Dr. Klaus Manhart

Schildkrötenwelten

Multi-Agenten-Simulation mit NetLogo und StarLogo

Eine Heerschar aus Molekülen, Ameisen, Menschen oder ganzen Staaten lenken? Mit NetLogo und StarLogo stehen leistungsfähige Multi-Agenten-Systeme zur Verfügung, um komplexe dynamische Systeme zu modellieren und zu erkunden.

Als Seymour Papert in den sechziger Jahren die Programmiersprache Logo erfand, hatte er Großes vor [1]. Der amerikanische Psychologe und Mathematiker wollte nicht nur eine einfach bedienbare neue Sprache kreieren, sondern vor allem Kindern und Schülern den Computer auf eine kreative und spielerische Weise nahe bringen.

Das Markenzeichen von Logo war die Turtle, ein schildkrötenförmiger Cursor, der nach dem Starten des Logo-Interpreters im Zentrum des Bildschirms steht. Der Befehl „vorwärts 100“ jagte die Schildkröte hundert Pixel in Blickrichtung voran, gleichzeitig hinterließ sie auf dem Bildschirm eine leuchtende Spur. Das Kommando „rechts 90“ wendete ihren Blick um 90 Grad nach rechts – jetzt noch „wiederhole 4“ und ein Quadrat war geboren. Schrieb man dies alles der Schildkröte mit dem Schlüsselwort `PR Quadrat` ins Gedächtnis, so verstand sie nun den neuen Befehl: `Quadrat`. Aus einfachen Wörtern wuchs so allmählich ein komplexes Vokabular von Prozeduren. Später war Logo sogar noch ein kleiner kommerzieller Erfolg beschieden, denn es wurde von Lego entdeckt – auf Paperts Ideen beruht das Lego-Produkt „Mindstorms“, das Kindern das spielerische Programmieren kleiner Roboter ermöglicht [2].

Inzwischen wirkt Logo etwas altbacken, doch die Sprache ist nicht tot. Mit NetLogo und StarLogo stehen Logo-Varianten im neuen, modernen Gewand zur Verfügung mit denen sich einiges anstellen lässt: Auf der Grundlage von Paperts Sprache entwickelte das Massachusetts Institute of Technology (MIT) ein leicht handhabbares Multi-Agenten-Simulationssystem, das sich wunderbar für die Modellierung und Erkundung von Komplexitäts- und Selbstorganisationsphänomenen eignet. Bisher musste man dazu weitaus schwerfälligere Simulations-Tools wie Swarm, SDML oder Repast beherrschen [3].

Die „Starlogo“ genannte Sprache lief allerdings zunächst nur auf Macintosh-Rechnern – 1999 legte daher das „Center for Connected Learning“ von der Northwestern University Evanston eine plattformunabhängige und um einiges erweiterte in Java implementierte Variante namens „NetLogo“ vor [4], die seit September 2003 bei Version 1.3 angekommen ist – noch bis Ende des Jahres soll NetLogo 2.0 herauskommen. Das MIT-Team zog im Februar 2000 mit einer eigenen Java-Variante nach und hat mittlerweile StarLogo 2.0.2 im Angebot, das sich in puncto Leistungsfähigkeit mit NetLogo durchaus messen kann.

