Der Informatik-Zugang zur Selbstreplikation

John von Neumann (40er Jahre): Überlegungen zu einem hypothetischen, replikationsfähigen Roboter

- ausgestattet mit Sensoren, Greifern, Schneide- und Schweissgeräten...
- angewiesen auf riesigen "See" von Einzelteilen

3 Module:

- universeller Konstruktor A
- Kopiermaschine B
- Kontrollmaschine C

Theoretische Herleitung der Arbeitsweise:

Sei X eine beliebige Maschine und $\Phi(X)$ ihre Beschreibung.
"+" bezeichne das Zusammenwirken von Untereinheiten einer Maschine und "$\rightarrow$" bezeichne Konstruktion.

Eigenschaft von A: $A + \Phi(X) \rightarrow X$ für jedes X,
Eigenschaft von B: $B + \Phi(X) \rightarrow \Phi(X)$ für jedes X.

Die Kontrollmaschine C hat die Aufgabe, A und B in der richtigen Reihenfolge zu kombinieren, um X und eine Kopie von $\Phi(X)$ produzieren zu lassen und diese dann untereinander zu verbinden und vom Rest der Maschine zu trennen:

$$A + B + C + \Phi(X) \rightarrow X + \Phi(X) \qquad (*)$$

Wenn man A+B+C für X einsetzt und D nennt, hat man:

$D + \Phi(D) \rightarrow D + \Phi(D)$,
d.h. eine sich selbst replizierende Maschine.

Eine Maschine, die "nur" sich selbst repliziert, ist nicht besonders "nützlich" – deshalb Fortführung der Überlegungen von Neumanns:

Sei F eine beliebige andere Maschine,
und $E_F = D + \Phi(D+F) = A + B + C + \Phi(A+B+C+F)$.

Gemäß (*) gilt:

$$E_F \rightarrow A + B + C + F + \Phi(A+B+C+F),$$

also $E_F \rightarrow E_F + F$

d.h. wir haben eine Maschine, die sich selbst *und* eine beliebige andere Maschine baut.

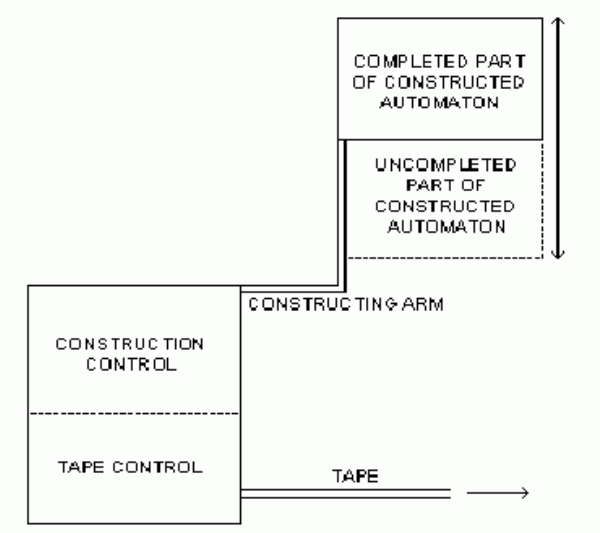
Nachteil: Logik zwar o.k., aber praktische Durchführbarkeit nicht gegeben – die Teile der Maschine haben "black box"-Charakter.

Ausweg (von Neumann 1953): betrachte zelluläre Automaten statt realer Hardware.

> "CA were originally designed to be the universe in which self-replicating 'creatures' could exist" (Pfeifer et al. 2002)

Entwurf eines selbstreplikativen CA durch von Neumann:

- quadratisches Gitter, von-Neumann-Nachbarschaft
- 29 Zellen-Zustände
- "Kontrollzentrum" in 80×400 - Box:
  universeller Konstruktor (im CA-Raum)
- hinzu kommt Band von 150 000 Zellen mit Bauplan
- wenn die Anfangskonfiguration gegeben ist, bildet sich ein "Arm" und beginnt die Konstruktion einer exakten Kopie; zum Schluss wird die "Verbindung" gelöst – beide "Organismen" können den nächsten Selbstreproduktionszyklus beginnen



- von Neumanns CA wurde nie auf einem "realen" CA-Simulator implementiert (zu umfangreich und komplex)
- 1968 Verringerung der Zustandszahl von 29 auf 8 durch Codd (aber mit Moore-Nachbarschaft)

benötigt man tatsächlich die "universelle Konstruktor"-Eigenschaft?

anderes Extrem: *keine* Zusatzforderungen an ein sich replizierendes Muster in einem CA-Gitter
$\rightarrow$ dann gibt es "triviale Selbstreplikation"
Bsp.: CA, der alles außer (00...0) zu 1 macht $\rightarrow$ einzelne "1" vermehrt sich, füllt das ganze Gitter aus

Man kann sogar beweisen:
zu *jeder* beliebigen endlichen Konfiguration *s* eines *d*-dim. CA-Gitters gibt es einen CA auf diesem Gitter, welcher *s* repliziert. (Alvy Ray Smith 1992)

somit: für "nicht-triviale Selbstreplikation" braucht man Zusatzforderung an den CA.

häufige Zusatzforderung: der CA soll als *universeller Computer* funktionieren können, d.h. eine universelle Turingmaschine (TM) emulieren können.

Emulation von TM mittels CA (Prinzip):
- eine Zeile von CA-Zellen simuliert die Felder des Bandes der TM und gleichzeitig den Kopf der TM
- Zustände der CA-Zellen setzen sich aus 2 Koordinaten (x, y) zusammen: x = Symbol auf entspr. Feld des Bandes, y = Zustand der TM (wenn TM gerade das Feld liest) oder Indikatorsymbol für Abwesenheit des TM-Kopfes
- geeignete Transitionsfunktion simuliert Operationen der TM

$\Rightarrow$ jede TM mit m Symbolen und n Zuständen kann durch einen 1-dim. CA mit $m(n+1)$ Zell-Zuständen emuliert werden

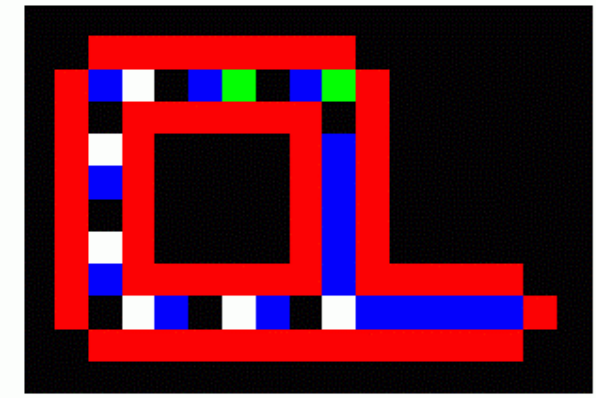
Satz (Alvy Ray Smith 1968, 1992):
Zu jeder Turingmaschine *M* existiert ein CA *Z*(*M*), der *M* emuliert, und eine Konfiguration *c*, die unter *Z*(*M*) selbstreplikativ ist.

