Neue Modder braucht das Land

Workshop – Einführung in LUA

Manuel Leithner aka Face
SFM-Modding
Kontakt: sfm_f@web.de
Homepage: http://forum.landwirtschafts-simulator.de

Gliederung

Gliederung

Gliederung

Tables

- Merkmale:
  - **einzige** Datenstruktur in LUA
  - vereint die Vorteile von Array, Sets, Listen, Queues und anderen Datenstrukturen
  - Indizierung über Zahlen, Strings
  - keine Größenbeschränkung
  - Tables sind **keine** Variablen, sondern **Objekte**
  - Erzeugung eines Tables über einen simplen Konstruktor-Aufruf

  - Tables sind ein sehr machtvolles Werkzeug um Daten zu strukturieren!
  - Repräsentieren auch Namespaces (Namensraum)

Tables

- Erzeugen von Tables und Zugriff auf die einzelnen Elemente

```
local myTable = {};                          -- Erzeugt einen leeren Table

local myTable2 = {1,45,15,10};               _=[[ Erzeugt einen Table mit 4 Einträgen
                                                print(myTable2[1])      -> 1
                                                print(myTable2[2])      -> 45
                                                print(myTable2[3])      -> 15
                                                print(myTable2[4])      -> 10
                                             ]]

local specialTable = {name=„Hans“, alter=15}; _=[[ Erzeugt einen Table mit 2 Einträgen und vorgegebener
                                                   Indizierung
                                                print(specialTable [1])                -> nil
                                                print(specialTable[„name“])            -> „Hans“
                                                print(specialTable[name])              -> nil
                                                print(specialTable.alter)              -> 15
                                             ]]
Häufiger Fehler:

local a ={};
local x = „y“;
a[x] = 10;

print(a[x]);      -> 10    (Wer von Feld „y“)
print(a.x);       -> nil   (Wert von Feld „x“ -> nicht definiert)
print(a.y);       -> 10    (Wert von Feld „y“)
```

CallByValue vs. CallByReference

- Tables sind Objekte!

CallByValue:

```
local a = 5;
local b = a;
b = 10;

print(a)          -> 5;
```

CallByReference:

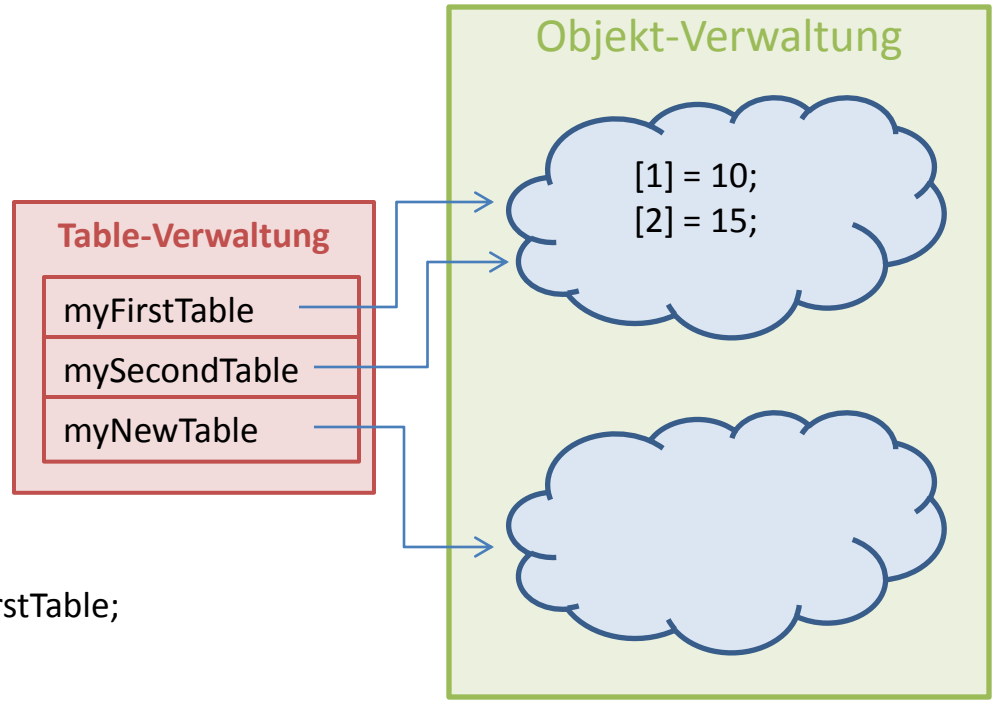
```
local myFirstTable = {};
myFirstTable[1] = 15;

local mySecondTable = myFirstTable;
mySecondTable[1] = 100;

print(myFirstTable[1])          -> 100;
```

