Regeln

Was andere Sprachen Methoden haben, hat Inform Regeln. (Auch wenn es traditionelle Methoden ebenfalls gibt, aber dazu später.) Diese Regeln regeln das Verhalten in der Modellwelt.

Wenn der Spieler etwas eintippt („take apple“), kümmern sich nach der Analyse durch den Parser ein Satz von Regeln darum

1. **ob die Aktion überhaupt möglich ist**
   (das übernehmen Check-Regeln, quasi der Controller der Welt) – vielleicht hat der Apfel ja das Attribut „fixed in place/not portable“, solche Gegenstände kann man nicht nehmen
2. **wie sich der Zustand der Modellwelt ändert** (wenn bei 1 nichts einzuwenden war)
   (das übernehmen Carry-Out-Regeln, quasi das Model der Welt) – der Apfel verschwindet vom bisherigen Ort und erscheint im Inventar des Spielers
3. **was dem Spieler als Text ausgegeben wird**
   (das übernehmen Report-Regeln, entspricht ein bisschem einer View der Welt)

Diese Standardregeln regeln das Standardverhalten, wenn man das ändern will, muss man sie ergänzen oder überschreiben. Dazu gibt es mehrere Möglichkeiten, wir wollen uns hier auf eine einzelne beschränken: Instead-Regeln („anstatt“).

```
Instead of eating a thing (called the target):
     if the target is edible:
          continue the action;
     otherwise:
          say "That's silly!"
```

Dieses Verhalten wird durch die Standardregeln bereits modelliert, aber doppelt hält besser. Ähnlich so:

```
Instead of taking a thing (called the target):
     if the target is portable:
          say "Quickly, you reach out...";
          continue the action;
     otherwise:
          say "That's silly!"
```

Wenn wir wollen, dass beim **Nehmen** eines Objekts unserer neuen Klasse Buch etwas anderes geschehen soll als standardmäßig, müssen wir das Standardverhalten überschreiben:

```
Instead of taking a book (called the target):
     if the target is portable:
          say "Ah, good literature!";
          continue the action;
     otherwise:
          say "That's a heavy book."
```

Wenn wir wollen, dass beim **Untersuchen** eines Objekts unserer neuen Klasse Buch etwas anderes geschehen soll als standardmäßig, müssen wir das Standardverhalten

überschreiben:

```
Instead of examining a book (called the reading material):
     say the description of the reading material;
     say " Its title is ";
     say the title of the reading material;
     say " It its about ";
     say the subject of the reading material;
     if the reading material is exciting:
          say ". It looks like a good read.";
     otherwise:
          say ". Probably boring."
```

Oder kürzer:

```
Instead of examining a book (called the reading material):
     say "[The description] Its title is [the title].  It its
about [the subject]";
     if the reading material is exciting:
          say ". It looks like a good read.";
     otherwise:
          say ". Probably boring."
```

Oder noch kürzer:

```
Instead of examining a book (called the reading material):
     say "[The description] Its title is [the title].  It its
about [the subject]. [if exciting]It looks like a good
read[otherwise]Probably boring[end if].";
```

Syntax der Regeln

- Sie beginnen immer mit `instead of`
- gefolgt von einer definierten Aktion – vorgegeben und für heute ausreichend sind: examining, taking, dropping, going to, going from, entering, opening, putting .. in , putting ... on, closing, eating und viele weitere. Man kann auch selber welche schreiben.
- gefolgt von der Klasse, auf die sich die Regel bezieht (aber: siehe unten)
- in Klammern gefolgt von einem temporären Bezeichner für das Objekt, auf das die Regel angewendet wird (temporäre Variable, lokaler Gültigkeitsbereich)
- gefolgt von einem Doppelpunkt und einer neuen Zeile.
- Jede Zeile endet entweder mit einem Strichpunkt oder – falls eine Kontrollstruktur eingesetzt wird – einem Doppelpunkt
- Die letzte Zeile und nur die endet mit einem Punkt.
- Nach einem Doppelpunkt wird eingerückt – das Ein- und Ausrücken entspricht den geschweiften Klammern in Java; Python benutzt dieselbe Syntax wie Inform, wo das Ein- und Ausrücken syntaktische Bedeutung hat.

Kontrollstrukturen:

```
if <Bedingung>:
     <Anweisung>;
otherwise if <Bedingung>:
     <Anweisung>;
otherwise:
     <Anweisung>;
```

Daneben gibt es auch noch while und Zählschleifen und Schleifen durch eine gegebene Menge von Objekten (wie bei Java iterable) und eine Switch-Entsprechung.

Ein häufiger Fehler: man vergisst Strichpunkt oder Punkt am Ende eienr Anweisung.

Bedingungen:

```
the player is carrying|wearing the <Objektbezeichner>
the <Objektbezeichner|player> is in the <Raumbezeichner>
```

Häufige Anweisungen (=Methoden oder Prozeduren):

```
say "..."; [System.out.println("...")]
let <Bezeichner> be <Wert|Datentyp> [lokale Variable]
now <Ding-Objektbezeichner> is in the location of the player;
move <Ding-Objektbezeichner> to the player;
```

Am besten lernt man an Beispielen, deshalb: Inform 4b Neue Regeln – Minispiel, und alle anderen Beispiele.

Ab hier brauchen nur noch besonders Neugierige weiter zu lesen

Amerkung

Sie sehen an den Beispielen vielleicht, dass Regeln sich nicht nur auf Klassen beziehen können („instead of examing a book“, sondern auch auf einzelne Objekte „instead of examining the diary“ oder Mengen von Objekten „instead of examining an exciting book).