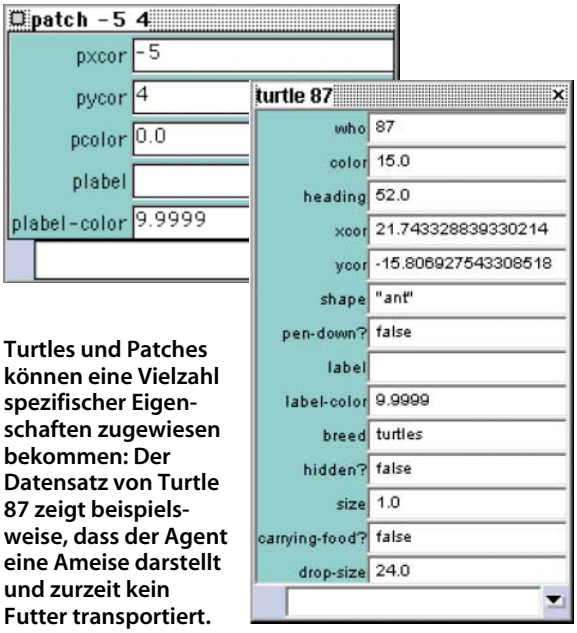
Von Turtles zu Agenten

Ihre neue Ausrichtung verdanken die neuen Logo-Varianten einer eigentlich recht simplen Änderung im Grunddesign der Sprache: Statt einer einsamen Schildkröte instruiert man nun mehrere Turtles – oder „Agenten“, wie sie allgemein genannt werden. Jeder dieser Agenten ist eine selbstständige Software-Einheit, die individuell ihr Programm abarbeitet. Wenn Agent A eine Handlung X verrichtet, kann gleichzeitig und unabhängig davon Agent B die Handlung Y ausführen – die Software verwaltet für jede Turtle einen persönlichen Datensatz. Die Turtles können außerdem mit anderen Turtles oder mit der Umgebung kommunizieren.

Diese Umgebung, in der die Agenten leben, beschränkt sich auf zwei Dimensionen und besteht aus einem schachbrettartigen Tablett mit Zellen, den so genannten „Patches“, die sich bei Bedarf ebenfalls programmieren und mit Informationen versehen lassen. Die Größe des Tabletts sowie die Anzahl und Eigenschaften der Agenten wird vom Modellierer oder „Observer“ festgelegt – wie er in der neuen Terminologie genannt wird.

Eigentlich ist, wie Michael Resnick in seinem Buch „Turtles, Termites, and Traffic Jams“ schreibt, StarLogo kein Simulationssystem im engeren Sinne, denn die Software kann viele Phänomene nur verhältnismäßig grob abbilden. Das System bietet aber – wie auch NetLogo – die Möglichkeit, das Zusammenwirken vieler Einzelteile in dezentralisierten Systemen spielerisch zu untersuchen und so deren wesentliche Funktionsprinzipien zu begreifen.

Und Agenten können in NetLogo und StarLogo vieles verkörpern – angefangen bei Atomen, Molekülen und Zellen über höhere Lebewesen und menschliche Individuen bis hin zu Gruppen oder gar Staaten lässt sich alles in das Agenten-Konzept pressen. Bei näherem Hinsehen kann man erkennen,



Turtles und Patches können eine Vielzahl spezifischer Eigenschaften zugewiesen bekommen: Der Datensatz von Turtle 87 zeigt beispielsweise, dass der Agent eine Ameise darstellt und zurzeit kein Futter transportiert.

dass es sich bei NetLogo um nichts anderes als einen zellulären Automaten handelt – einer regelmäßigen Anordnung vieler Zellen, deren Zustände in Abhängigkeit von den Zuständen der Nachbarzellen in diskreten Zeitschritten nach einer Vorschrift neu festgelegt werden. Bei der Bestimmung des zukünftigen Zustands einer Zelle werden meistens die aktuellen Zustände der Nachbarzellen berücksichtigt, manchmal geht dabei auch noch die entferntere Vergangenheit ein.

Spielplatz

Sowohl NetLogo als auch StarLogo präsentieren sich als übersichtliche und leicht bedienbare Java-Applikationen: Ganz oben unter „Controls“ befindet sich das Control-Panel mit den vier Reitern „Interface“, „Information“, „Procedures“ und „Errors“. Über die „Interface Toolbar“ kann man die Modelle laufen lassen, sie beobachten und verändern.

Auf dem schachbrettartigen Tablett rechts läuft die Simulation ab. Daneben kann der Programmierer diverse Anzeige- und Bedienelemente – Knöpfe und Schieberegler – platzieren. Im unteren „Chart“-Bereich kann man Ergebnisse der Simulation – beispielsweise die Größe der Populationen von Schafen und Wölfen – laufend als Kurve ausgeben lassen.

