Mars und Venus im Krieg der Kerne

Von Knirpsen, Mäusen, Viren, Würmern und der Evolution im Computer

von Helmut Witten

Böse Zungen behaupten, dass Viren und Würmer in vernetzten Informatiksystemen die bislang erfolgreichste Form künstlichen Lebens seien. A. K. Dewdneys Computerspiel *Core War* hat in der Geschichte dieser Computerschädlinge eine wichtige Rolle gespielt. Dieses Spiel ist aber auch noch unter anderen Aspekten für den Informatikunterricht interessant: Die assemblerartige Sprache REDCODE, in der die „Kämpfer" programmiert werden, lässt sich verwenden, um Prinzipien der maschinennahen Programmierung zu vermitteln. Das Programm *Mars*, das den Kampf der Gladiatoren im *Krieg der Kerne* steuert, ist ein didaktisch geschickt vereinfachtes Modell des Von-Neumann-Rechners. Darüber hinaus hat *Core War* auch die berühmten Programme *Venus*, *Tierra* und *Avida* zur Simulation der Evolution im Computer inspiriert.

IT-Sicherheit im Unterricht

Erstaunlicherweise spielen Viren, Würmer und Trojaner nach meinen Beobachtungen im Informatikunterricht nur eine marginale Rolle, z. B. als Thema für Schülerreferate. Auch in der Hochschulinformatik wird die Computersicherheit nur am Rande behandelt. Um dem abzuhelfen, hat der neue GI-Präsident Matthias Jarke angekündigt, die Verankerung der IT-Sicherheit in der Ausbildung „mit großer Aggressivität" voranzutreiben.

Der bekannte Datensicherheits-Experte Klaus Brunnstein verlangte bereits auf der *INFOS 2001* in Paderborn ein Umdenken: „Die bisherige Entwicklung der Schulinformatik in Deutschland hat es versäumt, den Lehrern wie auch den Schülern ein angemessenes Verständnis der Risiken und der Un-Beherrschbarkeit heutiger Computer- und Netzsysteme nahe zu bringen. [...] Der Autor fordert, die bisherige Curriculumentwicklung drastisch zu revidieren, um die Beherrschbarkeit unsicherer Informationstechniken in den Vordergrund zu stellen" (Brunnstein, 2001, S. 9).

Wie kann man dieser Forderung im hier betrachteten Bereich schädlicher Programme nachkommen? Sicherlich reicht es nicht aus, die üblichen Ratschläge zu propagieren: Installation von Virenscannern mit regelmäßiger Aktualisierung der Virensignaturen, Einrichtung einer „Firewall", zeitnahe Einspielung der Korrekturprogramme usw. Im Informatikunterricht soll es ja auch um ein *Verständnis* der zugrunde liegenden Techniken gehen. Aus nahe liegenden Gründen verbietet sich aber der Einsatz und ggf. die Weiterentwicklung oder Neuprogrammierung „echter" Viren.

In diesem Beitrag wird vorgeschlagen, die Frage „Wie kann ein Virus funktionieren?" in eine Unterrichtsreihe zur maschinennahen Programmierung einzubetten. Das Programmsystem der Wahl ist dabei der „Krieg der Kerne", da es dort möglich ist, die Wirkungen der Modifikation von fremden oder eigenen Programmen in einer geschützten Umgebung („sandbox") zu studieren, ohne dass weitere Schäden auftreten können.

Wie alles anfing

Im Jahr 1980 verfasste Jürgen Kraus am Fachbereich Informatik der Universität Dortmund eine Diplomarbeit mit dem Titel *Selbstreproduktion bei Programmen*, in der er beschrieb, dass und wie sich Computerprogramme analog zu biologischen Viren verhalten können. Da Kraus aber nicht auf das Problem der PC-Sicherheit einging, verschwand die Arbeit im Archiv der Universität (Kinnebrock, 1996, S. 45; BSI, 2004).

Daher wird in der Literatur meist Fred Cohen als derjenige genannt, der sich als Erster 1983/84 wissenschaftlich mit Computerviren beschäftigte. Von ihm stammt die noch heute gültige Definition des Begriffs:

Bild 1: Alexander Keewatin („Kee“) Dewdney.

```
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 160,255,200,185,255,3,153,127,4,192,95,208,245,
    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, 1988b, S. 132

Bild 2: Der Apfel-Wurm.

„Ein Computervirus ist ein Programm, das andere Programme ‚infizieren‘ kann, indem er sie so modifiziert, dass sie eine – möglicherweise mutierte – Kopie des Programms enthalten“. Die Bezeichnung „Virus“ wurde übrigens von Cohens Mentor Len Adleman vorgeschlagen, der u. a. dadurch bekannt wurde, dass er mit Rivest und Shamir den RSA-Verschlüsselungsalgorithmus erfunden hat (Harley u. a., 2002, S. 56).

Wann die ersten Viren und Würmer tatsächlich in Computern angetroffen wurden, ist nicht mehr zu klären; es kursieren dazu verschiedene Geschichten und Legenden. Alexander Keewatin „Kee“ Dewdney (Bild 1), der ab 1984 die Rubrik *Computer-Kurzweil* im *Spektrum der Wissenschaft* (*Computer Recreation* im *Scientific American*) betreute, schrieb dazu (Dewdney, 1988a, S. 124):

> „Die Anregung zum Krieg der Kerne kam von einer Geschichte, die ich vor einigen Jahren über einen boshaften Programmierer gehört hatte. Im Forschungslabor einer großen Firma schrieb er ein Programm namens *Kriecher* (Creeper), das sich bei jedem Lauf verdoppelte. Dabei konnte es im Verbundsystem der Firma auch von einem Computer zum nächsten überspringen. Das Programm hatte keine andere Funktion, als sich selbst fortzupflanzen. Nicht lange, und es gab von Kriecher so viele Kopien, daß nützlichere Programm und Daten überwuchert wurden.