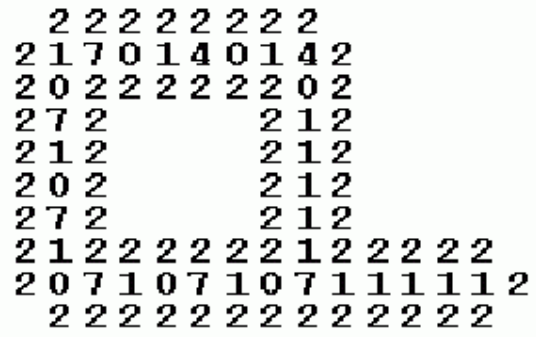
Korollar (wähle *M* als universelle TM):
Es existiert ein berechnungsuniverseller CA *X* und eine Konfiguration *c*, die unter *X* selbstreplikativ ist.

Albert & Culik (1987) fanden einen 1-dim. berechnungsuniversellen CA mit "nächste-Nachbarn"-Nachbarschaft und 14 Zellzuständen.

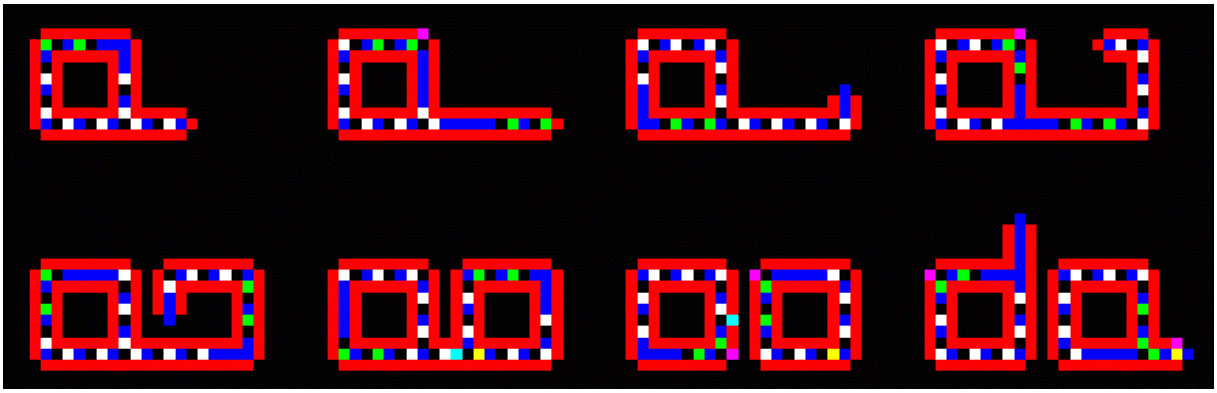
Langton (1979) verzichtete auf Berechnungsuniversalität und beschränkte sich auf nichttriviale morphologische Struktur.

- 2-dim. CA mit von Neumann-Nachbarschaft und 8 Zuständen, aufbauend auf Codd's Ansatz
- "selbstreplikative Schleife" als Konfiguration
- Unterscheidung von "interpretierter" und "nicht-interpretierter" Information beim Aufbau der Schleife
- ein Replikationszyklus dauert 151 Zeitschritte

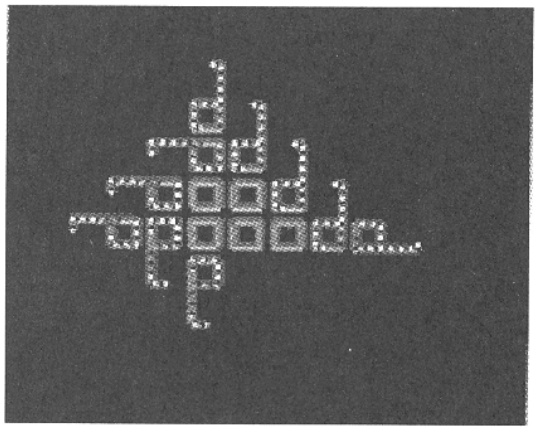


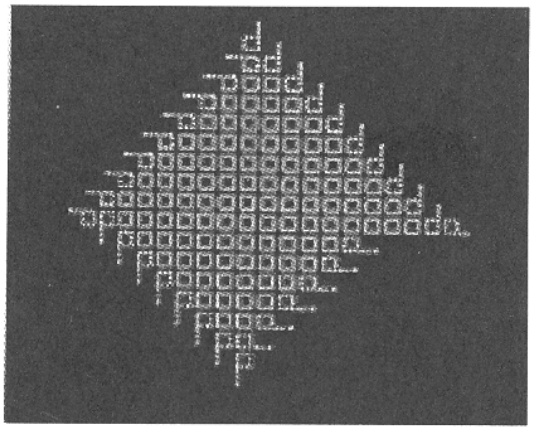


Selbstreplikationsprozess einer Langton-Schleife:

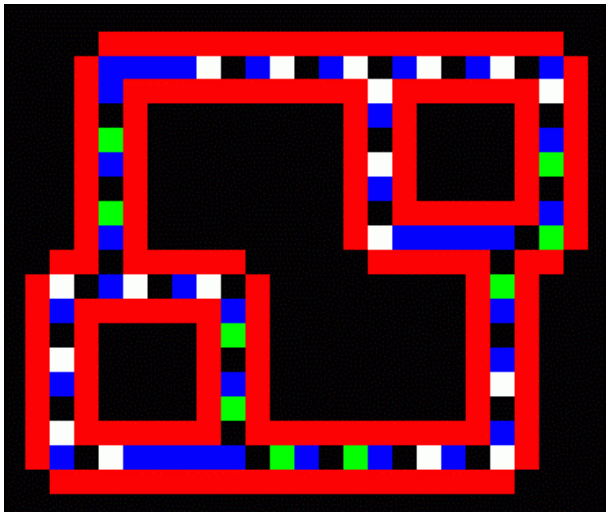


nach vielen Schritten: "Koloniebildung"