Die drei weiteren Reiter „Information“, „Procedures“ und „Errors“ enthalten Panels für die Programmdokumentation, einen Editor für die Eingabe des Programmcodes sowie Fehlermeldungen. Rechts unten befindet sich schließlich noch das „Command Center“. Am Prompt kann man direkt NetLogo-Befehle an Agenten eingeben. Bei StarLogo gibt es getrennte Kommandozentralen für den „Observer“ und für die „Turtles“.

Für einen Überblick über die Möglichkeiten von NetLogo lohnt es sich, in der umfangreichen Model Library zu stöbern. Sie wird zusammen mit der Software geliefert und ist Teil der Simulationsumgebung. Alle Modelle beinhalten den NetLogo-Programmcode und eine Dokumentation. Bedient werden fast alle wissenschaftlichen Richtungen: von Physik und Chemie über Biologie, Mathematik und Informatik bis hin zu den Sozialwissenschaften. Die Bibliothek von StarLogo bietet nicht ganz so viel Auswahl, hat aber dafür ebenfalls recht interessante Programme im Angebot.

Erste Übung

Die erste Fingerübung ist in NetLogo formuliert. Die grundlegenden Befehle sind nahezu identisch mit denen von StarLogo – die konzeptionellen Unterschiede sind in einem Kasten erklärt. Das Programm Random Walk ist eine simple Übung, um die elementaren Sprachelemente kennen zu lernen. Wer tiefer einsteigen will, findet umfangreiche Manuals und Tutorials auf den Projekt-Webseiten.

```
to setup
  ca
  crt 3
  ask turtles [ set color yellow ]
end

to do-random-walk
  ask turtle 0 [
    rt random 360
    forward 1
    stamp blue
    wait 0.1
    ]
  ask turtle 1 [
    rt random 360
    forward 1
    stamp green
    wait 0.1
    ]
  ask turtle 2 [
    rt random 360
    forward 1
    stamp red
    wait 0.1
    ]
end
```

Random Walk besteht aus den zwei Prozeduren `setup` und `do-random-walk`. Prozeduren beginnen in NetLogo immer mit dem Schlüsselwort `to`, gefolgt vom Prozedurnamen. Das `end` schließt die Prozedur ab. Die Prozedur `to setup` in der ersten Zeile dient der Initialisierung:

Agenten, Schwärme und Emergenz

Nicht erst seit Michael Crichtons Science-Fiction-Roman „Beute“ faszinieren Insektenschwärme Wissenschaftler und Techniker ebenso wie ein – sich gelegentlich wohlig gruselndes – Publikum.

In einem selbstorganisierten System sind die einzelnen Einheiten in der Regel sehr simpel gestrickt. Aus den zahlreichen Interaktionen dieser simplen Komponenten auf einer tiefen Ebene entstehen jedoch komplexe und sehr anpassungsfähige Verhaltensmuster. Das komplexe Verhalten eines Gesamtsystems bedeutet aber in der Regel nicht, dass die einzelnen Komponenten des Systems über komplexe Regeln miteinander wechselwirken. Fast scheint es, als besäßen solche Systeme eine eigene Art von „Intelligenz“, die sich auf geheimnisvollem Wege aus dem Nichts manifestiert.

Natürlich gibt es Erklärungsansätze: Bereits 1959 führte der französische Biologe Pierre-Paul Grassé erstmals zur Erklärung des Nestbaus bei Termiten den Begriff der „Stigmergy“ ein. Die einzelnen Termiten verändern demnach durch ihre Bautätigkeit im Laufe der Zeit ihre Umgebung. Der äußere Reiz der veränderten Umgebung führt schließlich dazu, dass die Termiten von einem Satz von Verhaltensweisen zu einem anderen Satz „umschalten“. Mit Hilfe von Multi-Agenten-Simulationen versuchen Wissenschaftler solche „Emergenzphänomene“ nachzubilden. Die mittlerweile entwickelten Theorien sind bestechend, aber bisher gibt es nur in wenigen Fällen technische Anwendungen. (wst)

`ca` (clear all) löscht alle Turtles vom Spielfeld, setzt alle globalen Variablen auf Null und weist allen Patch-Variablen ihren Default-Wert zu.

