Kara: Ein theoriebasierter Ansatz für Lernumgebungen zu fundamentalen Konzepten der Informatik

Markus Brändle, Werner Hartmann, Jürg Nievergelt, Raimond Reichert, Tobias Schlatter

Technology & Education
Swiss Federal Institute of Technology
ETH Zentrum, 8092 Zürich, Switzerland
(braendle | hartmann | nievergelt | reichert) @ inf.ethz.ch

Abstract: In der heutigen Informationsgesellschaft sind Informatikkenntnisse und die Beherrschung von Informations- und Kommunikationstechnologien wichtige Schlüsselqualifikationen. Zentrale Themen der Informatik sind die Formalisierung sowie Daten, Algorithmen und Programme. Das Unterrichten dieser Konzepte als Teil der Allgemeinbildung ist anspruchsvoll, und die heutigen Ansätze sind oft nicht zufriedenstellend. Benötigt werden einfache, intuitive, qualitativ hochstehende Lernumgebungen als Unterstützung für Lehrerinnen und Schüler. Die theoretische Informatik bietet verschiedene Modelle, die sich ausgezeichnet als Basis für solche Lernumgebungen eignen. Die von uns entwickelten Kara -Lernumgebungen basieren auf endlichen Automaten und umfassen: Eine Umgebung zur Einführung in die Programmierung (Kara); einen Ansatz zum Unterrichten der fundamentalen Konzepte der Concurrency (MultiKara); eine Einführung in die Lehre der Berechenbarkeit basierend auf zweidimensionalen Turing Maschinen (TuringKara); einen reibungslosen Übergang zum Programmieren mit Java (JavaKara). Die Kara-Lernumgebungen werden an in vielen Schulen auf unterschiedlichen Stufen eingesetzt. Nachfolgend stellen wir die einzelnen Umgebungen vor, im speziellen MultiKara und TuringKara.

1 Einleitung

In der heutigen Informationsgesellschaft spielt die Beherrschung von Informations- und Kommunikationstechnologien eine immer grössere Rolle. Diesem Umstand muss die schulische Grundausbildung Rechnung tragen. Im Zentrum sollten langlebige Konzepte der Informatik allgemein bildender Natur stehen. Unter anderem sollen Schülerinnen und Schüler ein Verständnis für Algorithmen entwickeln. Dieser Standpunkt wird ausführlich vorgestellt in [HN02].

Um ein Verständnis für Algorithmen zu entwickeln, müssen Schülerinnen die Gelegenheit haben, selber Algorithmen in konkrete Programme umzusetzen und so eigene Programmiererfahrung zu sammeln. Ein solcher Einstieg ins Programmieren sollte den Schülern so vermittelt werden, dass sie sich auf das Wesentliche konzentrieren können. Die Schüler müssen mit wenig Aufwand in kurzer Zeit erste lauffähige Programme erstellen können. Professionelle Programmiersprachen (Java, C, Ada) eignen sich nur beschränkt, da sie

wegen ihrer Komplexität viel Zeit zum Erlernen brauchen. Zudem sind typische Entwicklungsumgebungen zu komplex für den Einsatz im Unterricht. Es reicht auch nicht, dass die Schüler in solchen Umgebungen ein rein textbasiertes „Hello World“-Programm schreiben lernen. Vielmehr müssen die Lernumgebungen den Anforderungen der „Nintendo-Generation“ entsprechen [GS02]. Heutige Schülerinnen und Studenten sind mit multimedialen Anwendungen aufgewachsen und erwarten entsprechende Umgebungen.

Ein Verständnis für Algorithmen kann basierend auf grundlegenden Theorien der Informatik vermittelt werden, womit die Langlebigkeit der Lerninhalte garantiert ist. Zudem können intuitive, graphische Lernumgebungen die Schülerinnen beim Lernen unterstützen. Die von uns erstellten Kara-Lernumgebungen verfolgen diesen Ansatz und behandeln verschiedene fundamentale Aspekte der Informatik. Alle Umgebungen enthalten begleitende Materialien wie Aufgaben, Lösungen und Bedienungsanleitungen. Sie können auf diversen Plattformen (Windows, Mac OS X, Linux, Unix) eingesetzt werden und sind frei verfügbar: www.educeth.ch/informatik/karatojava.