CallByReference

```
local myFirstTable = {};
myFirstTable[1] = 10;
myFirstTable[2] = 15;
```




```
local mySecondTable = myFirstTable;
```

Es wird hier kein neuer Table angelegt!

Um komplett neue Table zu erzeugen:

```
local myNewTable = {};
```

CallByValue vs. CallByReference

- CallByReference nur bei Tables, sonst CallByValue
- Bei Table-Elementen ebenfalls CallByValue:

```
local myTable = {};
myTable[1] = 15;

local myVar = myTable[1];

myVar = 100;

print(myTable[1])              -> 15;
```

Generische For-Schleife

- Um über Tables zu iterieren bietet sich oft die generische Variante der For-Schleife an

```
local myTable = {};

myTable[„name“] = „Hansi“;        -- oder myTable.name = „Hansi“;
myTable[„alter“] = 15;            -- oder myTable.alter = 15;

for key, value in pairs(myTable) do
      print(key .. „: „ .. value);
end;


--[[
Ausgabe:

name: Hansi
alter: 15
]]
```

Table-Funktionen der LUA-API

local myTable = {};

Einfügen neuer Elemente:

```
table.insert(myTable, 15);      -- fügt die Zahl 15 in den Table ein
table.insert(myTable, 1, 10);   -- fügt die Zahl 10 an der Stelle 1 ein
                                -- Wenn Position > Anzahl Table-Elemente -> Wert wird nicht eingetragen!
```

Löschen von Elemente:

```
table.remove(myTable, 1);       -- Löscht das Element an der Position 1
```

Sortieren des Tables:

```
table.sort(myTable);            -- Sortiert den Table mit dem < Operator
```

Anzahl der Elemente des Tables:

```
table.getn(myTable);            --> Anzahl der Elemente
```