`crt x` (create turtles) erzeugt eine Anzahl von x Turtles, in diesem Fall drei im Mittelpunkt des Spielfeldes an der Koordinate 0,0. `ask turtles [ set color yellow ]` weist allen auf dem Spielfeld befindlichen Agenten den Farbwert gelb zu.

Mit der Anweisung `ask` können entweder alle Turtles angesprochen werden (wie hier) oder nur ganz bestimmte (zum Beispiel `ask turtle 4`). In jedem Fall folgt dahinter in eckigen Klammern eine Liste mit Anweisungen für die angesprochenen Turtles. In dem Beispiel wird also die Anweisung `set color yellow` für alle Turtles ausgeführt.

Die Prozedur `do-random-walk` stößt die eigentliche Aktion an. Der erste ask-Befehl gibt Turtle 0 folgende Anweisung: `rt random 360`: Drehe dich um eine zufällige Zahl von Graden, die zwischen 0 und 360 liegt nach rechts. `forward 1`: Bewege dich um eine Zelle vorwärts. `stamp red`: Ändere die Farbe des Patches, auf dem du stehst, in rot. `wait 0.1`: Warte 0,1 Sekunden. Die beiden anderen Turtles erhalten entsprechende Anweisungen, wobei Turtle 1 jede besetzte Zelle grün färbt und Turtle 2 rot.

Nach Eingabe des Codes im Prozedurfenster bringt man noch Initialisierungsbutton und Startbutton an, indem man die passenden Elemente aus der Toolbar auf die gewünschte Position zieht und sie mit dem gewünschten Prozedurnamen verknüpft.

Gruppenbildung

Sich zufällig bewegende Turtles sind inhaltlich nicht besonders ergiebig. Wesentlich faszinierender an Multi-Agenten-Systemen ist, dass sie das Zusammenwirken von individueller und kollektiver Ebene, von Teil und Ganzem, deutlich machen und „Emergenzphänomene“ veranschaulichen können. In den Sozialwissenschaften gehören solche „Mikro-Makro-Modelle“ inzwischen zu den wichtigsten Ansätzen, um soziale Phänomene besser zu erklären. Bei den Anfang der 70er Jahre von Thomas Schelling entwickelten Modellen beispielsweise geht es um ein elementares Verständnis der Ghettobildung [5].

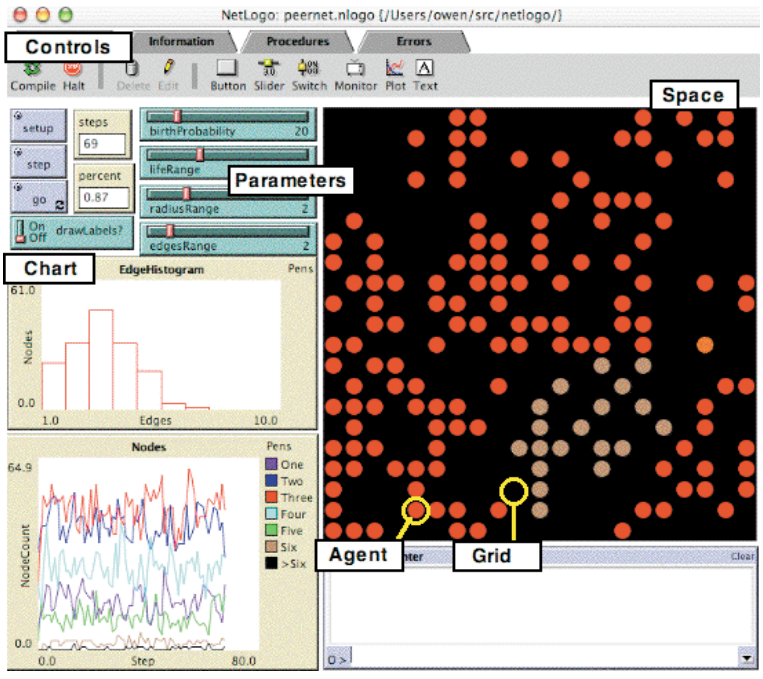
Stellen Sie sich eine Welt aus grünen und roten Menschen vor. Jeder bewohnt ein Haus, die Häuser in der Nachbarschaft sind entweder leer oder von einem Grünen oder Roten bewohnt. Da Menschen im Allgemeinen den Wunsch haben, mit Personen zusammenzuleben, die ihnen ähnlich sind, werden die Grünen eine Umgebung bevorzugen, in der die meisten der gleichen Gruppe angehören. Ähnliche Überlegungen stellen die Roten an.