2 Kara: Lernumgebungen für Programmierneulinge

Die Kara-Umgebung (Abbildung 1) vermittelt erste Schritte des Programmierens mit Hilfe von endlichen Automaten (siehe [RNH00]). Endliche Automaten sind ein fundamentales Konzept der Informatik und eignen sich gut als „Programmiersprache für Anfänger“. Sie sind konzeptionell einfach verständlich und können anhand von Alltagsgeräten wie zum Beispiel Uhren, Getränkeautomaten oder Videogeräten anschaulich erklärt werden.

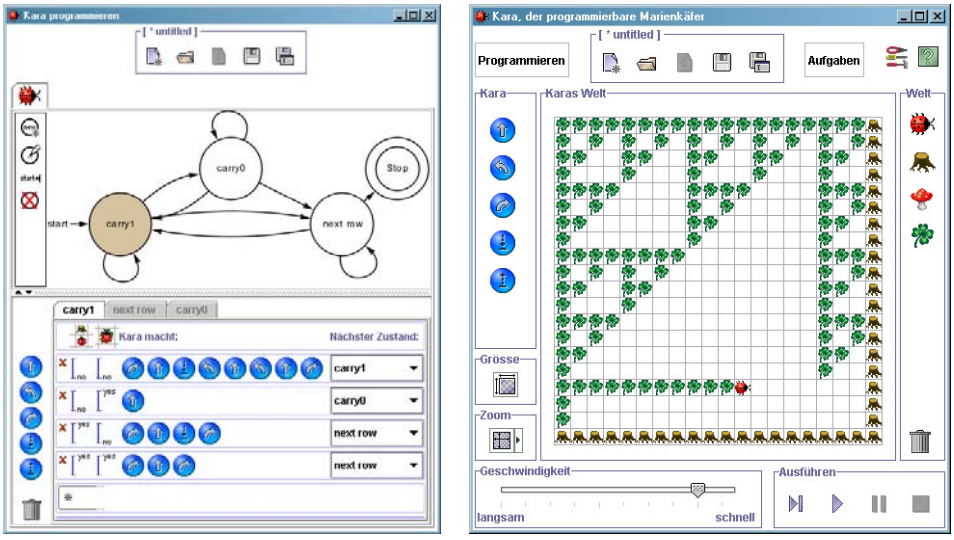

Abbildung 1: Die Kara Umgebung

Die Lernumgebung erlaubt es, den Marienkäfer Kara in seiner Welt zu steuern und mit ihm Aufgaben zu lösen. Obwohl die Welt mit nur vier verschiedenen Arten von Objekten (Kara, Kleeblätter, Pilze, Bäume) sehr einfach gehalten ist, lassen sich auch anspruchsvolle Probleme lösen, wie zum Beispiel das Zeichnen des Pascal-Dreiecks modulo 2 (Abb. 1).

3 MultiKara: Einführung in nebenläufige Programmierung

In den letzten 30 Jahren hat die nebenläufige Programmierung stark an Bedeutung gewonnen. Graphische Benutzeroberflächen und die damit verbundenen ereignisorientierten Programme machen die Nebenläufigkeit zu einer Notwendigkeit. Das Schreiben eines nebenläufigen Programms unterscheidet sich fundamental vom Schreiben eines sequentiellen Programms. Die grösste Herausforderung ist der Nichtdeterminismus, der durch die verschiedenen Prozesse und deren unvorhersehbare Ausführungsreihenfolge hervorgerufen wird. Nebenläufigkeit erfordert deshalb eine andere Denkweise beim Programmieren. Ben-Ari und Kolikant [BAK99] argumentieren, dass nebenläufiges und verteiltes Programmieren bereits auf „high school level“ unterrichtet werden sollte: „the challenging nature of the subject ensures that students must learn how to use critical thinking rather than hacking“.