- Nachteil der Langton-Schleifen: fehlende Robustheit
- Hiroki Sayama (1999): *Structurally Dissolvable Self-Replicating Loops*; Konfigurationen können sich auflösen und damit Platz für neue "Organismen" geben – ermöglicht "unendliches" Leben in einem CA-Universum mit Rechteck- oder Torus-Topologie

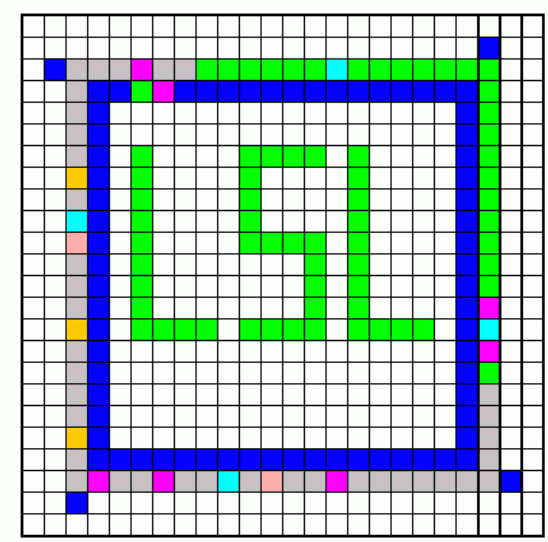


2 kollidierende SDSR-Schleifen

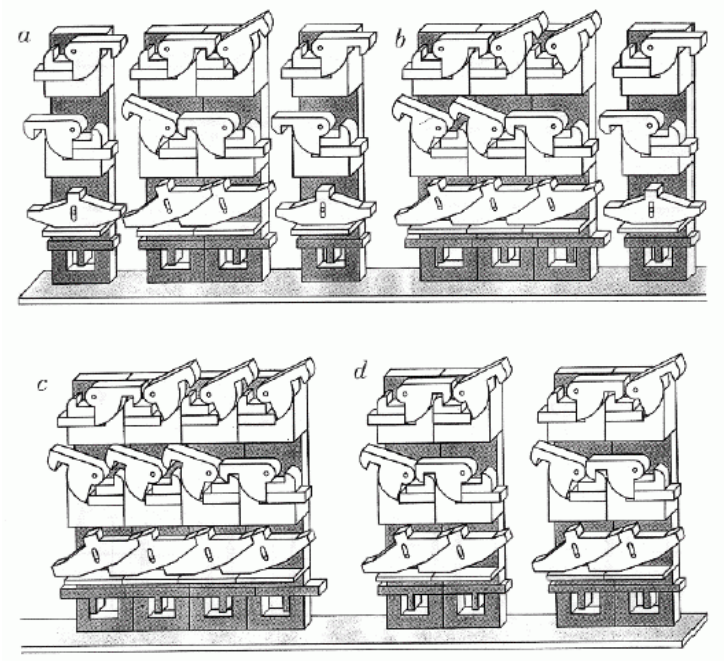
Verlangen nach einer "Zusatz-Fähigkeit", die selbstreplikative CA haben sollen

- Tempesti (1995): CA-Schleifen, die sich replizieren und anschließend endliche Berechnungen ausführen

Beispiel: Schreiben von Buchstaben ("LSL" = Logic Systems Laboratory) ins Innere
Information für die Buchstaben ist im zirkulierenden Datenfluss (um die Schleife) enthalten

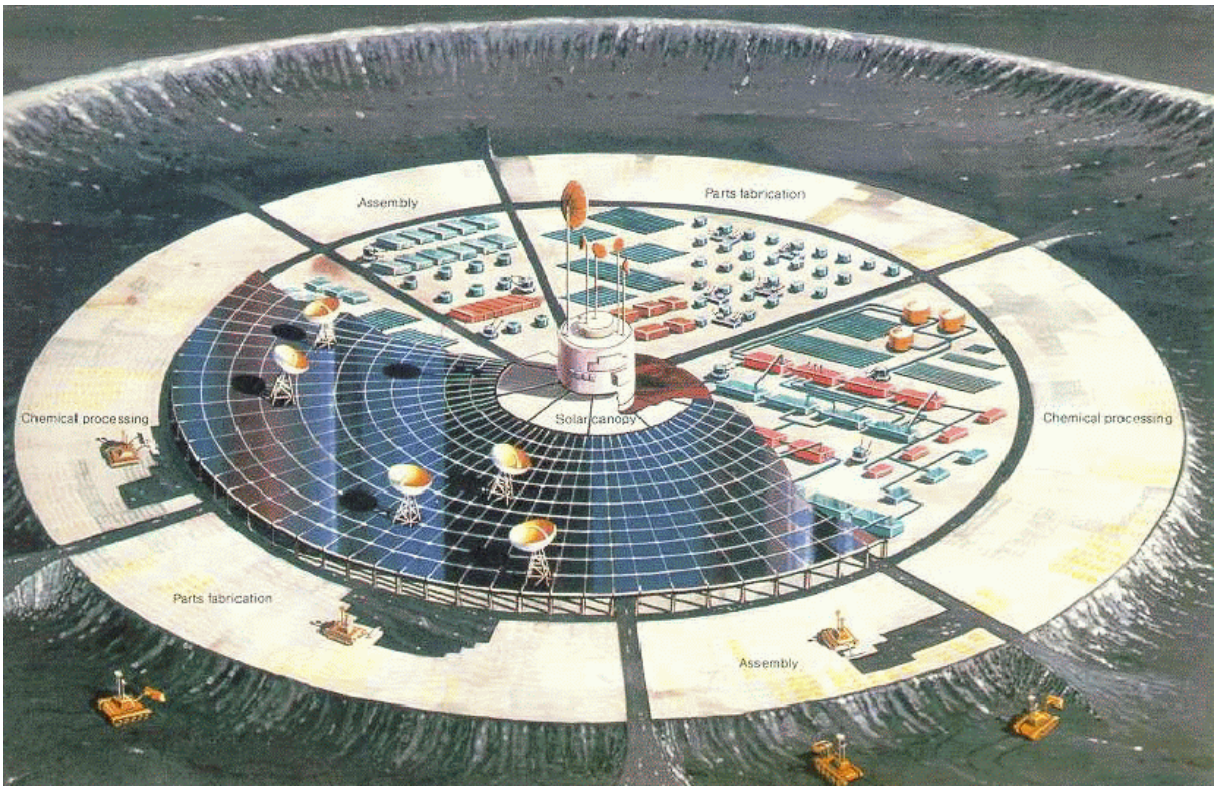


- Mechanische selbstreplikative Maschine von Penrose:



- setzt Vorrat von vorgefertigten Einzelteilen voraus

- 1980-1983: Planungen der NASA für selbstreplizierende Fabriken auf dem Mond



- Ausbeutung der natürlichen Ressourcen
- nur 4–10% der erforderlichen Teile sollten von der Erde nachgeliefert werden

Computerviren

Virus: Organismus, der für seine Vermehrung auf einen Wirtsorganismus angewiesen ist

biologische Viren: DNA-Viren, RNA-Viren
befallen Tiere, Pflanzen, Bakterien...