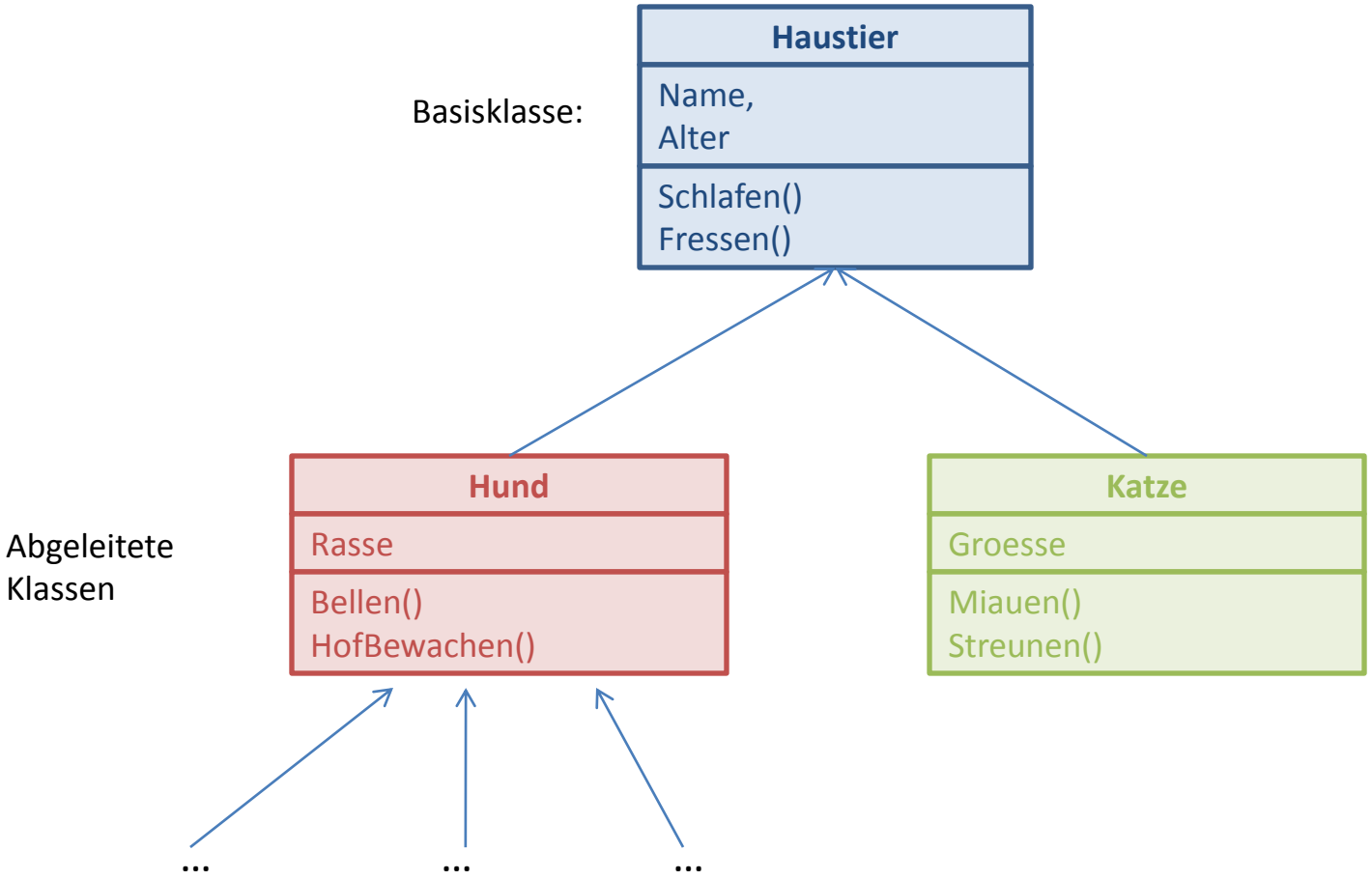
Weitere Funktionen in der API

Gliederung

Objekt-Orientierung

- Merkmale:
  - Zusammenfassung von Eigenschaften(Daten) und Funktionen in einzelne Objekte
  - Daten und Funktionen eines Objektes haben einen logischen Zusammenhang
  - Diese neuen Objekte (auch Datenstrukturen genannt) werden durch Klassen realisiert
- Vererbung:
  - Eine Klasse erbt von einer anderen sogenannten Basisklasse
  - Die neue Klasse besitzt nach der Vererbung alle Eigenschaften und Funktionen der Basisklasse und die, die in ihr selber definiert werden

Beispiel: Haustier



Wenn nun ein Hund-Objekt erzeugt wird, kann dieses sowohl Bellen und HofBewachen als auch Schlafen und Fressen.
Desweiteren hat es die Eigenschaften Name, Alter und Rasse

Gliederung

OO in LUA

- Merkmale:
  - Klassen werden über Tables realisiert
  - Da Tables verwendet werden -> CallByReference

- Definition einer Klasse:

```
Haustier = {};
Haustier.__index = Haustier;

function Haustier:new(name, alter)
	local instance = {};

	setmetatable(instance, Haustier);

	instance.name = name;
	instance.alter = alter;

	return instance;
end;

function Haustier:schlafen()
	print(self.name .. " schlaeft gerade");
end;

function Haustier:fressen()
	print(self.name .. " frisst gerade");
end;
```