MultiKara trägt diesen Punkten Rechnung und bietet eine Lernumgebung für den Einstieg in die nebenläufige Programmierung (Abbildung 2). Grundlegende Konzepte wie Nichtdeterminismus, Scheduling und Synchronisation können anschaulich vermittelt werden.

MultiKara ist eine Erweiterung von Kara auf vier Marienkäfer. Jeder Käfer hat einen eigenen Automaten, der unabhängig von den anderen ausgeführt wird. Der Benutzer kann die Prioritäten der einzelnen Käfer für das Scheduling einstellen. Die Wahl des Marienkäfers, der als nächster einen Zustands-Übergang ausführen darf, geschieht probabilistisch, in Abhängigkeit der Prioritäten der Käfer.

MultiKara bietet vier Concurrency-Mechanismen. Mit Hilfe der zwei inklusiven Mechanismen können verschiedene Prozesse synchronisiert werden. Die zwei exklusiven Mechanismen erlauben den gegenseitigen Ausschluss. Beide Konzepte können sowohl im Zustandsraum des Programms wie auch in dem Datenraum der Welt eingesetzt werden:

	inclusive	*exclusive*
world space	**meeting room**	**monitor**
state space	**barrier**	**critical section**

Von den vier Mechanismen sind *Monitor*, *Barrier* und *Critical Section* Standard-Mechanismen der nebenläufigen Programmierung. Der *Meeting Room* ist ein neues Konzept von MultiKara und erlaubt die zeitliche Synchronisation in der Welt der Marienkäfer.

