Informationen für die Lehrperson zu „Corewars“

Das Thema dieser Unterrichtseinheit liegt in der Schnittmenge mehrerer Gebiete der Informatik: Assembler-Programmierung, Rechnerarchitektur und Artificial Life. Letzteres ist ein Teilgebiet der künstlichen Intelligenz, das sich dem Studium einer besonderen Klasse von Algorithmen widmet. Diese Algorithmen zeigen in ihrer Ausführung Verhalten, die in ähnlicher Form bei biologischen und physikalischen Prozessen in der Natur beobachtbar sind. Ein bekanntes Beispiel dafür ist Conways "Game of Life" (Berlekamp et al., 1982).

Die StudentInnen analysieren gegebene Programme und entwickeln evtl. eigene Variationen, die sie auf einer virtuellen Maschine gegeneinander antreten lassen. Die Programmierumgebung „Corewars“ bietet dazu eine einfache Assemblersprache und die Visualisierung des Speichers der virtuellen Maschine. Das Laden und Ausführen der Programme geschieht komfortabel über ein grafisches Benutzerinterface.

Erforderliches Vorwissen

Assembler

Die StudentInnen kennen den grundsätzlichen Aufbau einer Assemblersprache: Labels, Opcodes und Möglichkeiten der Speicher-adressierung. Sie sind in der Lage, ein einfaches Corewars-Programm mithilfe der Sprachspezifikation selbständig zu verstehen.

Rechnerarchitektur

Die StudentInnen kennen einfache Speichermodelle, den Unterschied zwischen absoluter und relativer Adressierung und wissen, wie einfache Datentypen (z.B. Integer) im Speicher abgelegt werden.

Hilfreiche Techniken

Methodisches Vorgehen bei Experimenten

Das Schema „These - Vorgehen - Resultate - Interpretation“ bietet Hilfestellung, wie ein Experiment selbständig durchgeführt werden kann:

- *These:* Die Aussage, die bekräftigt oder widerlegt werden soll, wird präzise formuliert.
- *Vorgehen:* Die einzelnen Schritte des Experiments werden geplant, um die effiziente Durchführung zu ermöglichen.
- *Resultate:* Der Ausgang des Experimentes wird protokolliert, z.B. in Tabellenform.
- *Interpretation:* Die Resultate werden auf ihre Stichhaltigkeit überprüft. Gegebenenfalls wird das Experiment modifiziert und wiederholt.

Metakognition
Während der Phase des entdeckenden Lernens sind die Schüler aufgefordert, ihre Gedankengänge schriftlich festzuhalten und zu reflektieren.
Sind die StudentInnen darin noch unerfahren, ist die Methode der „Gedanken-Reisetagebücher“ (Ruf und Gallin, 1991) eine Möglichkeit, wie diese Praxis vorab im Unterricht eingeübt werden kann.

Ablauf

15min *Einführung in Corewars*
Die Lehrperson gibt eine Einführung in die Corewars-Umgebung, wie man Programme lädt, ausführt und Speicherdiagramme interpretiert. Weiter gibt sie die Kriterien für die Leistungsbeurteilung bekannt.

30min *Corewars kennenlernen*
Die StudentInnen lesen ihr Handout durch. Sie laden drei einfache Programme („first.cw“, „mover.cw“ und „mover2.cw“) und analysieren ihr Verhalten durch Studium des Codes und Simulation in Corewars. Bei technischen Schwierigkeiten und Verständnisfragen kann die Lehrperson in diesem Teil Unterstützung leisten. Das Tutorial von Walter Hofmann ergänzt die Ausführungen der Lehrperson und dient als Referenz.

120min *Entdeckendes Lernen*
Die StudentInnen analysieren selbständig weitere Programme und suchen nach interessanten Duell-Kombinationen. Sie können eigene Varianten entwickeln, diese gegeneinander antreten lassen und machen sich Gedanken über den Aufbau und mögliche Limitierungen von Corewars.