> Die wachsende Plage war nicht unter Kontrolle zu bringen, bis jemand auf die Idee kam, mit den gleichen Waffen zurückzuschlagen. Er schrieb ein zweites, selbstreproduzierendes Programm namens *Schnitter* (Reaper). Es hatte die Aufgabe, so lange Kopien von Kriecher zu zerstören, bis keine mehr zu finden waren und sich dann selbst zu vernichten. Schnitter tat seine Pflicht und Schuldigkeit, und bald nahmen die Dinge im Labor X wieder ihren gewohnten Lauf.
> Trotz recht offensichtlicher Lücken in der Geschichte glaubte ich sie – vielleicht, weil ich sie glauben wollte. [...] Mein Wunsch zu glauben wurde genährt von der faszinierenden Vorstellung zweier Programme, die in den dunklen, lautlosen Fluren des Kerns einander bekämpfen.
> Letztes Jahr kam ich zu dem Schluß, daß – auch wenn die Geschichte erfunden sein sollte – etwas ähnliches durchaus machbar sei. Ich schrieb eine erste Version vom *Krieg der Kerne* und brachte sie unter Mithilfe von David Jones, einem Studenten in meiner Abteilung an der Universität von Western Ontario, zum Laufen.“

Da sich *Creeper* und *Reaper* ohne Hilfe eines Wirtsprogramms vermehren konnten, handelte es sich – falls die Geschichte stimmen sollte – um Würmer und nicht um Viren. Andererseits ist die Unterscheidung zwischen Viren und Würmern etwas künstlich, da auch die Würmer ein Betriebssystem, Mailprogramm oder Netz zur Vermehrung benötigen (vgl. Harley, 2002, S. 42). Wie dem auch sei – Dewdney veröffentlichte in seinem zweiten Artikel zu *Core War* den Programmtext eines Wurms (geschrieben im Assemblercode für den Mikroprozessor 6502), der sich im *Apple II* ausbreiten konnte (Bild 2).

Obwohl es heute zum Ehrenkodex der Antivirenkämpfer gehört, kein Virenprogramm zu veröffentlichen, sollte man Dewdney deswegen nicht verurteilen, da Viren seinerzeit eher als intellektuelle Spielerei gesehen wurden. Immerhin schrieb er in dem genannten Artikel: „Einige Möglichkeiten sind so erschreckend, daß ich sie kaum wiederzugeben wage“ – eine wirklich hellsichtige Bemerkung.

Rückblickend wird die Rolle von *Core War* bei der Herausbildung der Virenplage von Steven Levy eher skeptisch beurteilt (Levy, 1996, S. 397):

> „Kee Dewdneys Veröffentlichung [hatte] eine deutliche, wenn auch unwillkommene Signalwirkung in der Computer-Subkultur, die sich mit der Erschaffung dieser Art von Kreaturen beschäftigte. Auf der einen Seite sahen viele Programmierer darin genau die Art von Zerstreuung, die Dewdney mit dem Krieg der Kerne hatte schaffen wollen [...]. Außerdem beeinflusste der Krieg der Kerne die Methode einiger wichtiger KL-Experimente. Auf der anderen Seite halfen Dewdneys Artikel aber auch, die Neuigkeiten über das zerstörerische Potential der aus Informationen bestehenden Organismen zu verbreiten, und waren so durchaus nicht unbeteiligt an einem epidemischen Auftreten gefährlicher Programme.“

Da heutzutage jeder von dem enormen Schadenspotenzial der „Malware“ weiß, kann dies nicht mehr als Argument gegen den Unterrichtseinsatz des *Kriegs der Kerne* angeführt werden. Natürlich hat sich Dewdney dagegen verwahrt, damit zur Virenseuche beigetragen zu haben: „Ein Programm für dieses Spiel überlistet nicht unschuldige Betriebssysteme, sondern legt sich mit seinesgleichen an. [...] Dabei besteht keine Gefahr, daß ein Kernkriegsprogramm der Kampfarena entkommen und in der realen Welt Schaden anrichten könnte“ (Dewdney, 1992, S. 126 f.).

Komponenten des „Kriegs der Kerne"

Der Ausdruck „Krieg der Kerne" spielt einerseits auf die Filme aus der Reihe *Krieg der Sterne* (Star Wars) an, andererseits ist der Name von einer mittlerweile veralteten Speichertechnologie (Magnetkernspeicher; vgl. LOG IN 2/2001, S. 64 ff.) aus den Fünfziger- und Sechzigerjahren des letzten Jahrhunderts abgeleitet.

Zum *Krieg der Kerne* gehören vier Komponenten:

- Erstens ist dies die *Arena* (auch *Kolosseum* genannt), ein ringförmig geschlossenes Feld mit 8000 Speicherzellen,
- zweitens eine Miniatur-Assemblersprache namens REDCODE („reduzierter Code"),
- drittens ein Organisationsprogramm, genannt *Mars* (Akronym für ***M***emory ***A***rray ***R***edcode ***S***imulator), und
- viertens zwei Programme, *Kämpfer* oder *Gladiatoren* genannt, die mit REDCODE programmiert wurden.