Computervirus: ein Maschinencode-Segment, das sich (oder eine modifizierte Version von sich selbst) bei Ausführung in 1 oder mehrere "Wirtsprogramme" hineinkopiert (durch Anhängen oder Überschreiben von Code) und sich so "vermehrt"

verwandtes Phänomen:
Wurm = vollständiges Programm, das sich selbst repliziert (evtl. über Netze in andere Rechner), aber nicht auf Veränderung fremder Programme angewiesen ist

- siehe zu diesem Thema auch ausführlicher:
Referat "Computerviren als künstliches Leben" von Maik Krüger im Proseminar "Ethische Aspekte der Informationsverarbeitung" (Sommersem. 2002); **`http://www-gs.informatik.tu-cottbus.de/~wwwgs/ea_th20.html`** -

- Begriff "Virus" im Zusammenhang mit Computer-Code tauchte zum ersten Mal 1972 in einem Science-Fiction-Roman auf
- Fred Cohen definierte den Begriff "Computervirus" 1983 und entwickelte experimentelle Computerviren
- erste Viren aber schon 1981 auf Apple-Rechnern
- erster Wurm 1982 von John Shoch und Jon Hupp (Xerox) programmiert, inspiriert durch einen SF-Roman von 1975
- 1986 Brain-Virus: erster bedeutender MS-DOS-Virus
- 1988 Internet-Wurm legt Tausende von Maschinen lahm
- seitdem viele Tausend Computer-Viren-Arten bekannt; Schutz vor Viren wirtschaftlich relevante Aktivität

Beispiel eines historischen Apple-Wurms in Basic:

```
1   IF PEEK (104) = 134 GOTO 10
2   POKE 104, 134: POKE 134 * 256,0
3   PRINT CHR$(4) "RUN APPLE WORM"
10  HOME : POKE - 16302,0: POKE - 16304,0: POKE 1023,160
20  FOR I = 0 TO 94: READ D: POKE 1024 + I, D: NEXT I
30  POKE - 16368,0
40  IF PEEK ( - 16384)<128 GOTO 40
50  CALL 1024
100 DATA 160,225,200,185,255,3,153,127,4,192,95,208,245,
    160,18,190,76,4,24,189,128,4,105,128,157,128,4,189,129,
    4,105,0,157,129,4,192,13,208,18,238,23,4,173,23,4
200 DATA 141,151,4,206,31,4,173,31,4,141,159,4,136,208,211,
    173,167,4,72,173,176,4,141,167,4,104,141,176,4,76,128,
    4,7,20,25,28,33,46,55,61,65,68,72,75,4,16,40,43,49,52
```

(nach Dewdney 1985)

"Da der Wurm in einen der Grafik-Bereiche des Rechners geladen wird, können Sie zusehen, wie er (im Krebsgang) in die oberen Speicherbereiche marschiert... Nachdem er den Grafikbereich verlassen hat, können Sie warten, bis er auch den ganzen oberen Teil des Speichers (samt Basic-Interpreter) gelöscht hat und in die System-ROMs kracht..."

Funktionstypen von Viren:

- Infektion der Partitionstabelle einer Festplatte:

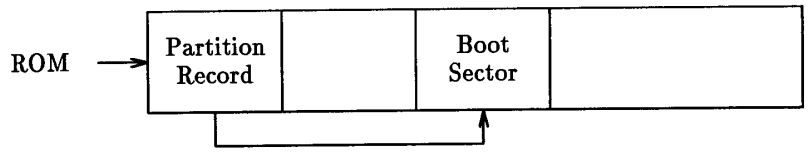


FIGURE 4 Hard disk before infection.

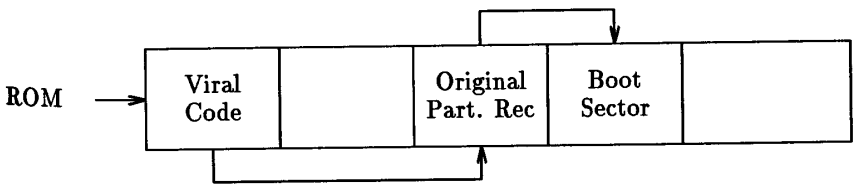


FIGURE 5 Hard disk after infection by New Zealand Virus.

- Infektion des Bootsektors:

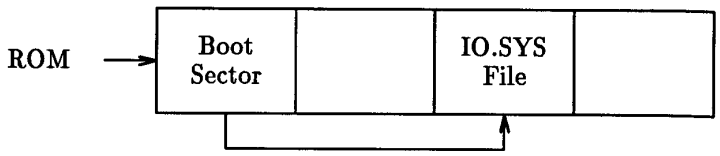


FIGURE 6 Floppy disk before infection.