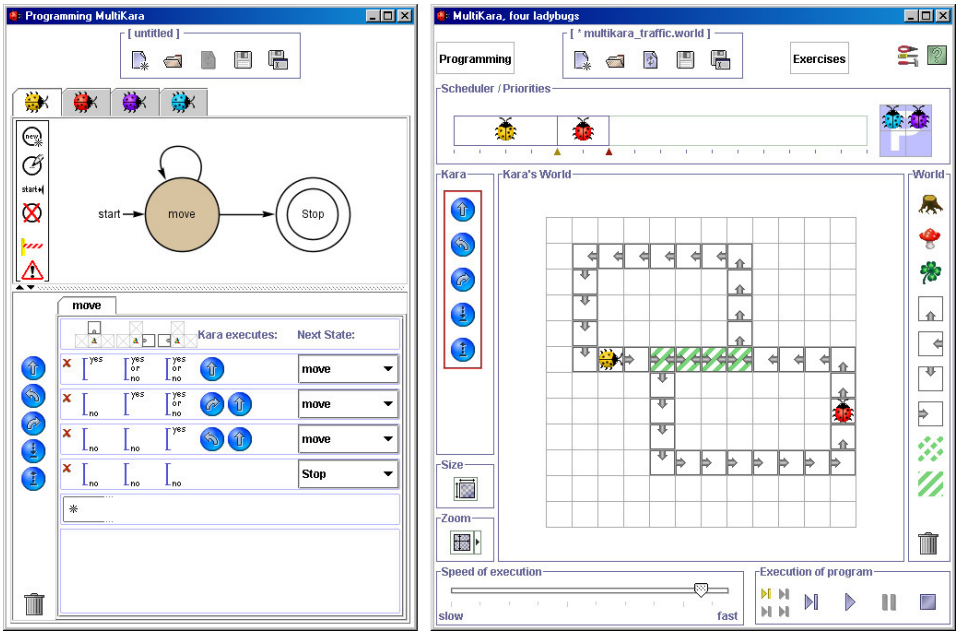

Abbildung 2: Die MultiKara-Umgebung

3.1 Verkehrssimulation: Ein Beispiel für gegenseitigen Auschluss im Datenraum

Monitore sind ein Mechanismus für den gegenseitigen Ausschluss und helfen sicher stellen, dass immer höchstens ein Prozess einen kritischen Abschnitt bearbeitet. In der Abbildung 2 wird die Aufgabe einer Verkehrssimulation gezeigt. Jeder Kara in der Welt befindet sich auf einem Strassenstück und soll diesem endlos folgen. Die Schwierigkeit dieser Aufgabe besteht im Vermeiden von Kollisionen beim Überqueren von Kreuzungen und beim Durchlaufen von bidirektionalen Wegstücken. Die gezeigte Lösung nutzt Monitorfelder um die kritischen Strassenfelder zu markieren (schraffierte Felder in Abbildung 2). Alle Monitorfelder sind Teil eines globalen Monitors. Dadurch wird garantiert, dass sich maximal ein Marienkäfer auf einem Monitorfeld befindet. Versucht ein zweiter Käfer einen mit Monitoren markierten Abschnitt zu betreten wird er schlafen gelegt, bis der Monitor wieder frei gegeben wird. Der Zustandsautomat muss den Marienkäfer lediglich endlos auf der Strasse führen und sich nicht um den gegenseitigen Ausschluss kümmern, da dies implizit in der Welt geschieht.

3.2 Füllen eines Rechtecks: Ein Beispiel für Synchronisation im Zustandsraum

Barriers sind ein Synchronisations-Mechanismus und erlauben es, Prozesse in verschiedene Phasen zu unterteilen und eine neue Phase erst dann zu beginnen, wenn alle Prozesse die vorhergehende abgeschlossen haben. Der *Barrier*-Mechanismus ist ein Spezialfall des „Rendez-vous“-Mechanismus mit dem Unterschied, dass beliebig viele Prozesse aufeinander warten und keine Daten ausgetauscht werden.

Abbildung 3 zeigt vier Karas innerhalb eines durch Bäume begrenzten Rechtecks. Die Marienkäfer sollen das Rechteck vollständig mit Kleeblättern füllen. Die Startposition der Karas, ihre Blickrichtung und Prioritäten sind beliebig. Die Grösse des Rechtecks muss mindestens 2x2 sein.

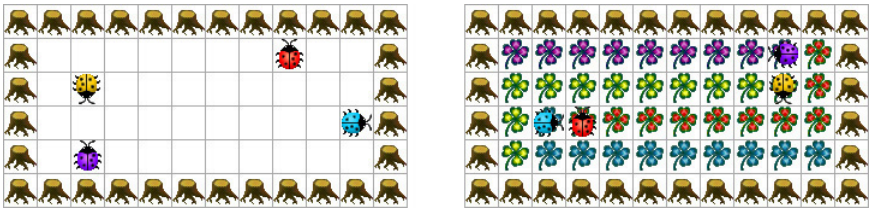

Abbildung 3: Füllen eines Rechtecks, Aufgabenstellung (rechts) und Lösung (links)

Ein möglicher Ansatz besteht darin, das Rechteck von aussen spiralförmig zu füllen. In einer ersten Phase müssen sich die Marienkäfer in die Ecken bewegen (Phase 1 in Abbildung 4). In der zweiten Phase füllen die Karas jeweils eine Kante der Spirale (Zustand „laufe“). Der Zustand „in Ecke“ ist als *Barrier* markiert und ist für die Synchronisation zuständig. Die zweite Phase wird wiederholt, bis das gesamte Rechteck gefüllt ist.