Sind die Kämpfer geschrieben, werden sie von *Mars* an zufällig gewählten Stellen in der Arena platziert. Sie haben dabei einen gewissen Mindestabstand voneinander, wissen aber weder, an welcher Stelle des Speichers sie stehen, noch, wo sich das gegnerische Programm befindet.

Wenn der Kampf beginnt, kommen beide Programme abwechselnd zum Zug. Sie können dabei jede Zelle der Arena verändern – egal ob diese unbesetzt ist, oder ob sich darin eine Anweisung des gegnerischen oder gar des eigenen Programms befindet. Diese Möglichkeit der Modifikation der Programme macht den Reiz des Spiels aus; mehr noch: Ohne diese Möglichkeit würde kein Kämpfer den anderen besiegen können.

In der „wirklichen Welt" der heutigen Computer ist die scheinbar gleichzeitige Ausführung von Programmen (Multitasking) Standard. Da die Architektur der meisten heutigen Computer trotz einiger Modifikationen im Wesentlichen auf John von Neumann zurückgeht (siehe Kasten: Die *Mars*-Maschine als Von-Neumann-Rechner, S. 27), gibt es im Speicher keinen Unterschied zwischen Daten und Programmen. Damit sich die laufenden Programme nicht gegenseitig stören, wird für jedes Programm ein eigener Speicherbereich (eine so genannten Partition) reserviert. Leider funktioniert das in der Praxis häufig nicht korrekt – die Windows-Fehlermeldung „Allgemeine Schutzverletzung" ist wohl schon jedem Anwender mehr als einmal begegnet (vgl. Harley, 2002, S. 50 f.).

REDCODE

REDCODE gehört zur Klasse der Assemblersprachen, deren Sprachumfang – wie der Name schon sagt – stark reduziert ist. REDCODE-Programme müssen mindestens eine Anweisung enthalten; länger ist auch das kürzeste Programm nicht, das den bezeichnenden Namen *Knirps* (*Imp* im englischen Original) trägt: MOV 0 1.

An diesem Programm lässt sich die generelle Form einer REDCODE-Anweisung erkennen: Jede Anweisung besteht aus einem Anweisungskürzel – hier MOV (nach dem englischen Wort *move* für bewege) – und zwei Operanden, die mit Operand A und Operand B bezeichnet werden.

Die beiden Operanden geben jeweils eine Adresse einer Speicherzelle an. Die MOV-Anweisung kopiert nun den Inhalt von der ersten Speicherzelle in die zweite. Da Adressen immer relativ zu der Adresse der aktuellen Anweisung angegeben werden, gibt der Operand A die Speicherzelle an, in der die Anweisung selbst steht. Der Operand B = 1 gibt also die direkt nachfolgende Speicherzelle an.

Speicher-adresse	Anzahl ausgeführter Anweisungen		
	0	1	2
N – 1			
N	→ **MOV 0 1**	**MOV 0 1**	**MOV 0 1**
N + 1		→ **MOV 0 1**	**MOV 0 1**
N + 2			→ **MOV 0 1**

Der Pfeil in der Tabelle markiert die Anweisung, die gerade ausgeführt werden soll. Zu Beginn wird die Anweisung MOV 0 1 in die Speicherzelle N + 1 kopiert. Da REDCODE-Anweisungen (wie bei Von-Neumann-Rechnern üblich) nacheinander abgearbeitet werden, wird *Knirps* im nächsten Schritt diejenige Anweisung ausführen, die in der (N + 1)-ten Speicherzelle steht – also (MOV 0 1). Schritt für Schritt schreibt *Knirps* jeweils in eine neue Speicherzelle MOV 0 1.

Somit ist *Knirps* das kürzeste selbstreproduzierende Programm, das je geschrieben wurde. Es kann gegnerische Programme modifizieren, indem es diese ebenfalls zu Knirpsen macht. Allerdings kann es den *Krieg der Kerne* nicht gewinnen, sondern nur ein Unentschieden erreichen. Um zu gewinnen bedarf es der DAT-Anweisung: Man kann sie nicht ausführen, und den Versuch der Ausführung muss ein Kämpfer mit dem „Leben" bezahlen (siehe Kasten, S. 27: Die Mars-Maschine als Von-Neumann-Rechner). Andererseits erfüllt die DAT-Anweisung noch einen weiteren Zweck: Man kann in ihren Operanden Daten ablegen.

Die weiteren REDCODE-Befehle finden sich in Bild 3, nächste Seite. Die beste Einführung zu REDCODE sind immer noch die Artikel von Dewdney (1985). Eine deutschsprachige Anleitung mit Arbeitsbögen und Aufgaben sowie die vom *Spektrum der Wissenschaft* 1993 entwickelte Programmversion, die inzwischen für den Einsatz im Unterricht freigegeben wurde, können im LOG-IN-Service (s. S. 72) oder unter http://bscw.schule.de/pub/bscw.cgi/0/151760 heruntergeladen werden.

Die o. a. Terminologie in den Artikeln zum *Krieg der Kerne* ist ziemlich martialisch. Man kann daher mit Recht problematisieren, ob dies im Informatikunter-

Bild 3: Übersicht der REDCODE-Befehle.

Anweisung	Kürzel	Code	Argumente	Erklärung
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 zu 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 größer	JMG	6	A B	Übergib die Ausführung an Adresse A, falls der Inhalt von B größer als null ist.