Von Zeit zu Zeit erhalten die Individuen die Chance umzuziehen. Ein unzufriedener Grüner wird dann versuchen in eine Umgebung zu ziehen, in der die meisten ebenfalls grün sind. Jeder solche Umzug erhöht natürlich den Anteil der Grünen in der neuen Gruppe, was nun wiederum Mitglieder der roten Gruppe zum Umzug in eine Gegend mit geringerem Anteil der Grünen veranlasst und so fort.

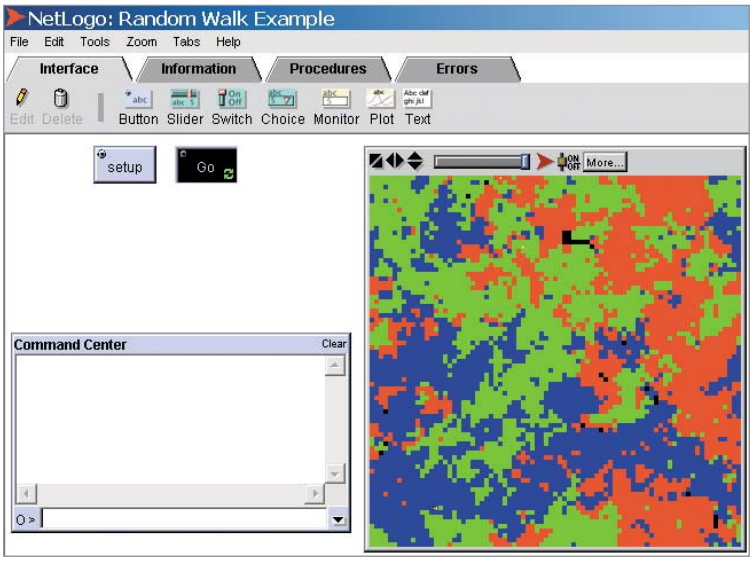
Wenn Sie das Modell „Segregation“ im „Social-Science“-Verzeichnis der Modellbibliothek von NetLogo laden und den „Setup-Button“ drücken, so sehen Sie eine Welt aus zufällig verteilten grünen und roten Turtles, die bestimmte Zellen (Häuser) besetzen. Die Anzahl der Turtles lässt sich über den Number-Slider einstellen. Jede Turtle ist mit einer Regel ausgestattet. Eine Regel für eine grüne Turtle lautet: „Wenn weniger als X Prozent Grüne in der Nachbarschaft sind, dann versuche eine Zelle zu besetzen, die mindestens X Prozent Grüne als Nachbarn hat“. Der %-similar-wanted-Slider legt die Toleranzschwelle X der Agenten fest.

Der „Percent-Similar-Monitor“ zeigt den durchschnittlichen Anteil gleichartiger Nachbarn für jedes Turtle. Am Anfang beträgt der Wert ungefähr 50 Prozent weil jedes Turtle im Schnitt mit der gleichen Anzahl von grünen und roten Nachbarn beginnt. Der Percent Unhappy Monitor zeigt den relativen Anteil der „unglücklichen“ Turtles, die also weniger gleichartige Nachbarn haben als sie möchten.