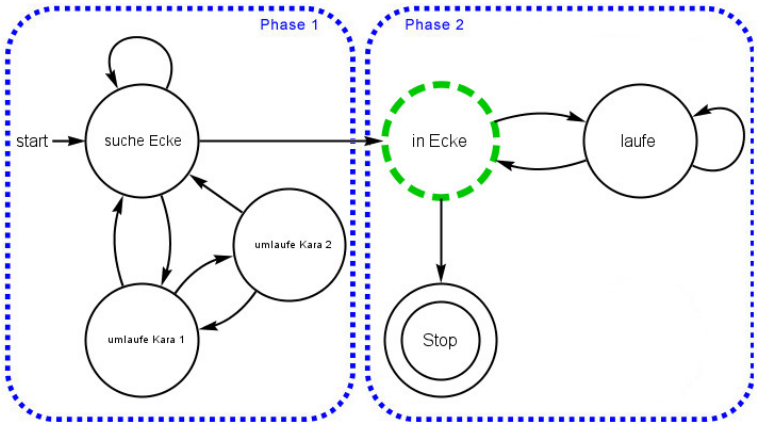

Abbildung 4: Programm, um Rechteck zu füllen

4 TuringKara: Zweidimensionale Turing-Maschinen

Die theoretische Informatik ist ein wichtiger Teil jeder Informatikausbildung. Universelle Berechnungsmodelle spielen dabei eine zentrale Rolle. Sie entstanden aus der Frage „Was ist algorithmisch berechenbar und was nicht? “. Für Anfänger ist es oftmals überraschend, dass universelle Berechnungsmodelle mit sehr einfachen Grundoperationen auskommen, solange sie zwei Eigenschaften besitzen: unbeschränkte Zeit und unbeschränkten Speicher.

Turing-Maschinen sind ein universelles Berechnungsmodell. Schülerinnen und Schüler, die bereits mit Kara vertraut sind, können den Übergang zu Turing-Maschinen einfach vollziehen. Die Kontrolllogik ist die gleiche wie bei endlichen Automaten. Der Unterschied liegt im externen Speicher: Turing-Maschinen arbeiten auf ein einem unbeschränkten Speichermedium. Typischerweise wird ein eindimensionales Band verwendet. Im Gegensatz dazu verwendet TuringKara eine zweidimensionale Welt. Aus Sicht der theoretischen Berechenbarkeit bringt ein zweidimensionales Speichermedium keine Vorteile. Es hat jedoch Vorteile in didaktischer Hinsicht: Das Lösen von Aufgaben wird vereinfacht, da die Anzahl der Bewegungen des Lese-/Schreibkopfes stark reduziert werden kann. Interessant ist, wie Turing aus der Sicht der Berechenbarkeit argumentierte, dass ein eindimensionales Band für Turing-Maschinen ausreichend sei [Tu37]:

> Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child’s arithmetic book. In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think that it will be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper, i.e., on a tape divided into squares.

Die Turing-Maschinen in TuringKara weichen aus didaktischen Gründen in einem weiteren Punkt von Standard-Turing-Maschinen ab: Sie lassen mehrere Schreib- und Bewegungsbefehle pro Übergang zu. Dadurch kann die Anzahl der Zustände einer Turing-Maschine stark reduziert werden.

4.1 Die TuringKara-Umgebung

TuringKara verwendet einen Lese-/Schreibkopf, der in alle vier Himmelsrichtungen bewegt werden kann. Die Felder der Welt können folgenden Symbole aufnehmen: 0, 1, #, □ (leeres Feld) sowie die vier Pfeilsymbole $\leftarrow, \rightarrow, \uparrow, \downarrow$. Die Symbole haben keine semantische Bedeutung und können beliebig verwendet werden. Die Pfeile können zum Beispiel als Markierungen in der Welt eingesetzt werden.

Abbildung 5 zeigt die TuringKara-Umgebung. In der dargestellten Aufgabe soll jedes Feld eines durch # begrenzten Labyrinths besucht und mit einer 0 markiert werden. Die Pfeile dienen als Markierung, aus welcher Richtung ein Feld betreten wurde.