Vermindere, springe, wenn null	DJZ	7	A B	Ziehe vom Inhalt der Adresse B 1 ab und übergib die Ausführung an Adresse A, falls der Inhalt von Adresse B dann 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.
Data-Anweisung	DAT	0	B	Nicht ausführbare Anweisung; B ist der Datenwert.

nach Dewdney, 1988b, S. 131

richt pädagogisch vertretbar ist. Andererseits ist es recht kniffflig, *Core-War*-Programme nachzuvollziehen, ganz zu schweigen von der Aufgabe, selber „Kämpfer“ zu entwickeln. Die Beschäftigung mit diesem Spiel ist daher vom Aggressionspotenzial her nicht zu vergleichen mit „Ego-Shootern“. Vielleicht kann man es so sehen, dass sich die Ambivalenz der Informatik, die in ihrer geschichtlichen Entwicklung ganz wesentliche Impulse aus der Anwendung im Militär erhalten hat, auch in diesem Spiel wiederfinden lässt. Interessanterweise gibt es in der Literatur Hinweise, dass der Name REDCODE in Wirklichkeit für „Redstone-Code“ steht, „weil die Simulator-Version an der Redstone Raketenentwicklungs- und -testbasis der US-Streitkräfte entwickelt wurde“ (Harley, 2002, S. 51).

In der ursprünglichen, von Dewdney entwickelten Version waren es erst neun Befehle (Dewdney, 1984). Eine weitere wichtige Anweisung wurde von Dewdney erst nach einigem Zögern zum REDCODE hinzugefügt: die Split-Anweisung SPL (Dewdney, 1985), ohne die kein Kämpfer auskommt, wenn er erfolgreich sein soll. Mit ihr kann sich ein Programm in mehrere Prozesse aufspalten. Wie SPL genau funktioniert, wird im Kasten „Die Mars-Maschine als Von-Neumann-Rechner“ erläutert (nächste Seite).

Das Spiel wurde bald immer beliebter, und man ging dazu über, sogar Weltmeisterschaften auszutragen. Im Jahr 1986 fand in Boston die erste dieser „Programmschlachten“ (Dewdney, 1987) statt, Weltmeister wurde das Programm *Mäuse* (Mice):

```
;       MICE
;       by: Chip Wendell

ptr     DAT     #0      #0
start   MOV     #12     ptr
loop    MOV     @ptr    <copy
        DJN     loop    ptr
        SPL     @copy   0
        ADD     #653    copy
        JMZ     start   ptr
copy    DAT     #0      #833
        END     start
```

Auch hier handelt es sich um ein selbstreproduzierendes Programm. Mit den ersten Zeilen kopiert sich das Programm an eine andere Stelle der Arena und startet dann mit der Split-Anweisung SPL die neue „Maus“, anschließend wird ein neues Kopierziel eingestellt. Wie in der Natur führte die hemmungslose Vermehrung zum Erfolg der „Mäuse“. Darüber hinaus werden bei jedem Kopiervorgang vier „DAT-Bomben“ in die Arena geworfen, sodass mit einigem Glück gegnerische Programme ausgeschaltet werden konnten. Außerdem wird bei Neustart des Programms mit der JMZ-Anweisung geprüft, ob die Zeile mit der Marke ptr noch intakt ist. Falls diese Anweisung vom gegnerischen Programm getroffen wurde, begeht die Maus „Selbstmord“, um die Rechenzeit ihren unversehrten Schwestern zu überlassen: Eine äußerst raffinierte Strategie in nur acht Zeilen Programmtext!

Bald nach der ersten Weltmeisterschaft wurde die ***I***nternational ***C***ore ***W***ars ***S***ociety (ICWS) gegründet. Diese Gesellschaft erarbeitete 1988 einen Standard (ICWS’88), der elf Befehle und vier Adressierungsarten (unmittelbar, direkt, indirekt, prädekrement-indirekt) umfasst. Der Vorschlag für einen neuen REDCODE-Standard (ICWS’94) umfasst etwa 20 Befehle und Erweiterungen, die das Erstellen von komplexeren Kämpfern ermöglichen. ICWS’94 ist eine Obermenge von ICWS’88, sodass Kämpfer, die nach dem alten Standard programmiert wurden, auch unter dem neuen funktionieren. Für den Einsatz im Unterricht ist ICWS’88 völlig ausreichend.