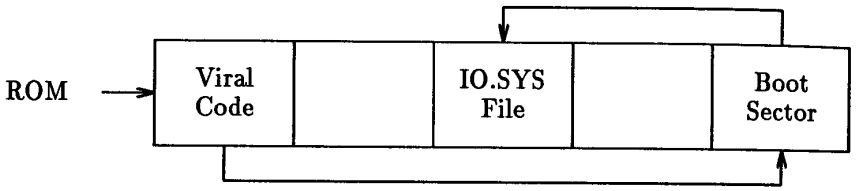
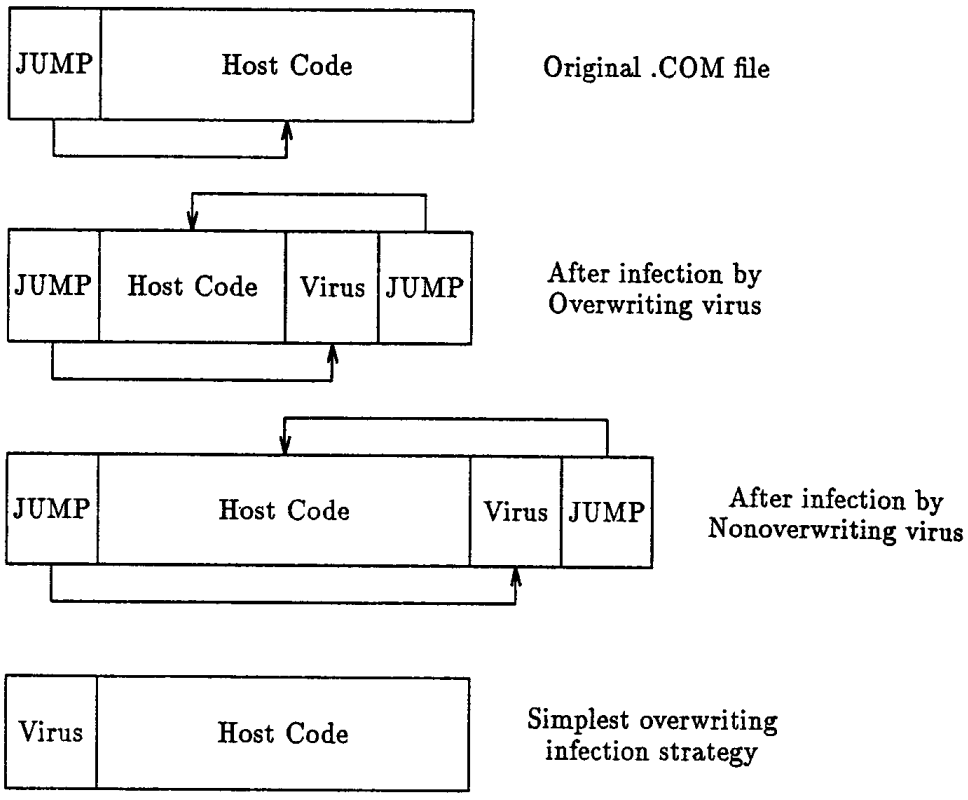


FIGURE 7 After Alameda Virus Infection.

- Infektion von Shell-Programmen wie **`command.com`**, **`autoexec.bat`**, **`profile`** ... die regelmäßig ausgeführt werden
- Infektion von Anwenderprogrammen oder von Makros in Anwenderdaten



- Interrupt-Viren:

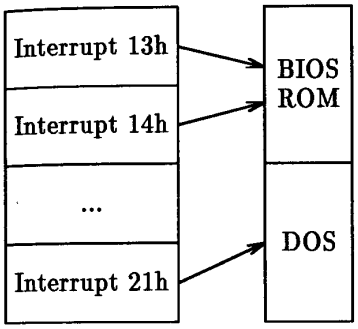


FIGURE 9 Normal interrupt usage.

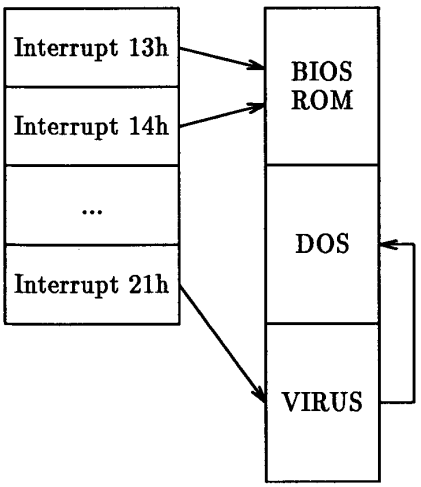


FIGURE 10 Interrupt vectors with TSR virus.

- Multipartite Viren (z.B. Infektion von Dateien + Master-bootsektor)

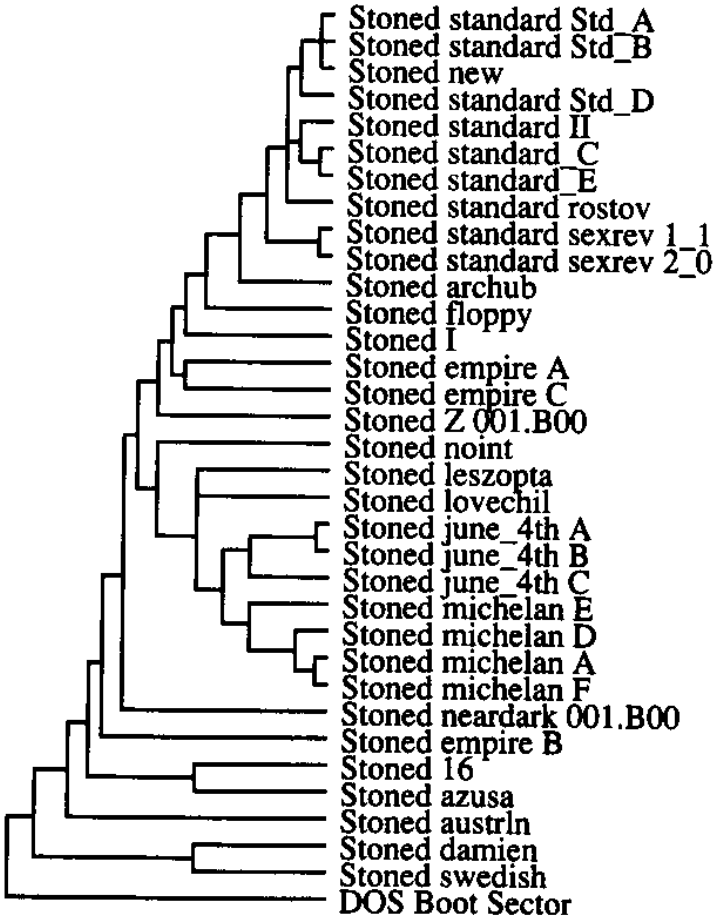
Die meisten Viren haben (neben der Replikation) "Neben-wirkungen" (mehr oder weniger destruktiver Art) – "Aktivität"

ALife-Eigenschaften von Computerviren und -würmern:

- Existenz in Zeit und (virtuellem) Raum (aber keine "Morphologie" im klassischen Sinne)
- Replikation
- Aktivität in ihrer Umgebung
- Metabolismus? (Verbrauch von Energie und Speicherplatz, Umwandeln fremder Programme)
- Persistenz
- Kooperation: eher nicht, aber "Räuber-Beute"-Relationen

zur *Evolution* von Computerviren:

"Mikro-Evolution" am Beispiel des "Stoned"-Bootsektorvirus (aus Adami 1999):



"Makro-Evolution":

1. Viren-Generation: einfache Viren (leicht zu erkennen an Vergrößerung des Wirtsprogramms, bes. bei Mehrfach-infektion)
2. Generation: selbsterkennende Viren (vermeiden Mehrfachinfektion)
3. Generation: Stealth-Viren (Irreführung von Antiviren-programmen)
4. Generation: Armored (Verschleiern des Virencodes, Angreifen der Antivirensoftware)
5. Generation: polymorphe / mutierende Viren (nicht anhand fester Muster zu entdecken)