Das ist der Vorteil und der Hintergrund dieser Regeln. In solchen Spielen verhalten sich – anders als beim reinen physikalischen Modellieren einer Welt – eben nicht alle Objekte einer Klasse gleich. Alle Instanzen der Klasse Buch sind gleich, aber nur eine davon hilft uns, ein Rätsel zu lösen.

In Java könnte man das so lösen, dass eben für jedes Objekt eine andere Unterklasse gebildet wird. Ungeschickt.

Oder man arbeitet mit dem Decorator-Pattern und ändert zur Laufzeit das Verhalten von Objekten. Dann bräuchte man sehr viele Decorator-Klassen.

Oder man baut in jede Klasse eine Methode mit booleschem Rückgabewert ein „istLösungVon(Rätsel)“. Auch nicht sehr elegant. Deshalb geht Inform den Weg über Regeln.

Damit kann man allerdings sprachlich knapp und doch verständlich Regeln aufstellen wie diese:

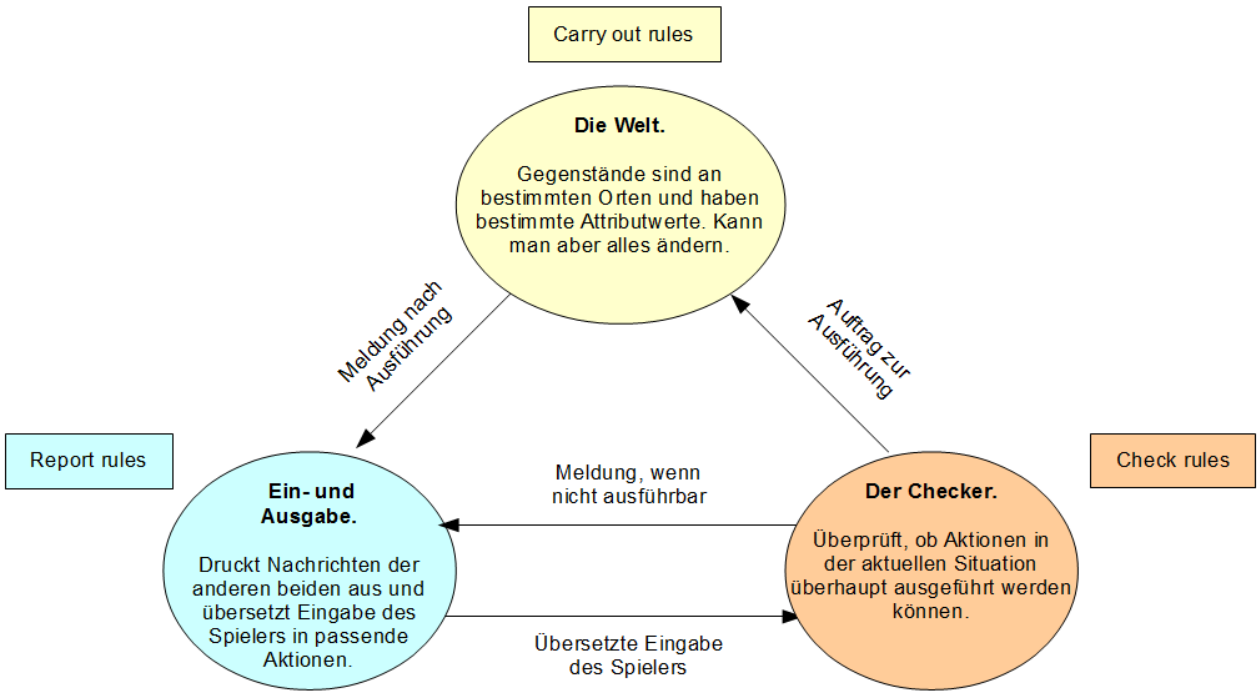
```
Instead of a suspicious person (called the suspect) burning
something which is evidence against the suspect when the
number of people in the location is at least two, try the
suspect going a random valid direction.
```

„Wenn eine verdächtige Person etwas verbrennen will, das als Beweis gegen ihn verwendet werden kann, wenn außer ihm mindestens noch jemand im Raum ist, dann soll der Verdächtige stattdessen in eine zufällig ausgewählte Richtung gehen, die irgendwohin führt.“

Für Neugierige

So sieht der Standard-Zyklus der Regeln aus: Nach der Analyse durch den Parser greifen Check-Regeln, die dann entweder an die Textausgabe rückmelden, dass die Aktion nicht ausgeführt werden kann, oder die Aktion weiterreichen an Carry-Out-Regeln.
Diese führen die Änderungen an der Modellwelt durch. Danach leiten sie die Verarbeitung an Report-Regeln weiter, die dann Text an die Ausgabe schicken.

Vor all diesen Regeln wird noch überprüft, ob der Gegenstand überhaupt existiert und sich im selben Raum befindet. Sonst kann man gar nichts damit machen.



Für Fortgeschrittene

Man kann an verschiedenen Stellen in diesen Verarbeitungszyklus eingreifen.

Eine selbst geschriebene After-Regel wird dann ausgeführt, wenn die Änderung an der Modellwelt abgeschlossen ist, aber *vor* der standardmäßigen Report-Regel und ersetzt die sozusagen.

(Wenn man möchte, dass die Report-Regel zusätzlich ausgeführt wird, beendet man die eigene Regel mit „Continue the action.“)

Eine selbst geschriebene Instead-Regel – das sind die wichtigsten – umgeht den ganzen Zyklus. Alle Modellwelt-Änderungen und Textausgaben an den Spieler nimmt man darin von Hand vor und ersetzt die Carry-out-Regeln sozusagen.

(Wenn man möchte, dass die Carry-Out-Regeln zusätzlich ausgeführt werden, beendet man die eigene Regel mit „Continue the action.“)