Genauere Informationen zu den aktuellen Entwicklungen sind auf der Seite für *Core War* abrufbar (neuerding als *Core Wars* bezeichnet; http://www.koth.org). Das Wort *Koth* in dieser Adresse steht dabei für „***K***ing ***o***f ***t***he ***H***ill“. Mit „Hügel“ (hill) wird ein (virtueller) Ort für einen permanenten Wettstreit von verschiedenen Kämpfern bezeichnet; Weltmeisterschaften mit der physischen Präsenz der Programmierer finden seit vielen Jahren nicht mehr statt. Die Kämpfer, die über elektronische Post an den Betreiber des jeweiligen Hügels geschickt werden, müssen dann gegen die anderen Kämpfer antreten. Wer am schlechtesten abschneidet, muss den Kampfhügel verlassen, der Beste ist dann natürlich der „König des Hügels“. Es gibt dabei auch Hü-

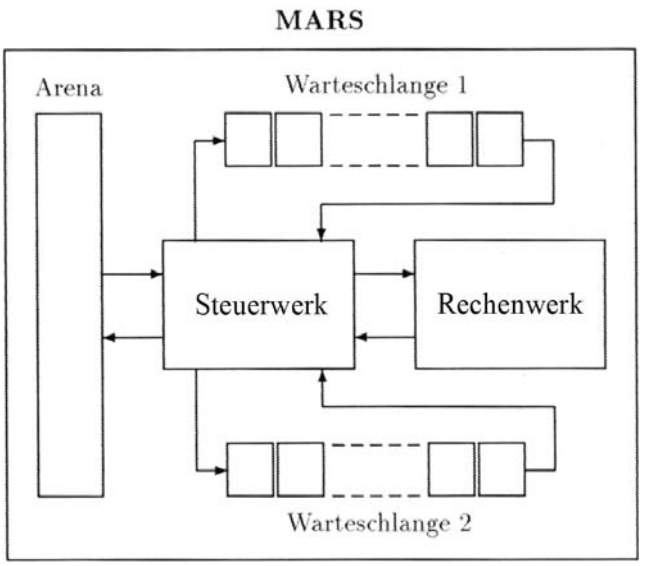
Die Mars-Maschine als Von-Neumann-Rechner

Es ist sinnvoll, sich als Schauplatz des *Kriegs der Kerne* einen eigenen, recht primitiven Computer namens *Mars* vorzustellen. Dieser ist aus rein praktischen Gründen (wer will sich schon eigens für dieses Spiel einen Computer zusammenlöten?) in den umgebenden Rechner eingebettet, indem ein Programm innerhalb des Wirtsrechners die Wirkung seiner Komponenten imitiert.

Ein Blick in die Struktur dieses virtuellen Computers ist schon aus dem Grund interessant, weil er – mit einer Variante – das Prinzip des klassischen, 1946 von Arthur W. Burks, Herman H. Goldstine und John von Neumann vorgeschlagenen Rechners in Reinform verwirklicht. Die meisten heutigen – nicht-parallelen – Computer arbeiten nach diesem Prinzip, obwohl der Benutzer sich dank einer Fülle von Anwendungsprogrammen in der Regel darüber keine Gedanken mehr machen muss. Allerdings kann ein Von-Neumann-Rechner in der Urform nur jeweils ein Programm abarbeiten. *Mars* ist insofern etwas moderner, als er zwei Prozesse zu steuern vermag, die ihrerseits mit der SPL-Anweisung noch Kindprozesse starten können.

Der *Mars*-Rechner besteht aus einem Speicher (der *Arena*), in dem sich die Programme mit ihren Daten befinden, zwei Warteschlangen, in denen die Prozesse der beiden Kämpfer verwaltet werden, einem Rechenwerk, das die arithmetischen und logischen Operationen einzelner Anweisungen ausführt, und schließlich einem Steuerwerk, das die jeweils nächste auszuführende Anweisung bereitstellt, eventuell vorverarbeitet (abhängig von der Adressierungsart) und an das Rechenwerk zur Ausführung übergibt.

Die Warteschlangen vcon *Mars* bestehen aus den Adressen von Anweisungen, die zur Ausführung vorgemerkt werden.



Der Aufbau von *Mars* mit Arena, Steuerwerk und Rechenwerk sowie zwei Warteschlangen.

Das Zusammenwirken der Komponenten von *Mars* ist einer näheren Betrachtung wert. Normalerweise bringt *Mars* die Kämpfer zu Beginn an zufällige Positionen in der Arena, wobei ein vorgegebener Minimalabstand einzuhalten ist. Jede der beiden Warteschlangen enthält am Anfang die Speicheradresse, mit der die Ausführung des zugehörigen Kämpfers begonnen werden soll. Nicht von den Kämpfern belegte Speicherzellen erhalten zu Beginn den Inhalt DAT #0 #0. Durch Zufallsauswahl wird festgelegt, welcher Kämpfer beginnt.

Wir nehmen an, dies sei Kämpfer Nr. 2. Dann entfernt das Steuerwerk aus der Warteschlange 2 die erste Adresse und kopiert die an dieser Adresse im Speicher liegende REDCODE-Anweisung. In der Kopie ersetzt sie zunächst irgendwelche in der Anweisung vorhandenen indirekten Adressierungen durch die effektiven Adressen. Die so bereinigte Form der Anweisung lautet nun z. B. MOV (2002) (2001), wobei die Klammern die effektiven Adressen kennzeichnen. Die bereinigte MOV-Anweisung wird an das Rechenwerk zur Ausführung übergeben. Die Ausführung einer Anweisung verändert stets die Arena oder die betreffende Warteschlange oder beides.

Nach Ausführung der ersten Anweisung von Kämpfer 2 ist Kämpfer l an der Reihe. Abgesehen davon, dass für ihn die Warteschlange 1 zuständig ist, ist der Ablauf völlig analog dem für Kämpfer Nr. 2 beschriebenen. So wird stets abwechselnd für beide Kämpfer die Adresse der auszuführenden Anweisung der entsprechenden Warteschlange entnommen und die Adresse der nächsten Anweisung an die Warteschlange angehängt. Was ist die nächste Anweisung? Im Normalfall nach dem Von-Neumann-Prinzip die mit der nächsthöheren Adresse. Wenn jedoch bei einer Sprunganweisung die Sprungbedingung erfüllt ist, kommt die Adresse des Sprungziels in die Warteschlange.