Beachte: diese "Evolution" ist keine Darwinsche (selbstorganisiert ablaufende) Evolution, sondern auf menschliches Wirken zurückzuführen

generell ist bei Computerviren die menschliche Beteiligung essenziell:

- Entwurf + Schreiben + in Umlauf bringen des Virus
- Verbreitung durch Aufrufen von Programmen, Versenden von e-mails usw.

extreme Form hinsichtlich Beteiligung des Menschen:

"*Hoaxes*" (Scherz-e-mails, z.B. haltlose Virenwarnungen; mitleiderregende Ketten-Mails ohne reale Basis)

$\Rightarrow$ hier ist die nur vom Menschen interpretierbare Botschaft schon das "Virus"

Somit: Studium von Viren nur eingeschränkt für ALife relevant

Eugene H. Spafford:

> More seriously, I would suggest that there is something to be learned from the study of computer viruses: the importance of the realization that experimentation with systems in some way (almost) alive can be dangerous. Computer viruses have caused millions of dollars of damage and untold aggravation. Some of them have been written as harmless experiments, and others as malicious mischief. All have firmly rooted themselves in the pool of available computers and storage media, and they are likely to be frustrating users and harming systems for years to come. Similar but considerably more tragic results could occur from careless experimentation with organic forms of artificial life. We must never lose sight of the fact that "real life" is of much more importance than "artificial life," and we should not allow our experiments to threaten our experimenters.

"Krieg der Kerne" (Core Wars)

A. K. Dewdney (Scientific American / Spektrum der Wissenschaft, 1984/1985/1987)
s.a. **`http://www.sci.fi/~iltzu/corewar/guide.html`** u.a. Webseiten

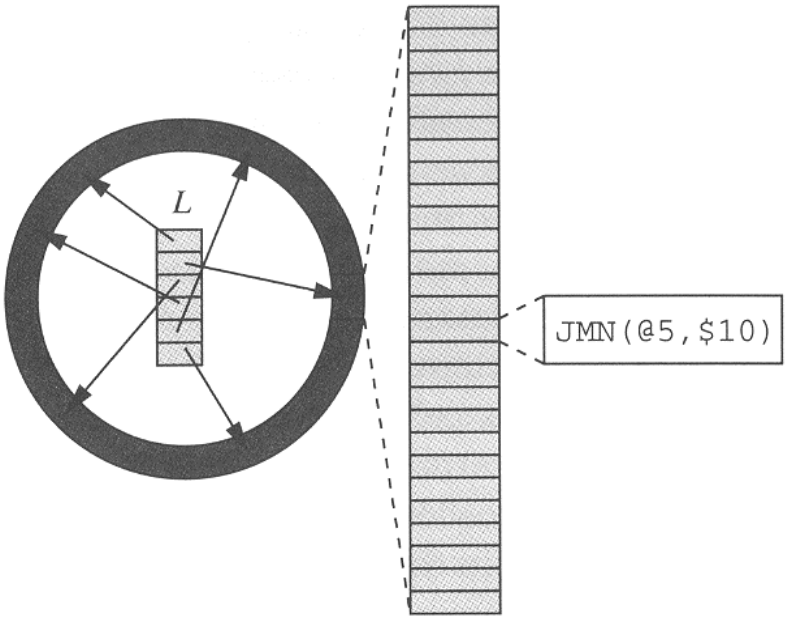
Spiel, bei dem mehrere Programme um die Vorherrschaft im Arbeitsspeicher "kämpfen"

4 Komponenten:

- ringförmig geschlossenes Speicherfeld mit 8000 (oder mehr) Adressen
- Assembler-Sprache "Redcode"
- Steuerprogramm MARS (= *Memory Array Redcode Simulator*)
- zwei oder mehr wettstreitende Kampfprogramme

Ablauf eines Spiels:

- Die Kampfprogramme werden auf zufällig ausgewählte Plätze des Speicherfelds geladen – kein Programm weiß, wo sich das andere befindet
- MARS führt die Programme nach einfachem Timesharing-Prinzip abwechselnd aus



L = von MARS verwaltete Ausführungsschlange

- ein Programm kann seinen Ablauf auf mehrere Anweisungen "aufspalten" (Befehl SPL, erst ab 1985) – die Instanzen des Programms kommen dann aber entsprechend seltener "an die Reihe"
- Der Kampf endet, wenn MARS auf einen nicht ausführbaren Befehl stößt (das Programm mit der fehlerhaften Anweisung wird dann zum Verlierer erklärt) oder wenn eine maximale Schrittzahl erreicht ist (wenn dann noch beide Programme vorhanden sind: unentschieden).

Der Befehlsvorrat von Redcode (Version von 1987):

Anweisung	Kürzel	Code	Argumente	Erklärung
Data-Anweisung	DAT	0	B	Nichtausführbare Anweisung; B ist der Datenwert.
Übertrage	MOV	1	A B	Übertrage Inhalt von Adresse A auf Adresse B.
Addiere	ADD	2	A B	Addiere Inhalt von Adresse A zu Adresse B.
Subtrahiere	SUB	3	A B	Subtrahiere Inhalt von Adresse A von dem von Adresse B.
Springe	JMP	4	A	Übergib die Ausführung an Adresse A.
Springe, wenn null	JMZ	5	A B	Übergib die Ausführung an Adresse A, falls der Inhalt von Adresse B null ist.
Springe, wenn nicht null	JMN	6	A B	Übergib die Ausführung an Adresse A, falls der Inhalt von B ungleich null ist.
Vermindere; springe, wenn nicht null	DJN	7	A B	Ziehe vom Inhalt der Adresse B 1 ab und übergib die Ausführung an Adresse A, falls Inhalt von Adresse B ungleich null ist.
Vergleiche	CMP	8	A B	Vergleiche Inhalt der Adressen A und B; falls er ungleich ist, übergehe die nächste Anweisung.
Spalte auf	SPL	9	A	Spalte die Ausführung auf in die nächste Anweisung und die bei A.

hinzu kommt (zur Erleichterung der Lesbarkeit) die Verwendung von Strings als Marken

3 verschiedene Adressierungsmodi:

#n unmittelbar (Zahl *n* statt Adresse *n*)
n direkt (relative Adresse *n*)
@n indirekte Adressierung (Inhalt von Zelle mit rel. Adresse *n* wird als neue rel. Adresse interpretiert)

