside_description
- Disambiguation/undo conflict
- Wrong message for GO [dir] [anything]
- Match list handled incorrectly
- Revealing contents in darkness
- Wrong messages for PUSH/TURN
- "Read" verb on out-of-scope objects [wobei hier die Beschreibung unklar war…]
- LEAVE tries too hard
- Object already in/on another
- Problem with additive 'describe' property