Eine weitere Ausnahme bilden die Anweisungen SPL und DAT. Die Anweisung SPL hat ja den Sinn, einen weiteren Prozess zu eröffnen. Also wird zuerst die in der SPL-Anweisung angegebene Adresse und sofort danach die Adresse der auf die SPL-Anweisung folgenden Speicherzelle an die Warteschlange angehängt. Da vorher die Adresse der Anweisung selbst aus der Warteschlange entfernt wurde, hat sich die Länge der Warteschlange insgesamt um eins erhöht: Ein neuer Prozess ist geboren. Im Gegensatz dazu bewirkt die Ausführung der DAT-Anweisung nur das Entfernen der eigenen Adresse, aber keinerlei neuen Eintrag in die Warteschlange: Der Prozess hat ein Leben verloren. Stößt das Steuerwerk auf eine leere Warteschlange, so hat der zugehörige Kämpfer sein letztes Leben ausgehaucht und damit den Kampf verloren.

Aufgabe: Informieren Sie sich z. B. im *Informatik-Duden* über den Von-Neumann-Rechner (VNR). Beschreiben Sie in einer Übersicht Gemeinsamkeiten und Unterschiede zwischen dem VNR und dem Steuerprogramm *Mars* für die „Kampfarena“ aus dem Programmsystem *Core Wars*.

nach Robitzsch, 1993, S. 83–86

gel für Anfänger, bei denen die Kämpfer nach einer bestimmten Anzahl von Kämpfen den Platz für die Neuankömmlinge frei machen müssen. Der Programmtext der erfolgreichen Kämpfer wird normalerweise nicht veröffentlicht, solange sie noch auf einem Hügel aktiv sind, auf der *Koth*-Seite gibt es aber umfangreiche Archive mit alten Programmen.

Evolution im Computer: *Venus, Tierra* und *Avida*

Angeregt von Dewdneys Spiel, gab es seit Ende der Achtzigerjahre mehrere Versuche, die Evolution im Computer nachzubilden. Einen der ersten unternahm der dänische Physiker Steen Rasmussen, der aus dem *Krieg der Kerne* eine *Welt der Kerne* (Core World) machte. Dabei wurde *Mars* zu *Venus,* das zum einen das Akronym für ***V***irtual ***E***volution in a ***N***onstochastic ***U***niverse ***S***imulator (Künstliche Evolution in einem nichtstochastischen, universellen Simulator) war, zum anderen in Anlehnung an „die römische Göttin der Fruchtbarkeit, Liebe und Schönheit, in der Hoffnung, interessante, lebensähnliche Fähigkeiten schaffen zu können“ (zitiert nach Levy, 1996, S. 175).

Die „Organismen“ von *Venus* waren dabei aus REDCODE-Anweisungen zusammengesetzt, die gleichzeitig als „Gene“ aufgefasst wurden. Am Anfang wurden die 8000 Zellen des Kerns mit einer „Ursuppe“ von Anweisungen gefüllt und dann in Zwölf-Stunden-Läufen auf einem IBM PS-80 PC über ungefähr einhunderttausend „Generationen“ getestet. Rasmussen ging dabei von einem völlig offenen Ansatz aus: Das System sollte seine Entwicklungsrichtung selbst bestimmen.

Dieser Ansatz brachte die Schwierigkeit mit sich, dass Rasmussen am Anfang überhaupt nicht wusste, wonach er suchen sollte. Nach vielen Versuchen fasste er die Ergebnisse so zusammen: „Trotz der Fehleranfälligkeit der individuellen Anweisungen und der bescheidenen Größe des Kerns ist unser System durchaus in der Lage, stabile, zusammenwirkende Strukturen zu entwickeln“ (zitiert nach Levy, 1996, S. 179). Einer der erfolgreichsten „Organismen“, der während der Venus-Durchläufe entstand, war eine Struktur, die MOV-SPL genannt wurde, weil diese beiden REDCODE-Anweisungen offenbar essenziell für die Reproduktion waren. Für diejenigen, die sich ein wenig mit den erfolgreichen Kämpfern im Krieg der Kerne befasst haben, ist dieses Ergebnis nicht überraschend.

Kurz nach *Venus* entwickelte der Biologe Thomas Ray das berühmte Programm *Tierra* (Spanisch für „Erde“; vgl. Schwill/Witten, 2004, in diesem Heft, S. 13 ff., sowie z. B. Kinnebrock, 1996, S. 50 ff. oder Levy, 1996, S. 269 ff.), das ebenfalls auf einer assemblerartigen Sprache beruht, die sich aber in einigen wesentlichen Punkten von REDCODE unterscheidet. Wie bei *Venus* gilt bei *Tierra*, dass der Genotyp mit dem Phänotyp übereinstimmt: Sowohl die „Gene“ als auch die „Organismen“ bestehen aus Anweisungen in einer speziellen Assemblersprache.

Worin unterscheiden sich die Assemblerbefehle von *Tierra* und *Venus*/REDCODE? Die REDCODE-Programme hatten sich als zu anfällig gegenüber Mutationen erwiesen; die Selbstreplikationsfähigkeit war zu „brüchig“. Der Grund hierfür ist die Größe des Befehlssatzes: Obwohl REDCODE in der von Dewdney entwickelten Fassung nur zehn Befehle umfasst, liegt die wahre Größe des Befehlssatzes bei 10^{11} (eine Dezimalstelle zur Codierung des Befehls, jeweils fünf Dezimalstellen für die Operanden A und B, wobei in der ersten Stelle die Adressierungsart und in den weiteren vier Stellen die Adresse verschlüsselt wird; vgl. Dewdney, 1984).