Kürzel	Argument A	Argument B	Anweisungscode	Modusangabe Argument A	Modusangabe Argument B	Argument A	Argument B
DAT		−1	0	0	0	0000	7999
ADD	#5	−1	2	0	1	0005	7999
MOV	#0	@−2	1	0	2	0000	7998
JMP	−2		4	1	0	7998	0000

Adressierungsmodi:			
	unmittelbar	#	0
	direkt		1
	indirekt	@	2

einfachstes "Kampfprogramm":

```
MOV 0 1
```

("Knirps") – kopiert Inhalt der rel. Adresse 0 (nämlich sich selbst) in die rel. Adresse 1 (= nächste Adresse)
⇒ überschreibt nach und nach gesamten Speicher mit Kopien von sich selbst

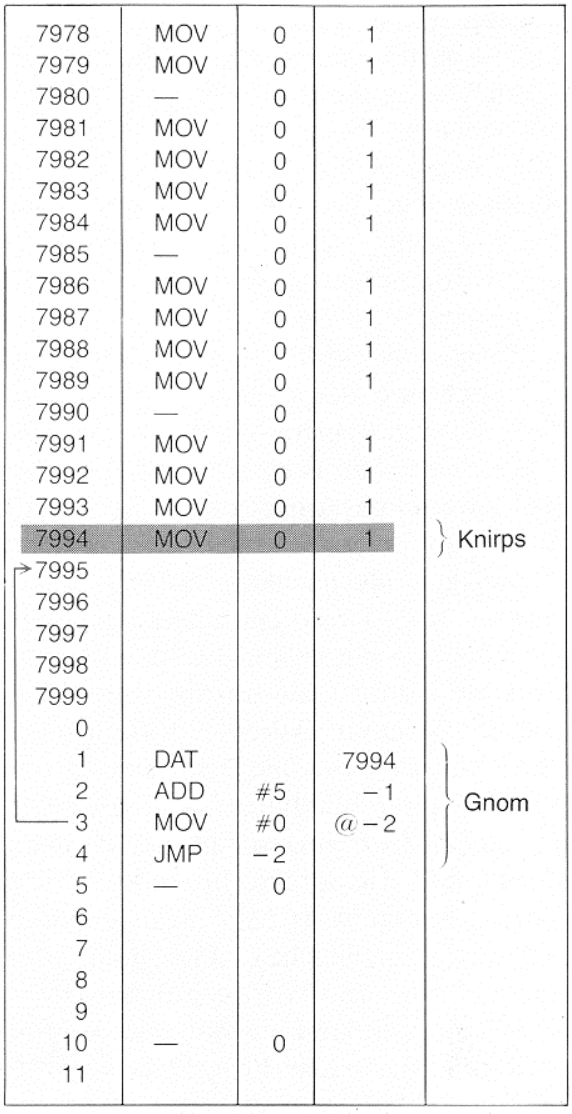
anderes Beispiel: "Gnom"

Adresse	1. Zyklus	2. Zyklus	9. Zyklus
0			
1	DAT −1	DAT 4	DAT 14
2	ADD #5 −1	ADD #5 −1	ADD #5 −1
3	MOV #0 @−2	MOV #0 @−2	MOV #0 @−2
4	JMP −2	JMP −2	JMP −2
5		— 0	— 0
6			
7			
8			
9			— 0
10			
11			
12			
13			
14			— 0
15			
16			
17			

Gnom legt Sperrfeuer aus "Null-Bomben" im Abstand 5

"Gnom" selbst ist unbeweglich, aber seine Artillerie bedroht das gesamte Speicherfeld. Schließlich erreicht Gnom die Adressen 7990, 7995 und 8000. Für MARS ist 8000 gleichbedeutend mit 0, und so hat Gnom haarscharf am Selbstmord vorbeigesteuert. Sein nächstes Geschoss landet wieder auf Adresse 5.

Kampf "Knirps gegen Gnom":



- offensive Strategien: z.B. "Knirps-Kanone" mittels SPL-Befehl; Programme mit exponentieller Selbstreplikation ("Mice" = Turniersieger von 1986 in Boston)
- defensive Strategien: Entdeckung von Angriffen, Ausweichmanöver (z.B. "Raubritter")

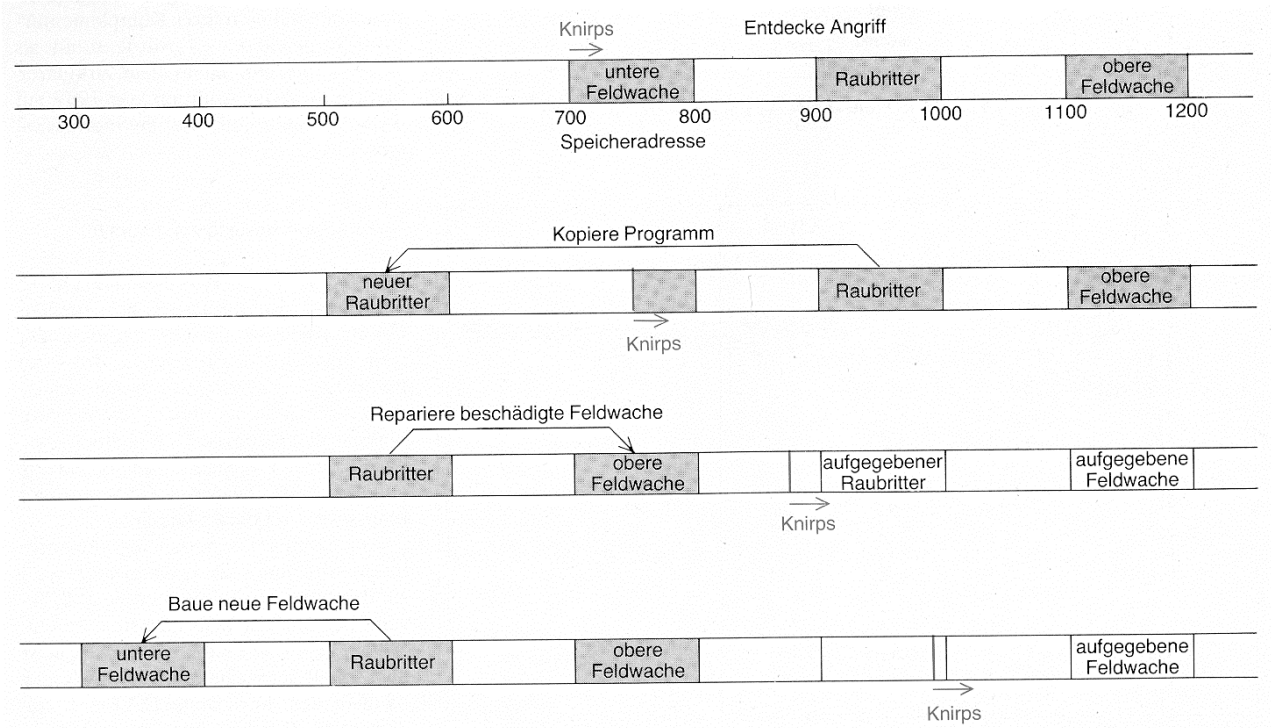


Bild 1: Raubritter, ein ausgefuchstes Kampfprogramm, weicht dem schlichteren Knirps auf dem Kernspeicher-Schlachtfeld des Kriegs der Kerne aus.