45min *Vorträge und Turnier*
StudentInnen, die ein eigenes Programm geschrieben haben, stellen den Code und ihre Überlegungen in einem Kurzvortrag vor. Festgehaltene Gedanken, Experimentier-Protokolle und Schlussfolgerungen werden (als Hausarbeit) in einen Kurzaufsatz zusammengefasst und der Lehrperson abgegeben.
Als Abschluss treten die programmierten Kandidaten in einem Turnier gegeneinander an. Die StudentInnen bestimmen gemeinsam das Programm mit der originellsten Idee, dem schönsten Code oder der besten Leistung im Turnier.

Erklärungen zum Programm / Tutorial

Eine ausführliche Erklärung der Architektur, Sprache und Bedienung von Corewars ist im Tutorial von Walter Hofmann enthalten, das Teil dieser Lerneinheit ist. Es dient gleichzeitig als Sprachreferenz für die StudentInnen.

Simulation möglicher Ergebnisse

1. Wie verhalten sich die Algorithmen?

Die Visualisierung des Speichers mit den Statistiken zur Programmausführung erlaubt eine detaillierte Analyse des Algorithmenverhaltens. Das Protokollieren der eigenen Experimente führt zu vielfältigen Gedankengängen auf unterschiedlichen Ebenen. Einige mögliche Erkenntnisse sind im folgenden ausgeführt:

a) Makroskopisches Verhalten als Folge mikroskopischer Instruktionen

Mögliche Beobachtungen der StudentInnen:

- Programm X verhält sich anders, wenn ich es gegen Y oder Z laufen lasse.
- Ich habe den Quellcode eigentlich verstanden, aber in der Simulation macht das Programm etwas unerwartetes.

Erkenntnis:

Die Mehrzahl der Programme in Corewars ist relativ kurz, d.h. in der Grössenordnung von einigen Dutzend Zeilen. Mit etwas Erfahrung lässt sich das Verhalten eines einzelnen Programms relativ genau vorhersagen.

In der Ausführung gegen feindliche Programme wird das Verhalten jedoch chaotisch, d.h. kleine Änderungen der Anfangsbedingungen haben grosse Auswirkungen auf das Verhalten des einzelnen Programms und des Systems als Ganzes.

Dies ist ein wichtiges Resultat der Simulation komplexer Systeme. Obwohl das Verhalten der einzelnen Prozesse bekannt ist, lassen sich keine genauen Vorhersagen für das Verhalten des Gesamtsystems machen. Diese Denkweise liegt einem ganzen Gebiet der Mathematik zugrunde, der Chaos-Theorie.

b) Beschreibungen auf höherer Ebene

Mögliche Beobachtungen der StudentInnen:

- Wenn ich diese Programme laufen lasse entsteht folgendes Muster: ... Beide Programme sind an dessen Entstehen beteiligt.
- Programm A scheint zu warten, bis B C besiegt hat.

Erkenntnis:

Durch das Beobachten des Speichers sind spontane Beschreibungen wie „diese zwei Prozesse kämpfen gemeinsam gegen den Dritten“ möglich, obwohl sich dieses Verhalten im Code nicht finden lässt.

Dies steht im Gegensatz zu rein zufälligen Prozessen, die zwar ebenfalls nicht vorhersagbar sind, aber auch keine nennenswerte Struktur aufweisen. Der Fachbegriff dafür ist „emergente Phänomene“.

c) Interessante Kombinationen

Nicht jede Paarung von Programmen erzeugt interessante Resultate. Oft dominiert ein Programm seine Konkurrenten klar. Dennoch kommt es vor, dass Programme mit ganz unterschiedlichen Strategien beinahe gleich stark sind.

d) Termination

Mögliche Beobachtungen der StudentInnen:

- Das Duell zwischen X und Y ist nach kurzer Zeit vorbei, bei X und Z ist dafür kein Ende abzusehen.
- Je nach Startposition im Speicher dauert ein Duell kürzer oder länger.

Erkenntnis:
In Corewars kann sich die Laufzeit der Ausführung durch verschiedene Anfangsbedingungen drastisch ändern. Es gibt Programme, die gegen gewisse Gegner nie terminieren werden, gegen andere Gegner schon.