Ein Druck auf den Go-Button – und los geht das Gewusel. Wenn alle Turtles zufrieden sind, stoppt die Simulation. Schon bei einer relativ geringen Toleranzschwelle von 30 Prozent bilden sich nach einigen Simulationsläufen zusammengehörige Gruppen von Grünen und Roten. Man kann nicht sagen, dass es sich bei diesen Turtles um Rassisten handelt, denn jeder wollte ja nur in der Nachbarschaft 30 Prozent von seiner Art haben – gleichwohl reicht das vollständig



Die zentralen Elemente der NetLogo-Simulationsumgebung: Die Agenten bevölkern das schwarze Spielfeld rechts.



Ballett mit drei Turtles: Drei Agenten folgen einem Zufallsweg und ziehen eine farbige Spur hinter sich her.

aus, um eine völlige Segregation einer Gesellschaft zu erzeugen. Dieses Segregationsmodell veranschaulicht sehr schön ein zentrales Anliegen der Sozialwissenschaften, nämlich zu zeigen, dass individuelles Verhalten kollektive Phänomene erzeugen kann, die niemand beabsichtigt hat.

Die Experimentiermöglichkeiten mit den zwei Parametern sind zwar beschränkt, dennoch kann man bestimmte Fragestellungen via Parameter-Variation beantworten. Welche Konsequenzen auf Makro-Ebene hat beispielsweise leichter Rassismus im Vergleich zu extremem Rassismus? Was passiert bei starkem Rassismus und geringer/hoher Bevölkerungsdichte? Unter welchen Bedingungen stoppt die Simulation nicht?

Das Programm

Programmtechnisch besteht das zugehörige NetLogo-Modell aus zwei Hauptprozeduren: Zum einen ist dies die Prozedur `setup`, die die Variablen initialisiert und die Turtles auf die Patches verteilt. Zum andern die Prozedur `go`, das Hauptprogramm, das die Zufriedenheit der Turtles prüft und sie gegebenenfalls umziehen lässt.

```
to setup
  ca
  cct number
  [
    setxy (random screen-size-x)
          (random screen-size-y)
    ifelse who < (number / 2)
      [ set color red ]
      [ set color green ]
    if any other-turtles-here
      [ find-new-spot ]
  ]
  update-variables
  do-plots
end
```

`cct number` (create-custom-turtles number) erzeugt eine über einen Schiebeschalter abgefragte Anzahl von Turtles und führt gleichzeitig für jeden erzeugten Agenten die Anwei-