OO in LUA

- Verwendung der definierten Klasse:

```
myAnimals = {};


table.insert(myAnimals, Haustier:new("Kiddy", 2));
table.insert(myAnimals, Haustier:new("Hansi", 6));
table.insert(myAnimals, Haustier:new("Doofy", 0.5));


for k, animal in pairs(myAnimals) do
      print(animal.name .. " ist " .. animal.alter .. " Jahre alt");
end;

myAnimals[2]:schlafen();
myAnimals[1]:fressen();

-- BZW:

Haustier.schlafen(myAnimals[2]);
Haustier.fressen(myAnimals[1]);
```

OO in LUA

- Vererbung einer Klasse:

```
Hund = {};
Hund.__index = Hund;

function Hund:new(name, alter, rasse)

    local instance = Haustier:new(name, alter);
    setmetatable(instance, Hund);
    instance.rasse = rasse;

    return instance;
end;

function Hund:Bellen()
    print("Wuff, Wuff, Wuff");
end;

function Hund:HofBewachen()
    print(self.name .. " bewacht den Hof");
end;


local myDog = Hund:new("Hansi", 10, "Schaeferhund");

myDog:Bellen();                 -- “Wuff, Wuff, Wuff”
myDog:HofBewachen();            -- “Hansi beacht den Hof”
print(myDog.alter);             -- 10
```

Metatables

- Merkmale:
  - Definiert Grundfunktionen, wie mit einem Table umgegangen werden soll
  - TableName.__index definiert den Table in dem nach einer Funktion gesucht werden soll, z.B. wenn eine aufgerufene Funktion nicht gefunden wird

  - Alternativ können noch Funktionen definiert werden:
    - __add
    - __sub
    - __mul
    - … genauere Infos hier: http://lua.gts-stolberg.de/MetaTables.php

- Metatables in LS:
  - __index – Definition nicht notwendig

  Stattdessen:

  - *setmetatable(instance, Class(KLASSENNAME));*

  - *Class() setzt intern die Variable __index*

Gliederung

Application Programming Interface

- Merkmale:
  - Auflistung aller Programmiersprachen-internen Funktionen
  - Nachschlagewerk

- http://www.lua.org/manual/5.1/

- Richtig lesen:

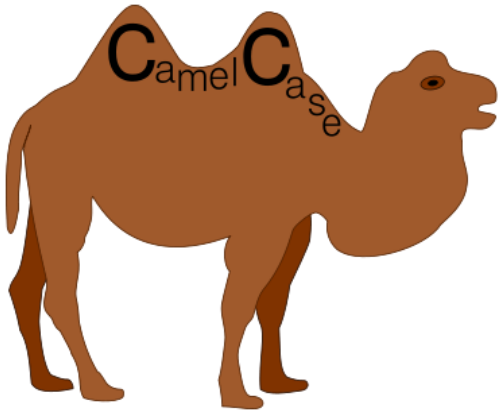
  - Beispiel: *table.remove (table [, pos])*

  - Parameter in [] sind immer Optional!
  - Wenn diese nicht angegeben werden wird ein Default-Wert angenommen
  - Dieser ist in der API nachlesbar

  - Rückgabe-Werte sind ebenfalls im Text angegeben

Gliederung

Coding-Guideline



- Englische Bezeichnungen
- CamelCase-Bezeichungen
- **Aussagekräftige** Bezeichungen!!!!!!!!
- Jeden neuen Block einrücken (entweder per Tab oder mit 4 Spaces)
- Lokale Variablen mit **local** definieren
- Code strukturieren -> Zusammenhängende Blöcke
- Keine unendlich langen Boolean-Ausdrücke
- Klassennamen mit Großbuchstaben beginnen
- Aufzählungen im **Plural-Schreiben**, z.B. myAnimals = {};
- Leerzeichen zwischen Operatoren
- Klammern von boolschen Ausdrücken
- **Kommentare** in Script-Datei
- Zugriffe auf Tables und deren Eigenschaften **nur mit vorhergehender NIL-Prüfung**

```
if myTable.innerTable ~= nil then
        print(myTable.innerTable.name);
End;
```

Fragen / Feedback

- Gibt es noch Fragen?
- Was sollte besser gemacht/geändert werden?
- Welche Inhalte würden euch interessieren oder fehlen noch?