Das selbstreplikative Programm "Mice" ("Mäuse"):

MICE

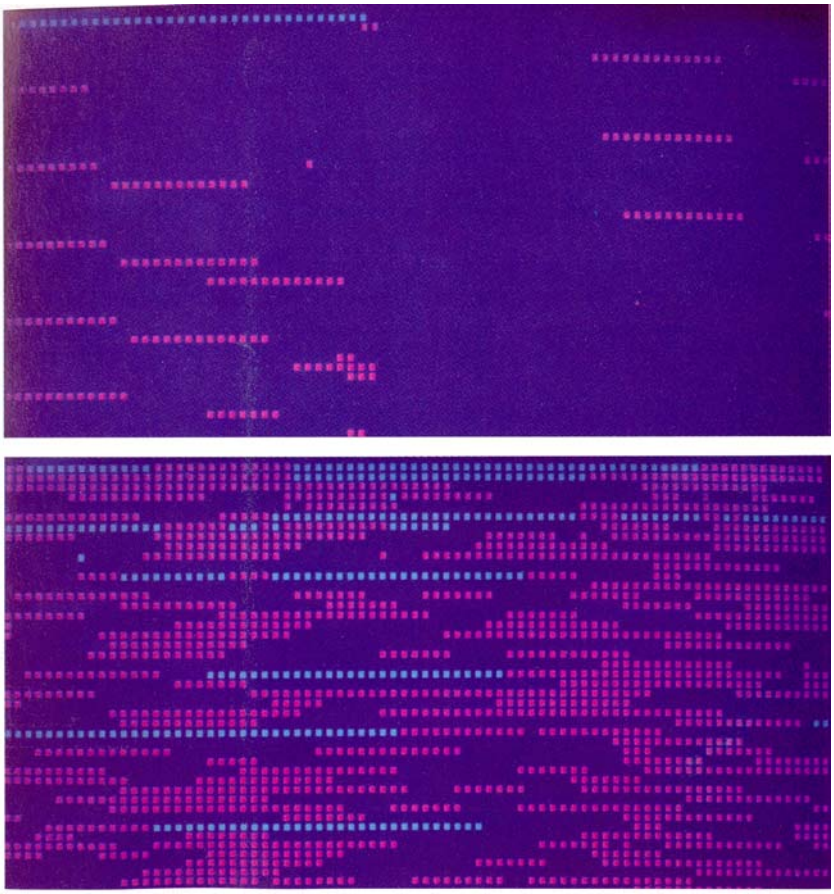
```
ptr   DAT   #0
start MOV   #12    ptr
loop  MOV   @ptr   <5
      DJN   loop   ptr
      SPL   @3
      ADD   #653   2
      JMZ   -5     -6
      DAT   833
```

Visualisierung des Kampfes durch Färbung der durch verschiedene Programme belegten Speicherfelder mit unterschiedlichen Farben und ihre Darstellung auf dem Bildschirm

Ausscheidungsrunde beim Turnier: 4 aufeinanderfolgende Schlachten, Zeitlimit pro Kampf = 15 000 Befehle für jede Seite

Schilderung des Entscheidungskampfes zwischen "Mice" und seinem Final-Konkurrenten "Chang1" (Dewdney 1987):

> "Ein auf die Kopierschleife folgender SPL-Befehl erweckt die neue Kopie [von *Mice*] zum Leben. Zugleich aber beginnt das alte Programm mit der Zeugung des nächsten Sprösslings (den es 653 Adressen hinter dem vorigen plaziert). ... Dementsprechend pflanzte sich das *Mice*-Programm... mit unglaublicher Geschwindigkeit fort. Bald war der Schirm über und über bedeckt mit kleinen roten Streifen. In der Zwischenzeit hatte *Chang1* an seinem Ende allerdings eine Art *Knirps*-Fabrik in Betrieb genommen ... die gewaltige Horde schob sich unheildrohend als immer längerer blauer Streifen am oberen Ende des Bildschirms langsam, aber stetig nach rechts voran. Würde sie *Mice* überrollen können?
> Während sich die *Knirps*e reproduzierten, verschoss *Chang1* zugleich eine Serie von Datenbomben... Während dieses Granatfeuer bereits einigen Mäusen den Garaus machte, begannen die Knirpse ihr zusätzliches Unterwanderungswerk. Aber jede Kopie des Mäuseprogramms enthält eine Selbstmord-Option. Sie prüft regelmäßig, ob ihre erste Anweisung (eine Datenanweisung, die nur aus einer 0 besteht) immer noch gleich 0 ist. Wenn nicht, läuft *Mice* auf eine (nicht ausführbare) Datenanweisung und stirbt lautlos, anstatt seine Seele an den bösen Feind zu verlieren..."



spätere Entwicklung:

- statt der Einzel-Turniere ein "Dauerturnier", wo jeder sein Programm in die "Arena" entlassen kann
  – Siegerprogramm (immer nur temporärer Sieger): "King of the Hill" (*KOTH*)

- "Nachfolge-Spiel": *RoboCom* (auf 2-dim., schachbrett-artigem Spielfeld und komplexeren Möglichkeiten), siehe WWW

Fazit:

- Kampfprogramme von menschlichen Programmierern entworfen
- Determinismus (keine Zufallskomponente, keine Mutationen)
- Emergenz liegt im Verhalten beim Zusammentreffen der Programme (mühsam vorauszuberechnen)
- Replikation
- Parasitismus
- Selektion (Fitness unmittelbar durch Überlebenserfolg definiert)

- "Krieg der Kerne" lieferte Ideen für spätere ALife-Programme mit Evolutionsfähigkeit (*tierra*, *avida*)