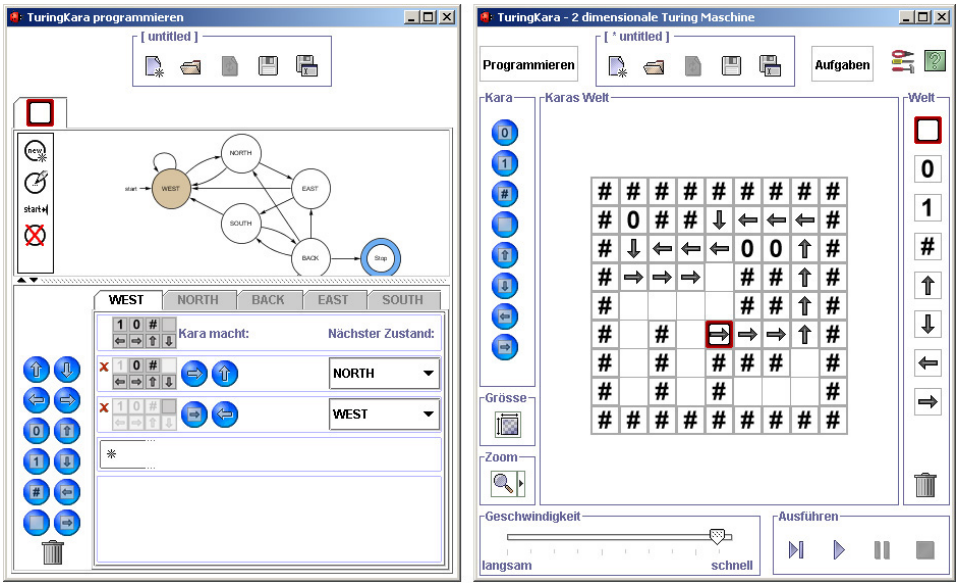

Abbildung 5: Die TuringKara Umgebung

4.2 Beispiel: Binäre Multiplikation

Um zwei Zahlen zu multiplizieren, haben wir in TuringKara die so genannte Zigeuner-Multiplikation implementiert. Bei der Zigeuner-Multiplikation wird die erste Zahl jeweils durch zwei dividiert, während die zweite verdoppelt wird. Ist der erste Multiplikand ungerade, so wird der zweite zum Zwischenresultat addiert. Dies wird wiederholt bis der erste Multiplikand eins ist. Das Halbieren und das Verdoppeln einer binären Zahl kann einfach mit einem Shift nach rechts und einem Shift nach links erreicht werden.

5 * 4		1	0	1	#	1	0	0
0		0	0	0	0	0	0	0
2 * 8		1	0	#	1	0	0	0
4		0	0	0	0	1	0	0
1 * 16		1	#	1	0	0	0	0
4		0	0	0	0	1	0	0
20		0	0	1	0	1	0	0

Abbildung 6: Multiplikation von $5 * 4$ in TuringKara

Abbildung 6 zeigt den Ablauf der Multiplikation in der Welt. Die beiden Multiplikanden stehen auf der ersten Zeile getrennt durch ein #, das aktuelle Resultat steht auf der zweiten Zeile. Die Division und Multiplikation der Multiplikanden findet ausschliesslich auf der oberen Zeile statt. Dieser Algorithmus ist dank der zweidimensionalen Welt mit Turing-Kara relativ einfach mit nur neun Zuständen umsetzbar.

4.3 Die universelle Turing-Maschine veranschaulicht in TurigKara

Eine universelle Turing-Maschine (UTM) nimmt als Eingabe die Beschreibung einer anderen Turing-Maschine (Programm und Band) und simuliert deren Ausführung. Wir haben in TuringKara eine UTM mit 41 Zuständen implementiert. Sie kann Turing-Maschinen simulieren, die ein eindimensionales Band mit den Symbolen 0, 1, #, □ benutzen. Obwohl die UTM selber komplex ist, ist ihr Ablauf dank der zweidimensionalen Welt intuitiv und visuell einfach nachvollziehbar. Auf diese Art kann mit TuringKara Studierenden ein Gefühl für das Konzept der UTM vermittelt werden. Abbildung 7 zeigt als Beispiel die Codierung einer einfachen Turing-Maschine für die UTM.