Die Termination eines Algorithmus ist eine zentrale Eigenschaft. Ein Kernsatz der theoretischen Informatik besagt, dass es algorithmisch nicht möglich ist zu entscheiden, ob ein Programm terminiert oder nicht (das sog. „Halteproblem“).

2. Analyse des Programmcodes

Auch das direkte Studium des Programmcodes bietet Anstösse für eigene Gedankengänge. Fundamentale Konzepte wie Laufzeit und Komplexität spielen für ein kompetitives Programm eine wichtige Rolle.

Die Analyse bestehender Programme ist auch Ausgangspunkt für eigene Entwicklungen.

a) Komplexität und Laufzeit

Mögliche Beobachtungen der StudentInnen:

- Corewars führt der Reihe nach je eine Instruktion jedes Programms aus.
- Ein kurzes Programm ist zwar schnell, aber dafür nicht besonders raffiniert.

Erkenntnis:
Kürzere Programme werden vergleichsweise öfters ausgeführt als lange. Andererseits haben längere Programme eine potentiell höhere Komplexität, und damit ist ihre Gewinnstrategie möglicherweise erfolgreicher.

Schnelligkeit und Mächtigkeit sind wünschenswerte Eigenschaften, die miteinander konkurrenzieren. Die optimale Gewichtung kann von Fall zu Fall verschieden sein.

b) Minimum Description Length

Neben offensichtlich ineffizienten Berechnungen und redundanten Befehlen gibt es Optimierungen, die eine kreative Leistung benötigen. Ein mögliches Beispiel sind diese zwei Programme:

```
first:    move second,fourth
second:   move first,third
third:    data 0
fourth:   data 0

first:    move first,second
second:   data 0
```

Beide Programme verfolgen dieselbe Idee: Sie kopieren sich selbst an die nächstmögliche Stelle im Speicher. Das zweite Programm benutzt die Idee, dass sich ein Befehl selbst referenzieren kann, und spart damit die Hälfte der Instruktionen.

Mehr Instruktionen sind nicht gleichbedeutend mit mehr Komplexität. Die Länge der kürzestmöglichen Beschreibung eines Algorithmus ist ein mögliches Mass für seine Komplexität, bekannt unter dem Fachbegriff „Minimum Description Length“ oder „Kolmogorov-Komplexität“, benannt nach ihrem Begründer.

3. Entwickeln eigener Programme

Das Schreiben eigener Programme fordert die Fähigkeiten zu Analyse, Abstraktion, Synthese und Kreativität. Die überschaubare Anzahl Befehle und das einfache Rechnermodell helfen, dass sich erste Erfolge schnell einstellen und motivieren. Andererseits ist die Komplexität von Corewars gross genug, dass eine Vielzahl von Ansätzen erfolgsversprechend ist.

Das folgende Programm liegt im Bereich dessen, was StudentInnen in der gegebenen Zeit erreichen können. Es verfolgt die Strategie, noch nicht dominierte Speicherzellen so schnell wie möglich zu besetzen – ganz nach dem Motto „Angriff ist die beste Verteidigung“. Die Speicherstellen werden zufällig ausgewählt, um das eigene Verhalten zu randomisieren.

```
L:  info 2, A   # Schreibe zufällige Speicheradresse nach A
    own [A]     # Falls die von A referenzierte Stelle schon
    jump L      # dem Programm gehört: zurück zu L,
    move 0, [A] # sonst besetze sie.
    jump L      # Zurück zum Anfang
A:  data 0
```

Dieses Programm schlägt das schon erwähnte „mover2.cw“ in den meisten Fällen.

a) Gewinnstrategie

Mögliche Beobachtungen der StudentInnen:

- Dieses Programm hat zwar eine gute Offensive, aber kann sich nicht verteidigen.
- Bei mehreren Kotrahenten kann es sich lohnen abzuwarten.
- Dieses Programm schien gegen alle anderen zu gewinnen, doch jetzt verlor es gegen ein sonst schwaches Programm.