Die Befehle von *Tierra* ähneln der i860-Maschinensprache, aber die wahre Größe des Befehlssatzes ist nur 32 (5 Bit), die Befehle haben keine Operanden. Der Clou ist dabei die Template-(Schablonen-)basierte Adressierung durch die Markerbefehle NOP_0 und NOP_1 (NOP steht für ***n***o ***op***eration). Auf einen JMP-(Sprung-)Befehl folgen in der Regel vier NOP-Befehle. Bei der Ausführung wird das nächste komplementäre Template (0 und 1 vertauscht) gesucht und der Befehlszeiger dorthin gelenkt. Ein weiterer Unterschied zu REDCODE ist, dass jeder „Organismus“ innerhalb von *Tierra* über Register und Kellerspeicher (stack) verfügt. Durch den kleinen Befehlsumfang wurde es möglich, durch „Evolution“ stabile „Organismen“ zu erhalten (vgl. Kurth, 2003).

Ein weiteres bekanntes, von REDCODE und *Tierra* inspiriertes KL-Programm ist *Avida*. Hier ist der Befehlssatz noch kleiner (26 Anweisungen), dafür hat die „Zentraleinheit“ von *Avida* drei Register und bis zu zwei Kellerspeicher. Neben der Selbstreplikation sollen die Programme bestimmte Aufgaben erfüllen, die bei Erfolg durch die Vergabe von „Bonuspunkten“ (= mehr Rechenzeit) belohnt werden. Erfolgreiche Programme können sich also schneller vermehren. Typischerweise bestehen die Aufgaben darin, logische Verknüpfungen durchzuführen. Es zeigen sich bei den Experimenten erstaunliche Parallelen zur biologischen Evolution. Pöppe führt dazu aus (2003, S. 94):

> „Noch kann man gegen das ganze Experiment der amerikanischen Informatiker die üblichen Einwände erheben: Es ist unklar, ob die Ergebnisse etwas über das echte Leben aussagen oder nur ein Artefakt der radikalen Vereinfachung sind. Vielleicht bildet das Computermodell eine zentrale Eigenschaft der echten Evolution nicht richtig ab, und wenn es nachgebessert wird, ist auf einmal alles ganz anders. Aber schon jetzt eignet sich die Welt *Avida*, Theorien über die Evolution zu überprüfen – vorausgesetzt, die Theorie ist so allgemein, dass auch diese künstliche Welt darunter fällt. Jedermann ist eingeladen, eine Kopie von *Avida* aus dem Internet zu laden und damit seine eigene Welt zu betreiben. Mit weiteren überraschenden Ergebnissen ist zu rechnen.“

Hinweise zum Unterrichtseinsatz

Wie am Anfang erwähnt, kann der *Krieg der Kerne* im Informatikunterricht zur Behandlung der maschinennahen Programmierung eingesetzt werden, die

Mars-Maschine übernimmt dabei die Rolle des Von-Neumann-Rechners. Als Programmsystem kann dabei das Spiel eingesetzt werden, das 1993 vom *Spektrum der Wissenschaft* vertrieben wurde – es läuft auch in der Eingabeaufforderung von WindowsXP, und mit den entsprechenden Emulatoren vermutlich auch auf Linux- und Mac-Rechnern, da es sehr sauber programmiert ist. Das Programm ist eine vollständige Entwicklungsumgebung; die Kämpfe können am Bildschirm verfolgt werden.

Im Internet finden sich für alle gängigen Betriebssysteme noch zahlreiche weitere Versionen von Programmsystemen für *Core Wars*, diese halten sich aber z. T. nicht an den Standard. Die „offizielle" Version kann von der *Koth*-Seite bezogen werden, dort können auch die Original-Artikel von Dewdney heruntergeladen werden. Im Unterricht hat sich ebenfalls das Programmsystem *CoreWin* von Chip Wendell, dem Autor des Weltmeister-Programms von 1986, bewährt.

Ein anspruchsvolles Projekt im Informatikunterricht wird von Wolfgang Ambros beschrieben: Die Schüler programmierten eine eigene Umgebung für den Krieg der Kerne (Ambros, 1992). Dies war an der TU Braunschweig über mehrere Jahre die Aufgabe für das Software-Praktikum; in der Schule wird man dergleichen nur mit besonders leistungsstarken Gruppen realisieren können.

Die grundsätzliche Funktionsweise von Viren und Würmern wird den Lernenden durch die Beschäftigung mit *Core Wars* deutlich. Aufgrund der zahlreichen Informationen im Internet können sich die Schülerinnen und Schüler selbstständig mit den weiteren Details beschäftigen; ein guter Ausgangspunkt ist die Seite des *Bundesamtes für Sicherheit in der Informationstechnik* (BSI, 2004).

Für die Erarbeitung der Programme zum künstlichen Leben, die auf den *Krieg der Kerne* zurückgehen, eignen sich neben den üblichen, in der Regel etwas langweiligen Schülerreferaten auch andere Unterrichtsmethoden wie etwa das *Stationenlernen* oder das so genannte *Gruppenpuzzle* mit der Arbeit in Stamm- und Expertengruppen.

Der Text zu den *Risiken und Nebenwirkungen* des künstlichen Lebens enthält Anregungen zur kritischen Beleuchtung der Fragen im Zusammenhang mit dem KL (Grimmer/König, 2002), eine kurze Zusammenfassung findet sich im letzten Kapitel der KL-Vorlesung von Winfried Kurth (Kurth, 2003). Mit diesen Materialien kann eine abschließende, kontroverse Diskussion vorbereitet werden.