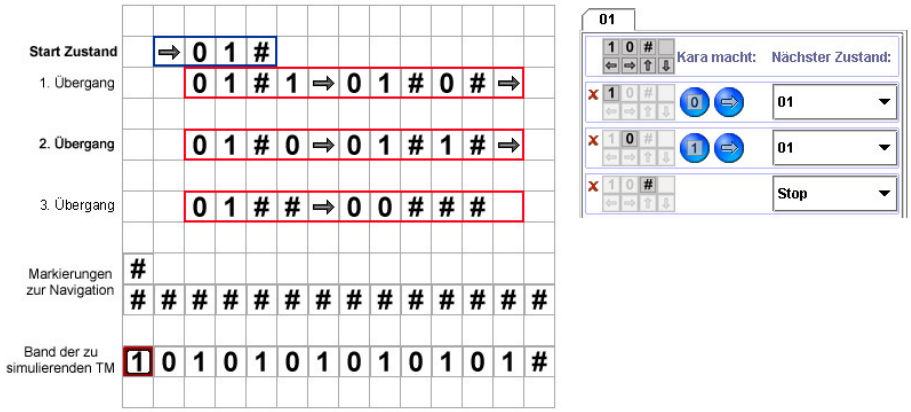

Abbildung 7: Zu simulierende Maschine (rechts); Kodierung für die TuringKara-UTM (links)

5 JavaKara: Ein reibungsloser Übergang zu Java

JavaKara überbrückt die Kluft zwischen den Programmierumgebungen mit endlichen Automaten und der „Realität“. JavaKara benutzt die gleiche Welt wie Kara, verlangt jedoch das Schreiben der Programme in Java. JavaKara bietet drei Vorteile:

1. Der Benutzer bleibt in einer ihm bekannten, einfachen Umgebung und kann sich auf das Erlernen von Java konzentrieren.

2. Jedes JavaKara-Programm produziert eine visuelle Ausgabe. So kann der Programmablauf graphisch verfolgt werden.

3. Das Verwenden von Java setzt den Benutzern keine Schranken, da sie auf die kompletten Java-Bibliotheken zurückgreifen können.

JavaKara erlaubt es, den Marienkäfer mittels einfacher Befehle zu kontrollieren (Abbildung 8). Fortgeschrittene können direkt auf die Welt als zweidimensionalen Array zugreifen. So können auch algorithmisch anspruchsvolle Aufgaben gelöst werden, zum Beispiel das Zeichnen eines „schwarz-weiss“ Bildes der Mandelbrotmenge oder das Suchen des kürzesten Weges aus einem Labyrinth.

```
public class Spiral extends JavaKaraProgram {

  void walk (int distance) {
    for (int i = 0; i < distance; i++) {
      kara.putLeaf();
      kara.move();
    }
  }

  public void myProgram() {
    final int MAX_LENGTH = 20;
    int d = 1;

    while (d < MAX_LENGTH) {
      walk (d);
      kara.turnRight();
      d ++;
    }
  }
}
```

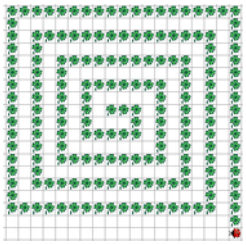

Abbildung 8: Zeichnen einer Kleeblatt-Spirale

6 Fazit

Die Kara-Umgebungen vermitteln einfach und intuitiv verschiedene grundlegende Konzepte der Informatik. Die Umgebungen sind aufeinander abgestimmt, können aber auch unabhängig voneinander eingesetzt werden. Alle Umgebungen basieren auf fundamentalen Ideen der Informatik und garantieren so eine Langlebigkeit der vermittelten Lerninhalte. Das Zielpublikum ist breit gefächert und reicht von Informatik-Neulingen bis hin zu Informatik-Studenten. Entsprechend werden die Umgebungen an vielen Schulen eingesetzt.

Literatur

[BAK99] Ben-Ari, M. und Kolikant, Y. B.-D.: Thinking parallel: the process of learning concurrency. In: *Proceedings of the 4th annual SIGCSE/SIGCUE on Innovation and technology in computer science education*. S. 13–16. ACM Press. 1999.

[GS02] Guzdial, M. und Soloway, E.: Teaching the nintendo generation to program. *Communications of the ACM*. 45(4):17–21. 2002.

[HN02] Hartmann, W. und Nievergelt, J.: Informatik und Bildung zwischen Wandel und Beständigkeit. *Informatik-Spektrum*. 25(6):465–476. December 2002.

[RNH00] Reichert, R., Nievergelt, J., und Hartmann, W.: Ein spielerischer Einstieg in die Programmierung mit Java. *Informatik-Spektrum*. 23(5). October 2000.

[Tu37] Turing, A. M.: On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*. Series 2(42):230–265. 1936–1937.