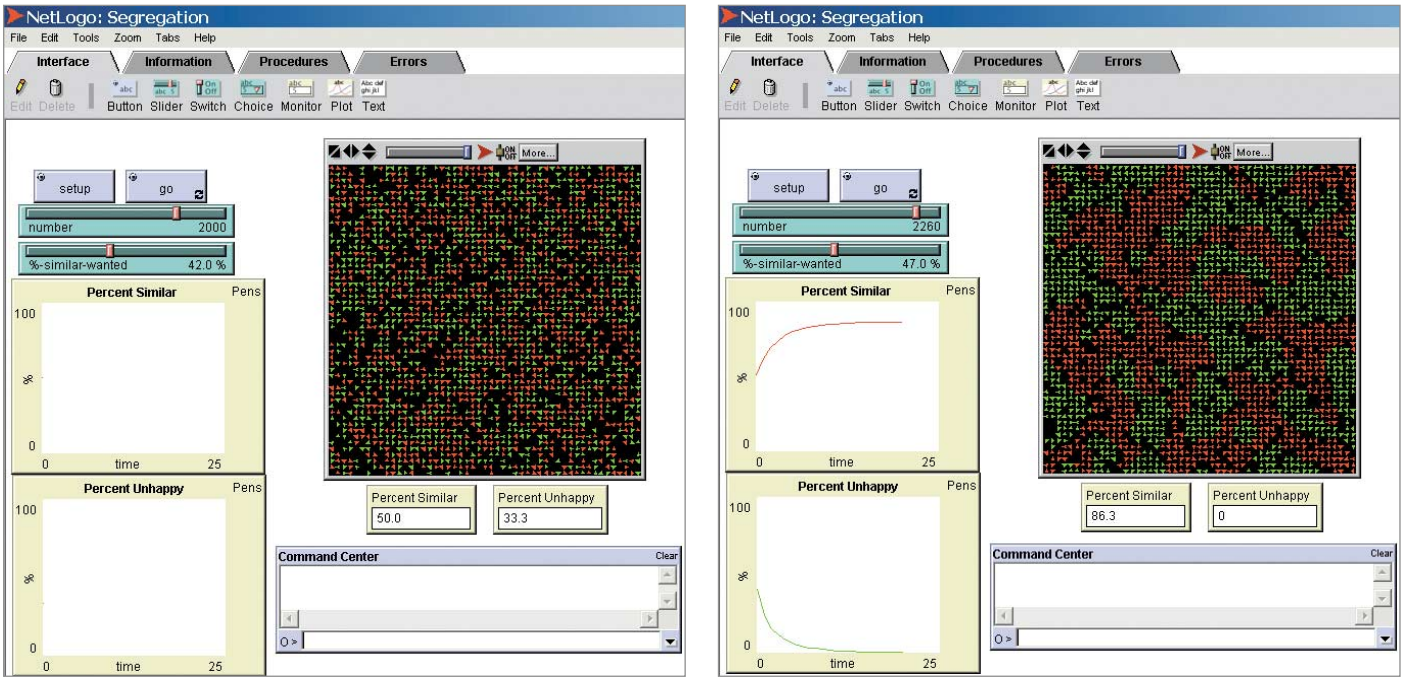
Sprachliche Unterschiede

StarLogo unterscheidet bereits bei der Programmeingabe streng zwischen Prozeduren, die der Observer ausführen darf, und Prozeduren, die Turtles ausführen dürfen – konsequenterweise müssten eigentlich auch noch Patch-Prozeduren separat behandelt werden, die haben die StarLogo-Schreiber aber dem Observer zugeschlagen. Das – leicht abgewandelte – Beispielprogramm muss also umorganisiert werden: Die einzelnen Schildkröten bekommen im `setup` einen zufälligen Farbwert zugewiesen, auf den sie später wieder zugreifen. Die Observer-Prozedur `to walk` ruft jetzt die Turte-Prozedur `to random_walk` auf. Im Observer-Kommandozentrum steht jetzt also:

```
to setup
  ca
  crt 3
  ask-turtles [ set color random 254 ]
end
to walk
  ask-turtles [ random_walk ]
end
```

Und im Turtle-Kommandozentrum ist definiert, was die Schildkröte nun tatsächlich tun muss. `stamp color` färbt dabei ein Patch in der Farbe der aktuellen Schildkröte, die diesen Befehl gerade ausführt:

```
to random_walk
    rotate random 360
    forward 1
    stamp color
    wait 0.1
end
```



Gruppentrennung: Schon eine leichte Vorliebe für Turtles der gleichen Farbe (linkes Bild) führt von der anfänglich zufälligen Verteilung zur Ghettobildung (rechtes Bild).

sung in eckigen Klammern aus. Im Gegensatz zu crt können mit cct jedem Turtle gleich bestimmte Eigenschaften zugeordnet werden.

Der Befehl setxy x y setzt jede Turtle auf eine Zelle mit den Koordinaten x und y. Im Beispiel werden die Koordinaten x und y zufällig bestimmt. Die x-Koordinate liefert random screen-size-x mit einem Zufallswert zwischen 0 und der Spielfeldbreite (screen-size-x). Analoges gilt für die y-Koordinate.