StD Helmut Witten
Fachseminar für Informatik
Walther-Rathenau-Oberschule (Gymnasium)
Herbertstraße 4
14193 Berlin

E-Mail: helmut@witten-berlin.de

Über den **LOG-IN-Service** (s. S. 72) erhalten Sie das DOS-Programmsystem Core War vom Spektrum der Wissenschaft aus dem Jahr 1993 (nur zu Bildungszwecken kostenfrei einsetzbar) sowie eine PowerPoint-Präsentation zum „Krieg der Kerne" (A. K. Dewdney's Core War), eine Liste mit Internetquellen und ein Handbuch zum „Krieg der Kerne", 88 Seiten, als PDF-Datei.

Literatur und Internetquellen

Ambros, W.: Das Informatikprojekt als fachtypische Arbeitsmethode. In: LOG IN, 12. Jg. (1992), H. 5/6, S. 28–32.

Avida-Homepage: http://dllab.caltech.edu/avida/ [Stand: August 2004]

Brunnstein, K.: Mit IT-Risiken umgehen lernen – Über Probleme der Beherrschbarkeit komplexer Informatiksysteme. In: Keil-Slawik, R.; Magenheim, J. (Hrsg.): Informatikunterricht und Medienbildung. Tagungsband der INFOS 2001 – Reihe „Lecture Notex in Informatics", Bd. P-8. Bonn: Köllen Verlag, 2001.

BSI (Bundesamt für Sicherheit in der Informationstechnik): Ins Internet – mit Sicherheit. Bonn, 2004:
http://www.bsi-fuer-buerger.de/ [Stand: August 2004]

Core Wars – King Of The Hill (korth): http://www.koth.org/ [Stand: August 2004]

Core Wars Homepage (2001): http://www.corewars.org/ [Stand: August 2004]

Dewdney, A. K.: Krieg der Kerne. In: Spektrum der Wissenschaft (Hrsg.), 1988a, S. 124–129. Erstabdruck: Spektrum der Wissenschaft, August 1984.

Dewdney, A. K.: Computer-Viren. In: Spektrum der Wissenschaft (Hrsg.), 1988b, S. 130–134. Erstabdruck: (als: Krieg der Kerne 2): Spektrum der Wissenschaft, Mai 1985.

Dewdney, A. K.: Die große Programmschlacht (Hrsg). In: Spektrum der Wissenschaft, 1988c, S. 135–188. Erstabdruck: (als: Krieg der Kerne 3): Spektrum der Wissenschaft, April 1987.

Dewdney, A. K.: Vorsicht, infektiös! In: Spektrum der Wissenschaft (Hrsg.), 1992, S. 124–128. Erstabdruck: Spektum der Wissenschaft, Juni 1989.

Grimmer, C.; König, P.: Artifical Life – Risiken und Nebenwirkungen. Hausarbeit zur Veranstaltung „Informatik und Gesellschaft", WS 2001/02, Universität Bremen:
http://www.informatik.uni-bremen.de/~pkoenig/ALifeOB.pdf [Stand: August 2004]

Harley, D.; Slade, R.; Gattiker, U. E.: Das Anti-Viren Buch. Bonn: mitp, 2002.

Kinnebrock, W.: Künstliches Leben – Anspruch und Wirklichkeit. München; Wien: Oldenbourg, 1996.

Kurth, W.: Artificial Life. Aus der Vorlesung WS 2002/03 an der TU Cottbus (2003):
http://www-gs.informatik.tu-cottbus.de/~wwwgs/al_home.htm [Stand: August 2004]

Levy, S.: Künstliches Leben aus dem Computer. München: Knaur, 1996.

Pöppe, C.: Die Evolution der Computerwürmer. In: Spektrum der Wissenschaft, 26. Jg. (2003), H. 9, 92–94, sowie unter
http://www.wissenschaft-online.de/spektrum/index
.php?action=rubrik_detail&artikel_id=6812 [Stand: August 2004]

Robitzsch, H.: Neues vom Krieg der Kerne. In: Spektrum der Wissenschaft, 16. Jg. (1993), H. 1, S. 80–88, sowie unter
http://www.wissenschaft-online.de/spektrum/index
.php?action=rubrik_detail&artikel_id=519 [Stand: August 2004]

Schwill, A.; Witten, H.: Künstliches Leben – Ein Überblick. In: LOG IN, 24. Jg. (2004), H. 130, S. 8–14, *in diesem Heft*.

Spektrum der Wissenschaft (Hrsg.): Computer-Kurzweil. Heidelberg: Spektrum Akademischer Verlag – Verständliche Forschung, 1988 .

Spektrum der Wissenschaft (Hrsg.): Computer-Kurzweil 2. Heidelberg: Spektrum Akademischer Verlag – Verständliche Forschung, 1992.

Spektrum der Wissenschaft (Hrsg.): Core War – Krieg der Kerne. Begleitheft zur Diskette, Heidelberg: Spektrum der Wissenschaft Verlagsgesellschaft, 1993.

Tierra-Homepage: http://www.his.atr.jp/~ray/tierra/index.html
[Stand: August 2004]

Wendell, C.: CoreWin-Homepage. http://www.geocities.com/corewin2/ [Stand: August 2004]