Erkenntnis:
Sowohl aktive wie passive, aggressive wie defensive Strategien können je nach Paarung erfolgsversprechend sein. Bei mehr als einem Gegner kann es sich auch lohnen, mit einem der beiden zu kooperieren.

Mittlerweile sind zahlreiche Gewinnstrategien in der Literatur bekannt (Lindahl, 1994). Die Problematik der gegenseitig abhängigen Gewinnstrategie ist auch bekannt unter dem Namen „Prisoner's Dilemma“ (Dawkins, 1989): Zwei Verbrecher werden getrennt verhört. Wenn beide schweigen, kommen sie mit einer mittelschweren Strafe davon. Verpfeift ein Verbrecher den anderen und dieser schweigt, bekommt der erste eine milde Strafe, der zweite aber wird schwerer verurteilt. Verpfeifen sie sich gegenseitig, erhalten beide die schwere Strafe (siehe Tabelle 1 für eine Illustration).

	Verbrecher 1 kooperiert	**Verbrecher 1 verpfeift 2**
Verbrecher 2 kooperiert	Mittlere Strafe für beide	Milde Strafe für 1, schwere Strafe für 2
Verbrecher 2 verpfeift 1	Milde Strafe für 2, schwere Strafe für 1	Schwere Strafe für beide

Tabelle 1. „Prisoner's Dilemma“: Der Ausgang der eigenen Strategie hängt vom Verhalten des Anderen ab.

b) Minimale Komplexität

Es ist möglich, mit einem einzigen Befehl ein funktionierendes Programm zu schreiben. Auch die meisten Beispielprogramme sind kurz, in der Regel weniger als zehn Zeilen lang. Dennoch zeigen sie verschiedene Verhaltensweisen. Eine einfache Idee lässt sich später ausbauen, indem man sukzessiv Spezialfälle behandelt.

Die schrittweise Verfeinerung einer einfachen Grundidee, und damit die stufenweise Erhöhung der Komplexität, ist ein bekanntes Vorgehen in der Informatik. Auch in der Biologie wird zur Zeit das minimale Genom eines funktionierenden Organismus diskutiert.

4. Mögliche Erweiterungen von Corewars

Beim Schreiben eigener Programme stossen die StudentInnen möglicherweise auf Strategien, die sich mit den verfügbaren Befehlen nicht effizient umsetzen lassen. Das Einführen neuer Befehle könnte dem Abhilfe schaffen.

Auch die grundlegende Architektur könnte erweitert werden. So unterscheidet Corewars zum Beispiel zwischen Daten und Befehlen in Speicherzellen. Die direkte Manipulation von Befehlen mittels mathematischer Operationen könnte eine Vielzahl neuer Verhaltensweisen ermöglichen.

Was wird von den StudentInnen erwartet?

StudentInnen, die ein eigenes Programm geschrieben haben, präsentieren den Code sowie dessen Idee und Entstehung. Die anderen Studenten sind eingeladen, das Programm zu diskutieren oder Verständnisfragen zu stellen. Die Präsentatoren müssen in der Lage sein, die Funktionsweise ihres Programms zu erläutern.

StudentInnen, die bestehende Programme, ihre Strategien und ihr Verhalten studiert haben, schreiben einen Kurzaufsatz aufgrund ihrer schriftlich festgehaltenen Gedankengänge. Der Aufsatz soll zwei wesentliche Erkenntnisse mit den dazu führenden Gedankengängen und Protokollen ihrer Experimente enthalten.

Literatur

Berlekamp E., Conway J., and Guy, R.: Winning Ways (for your Mathematical Plays), Volume 2. London 1982 (Academic Press).

Dawkins, R.: The Selfish Gene. Oxford, 1989, 2. Auflage (Oxford University Press).

Gallin, P., Ruf, U.: Aufbau von Sprach- und Fachkompetenz beim Lernen mit Kernideen und Reisetagebüchern. In: Schweizer Schule. 78 (1991) 18-29.

Lindahl, G.: Modern Core Wars. 1994. Zu finden unter
http://www.pbm.com/~lindahl/pbem_articles/corewars