Die ifelse-Anweisung färbt – unter Ausnutzung der eingebauten Turtle-Variable who, die die laufende Nummer der Turtle enthält – die Hälfte der Turtles rot, die andere grün. Trifft der Bedingungsteil von ifelse zu, wird set color red ausgeführt, ansonsten set color green. Der eingebaute Befehl other-turtles-here prüft, ob noch weitere Turtles auf einem Patch sitzen. Trifft die Bedingung zu, wird die Prozedur find-new-spot aufgerufen:

```
to find-new-spot
  rt random 360
  fd random 10
  if any other-turtles-here
    [ find-new-spot ]
end
```

find-new-spot lässt die Turtle umziehen. Dazu wählt sie zufällig eine Richtung zwischen 0 und 360 Grad und marschiert zwischen einem und zehn Zufallsschritten vorwärts. Daraufhin erfolgt erneut die Prüfung, ob schon eine Turtle den Patch besetzt hält. Im positiven Fall wird die Prozedur rekursiv nochmals aufgerufen – und dies so lange, bis die Turtle eine freie Heimstätte gefunden hat.

Die nächste setup-Anweisung update-variables aktualisiert alle Variablen. Sie besteht wiederum aus drei weiteren Prozeduren: update-patches aktualisiert die Patches, update-turtles die Turtles und update-globals globale Variablen.

```
to update-variables
  update-patches
  update-turtles
  update-globals
end
```

update-patches zählt für jede Zelle die Anzahl ihrer roten und grünen Nachbarn und bildet die Gesamtsumme der Zellnachbarn:

```
to update-patches
  ask patches [
    set reds-nearby count neighbors with [any turtles-
    here with [color = red]]
    set greens-nearby count neighbors with [any turtles-
    here with [color = green]]
    set total-nearby reds-nearby + greens-nearby
  ]
end
```

Die nutzerdefinierte Patch-Variable reds-nearby zählt alle aktuellen roten Nachbarn auf den umgebenden acht Zellen, greens-nearby alle grünen Nachbarn, total-nearby addiert diese beiden Werte.

neighbors zeigt eine grundlegende Schwierigkeit in NetLogo und StarLogo: Das Kommando liefert tatsächlich die acht Nachbarn eines Patches zurück – die Eigenschaften dieser Patches müssen dann wieder mit Befehlen festgestellt werden, die man darauf anwendet. Was eine Funktion tatsächlich zurückliefert – eine Turtle, eine Zahl oder eine boolesche Variable – geht aus der Dokumentation nicht immer klar hervor.

Als Nächstes werden die Turtles-Werte aktualisiert. update-turtles stellt fest, ob eine Turtle zufrieden oder unzufrieden ist:

```
to update-turtles
  ask turtles [
    if color = red
      [ set happy? reds-nearby >= ( %-similar-wanted *
      total-nearby / 100 ) ]
    if color = green
      [ set happy? greens-nearby >= ( %-similar-wanted
      * total-nearby / 100 ) ]
  ]
end
```

ask turtles weist wieder jede Turtle an, den Befehl in eckigen Klammern auszuführen. Bei einer roten Turtle ist die Turtle-Variable happy? wahr, wenn die Anzahl der roten Nachbarn (reds-nearby) größer oder gleich der Toleranzschwelle multipliziert mit der Summe aller Nachbarn ist (total-nearby). Jetzt sind die Turtles platziert und alle Variablen gesetzt. Das Hauptprogramm go startet nun die Simulation.

```
to go
  move-unhappy-turtles
  update-variables
  do-plots
  if not any turtles with [not happy?] [ stop ]
end
```

move-unhappy-turtles lässt die unzufriedenen Turtles umziehen:

```
to move-unhappy-turtles
  ask turtles [
    if not happy?
      [ find-new-spot ]
  ]
end
```

Sie fragt jede Turtle ab, ob sie unzufrieden ist. Ist dies der Fall, versetzt sie sich mit find-new-spot an einen zufällig gewählten freien Platz in der Nähe. Die Simulation stoppt, wenn es kein unzufriedenen Turtles mehr gibt.

Herdentrieb

Mit Hilfe des breeds-Befehls kann man in einer Simulation auch mehrere Agenten-

NetLogo und StarLogo – Download und Service

NetLogo und StarLogo laufen unter allen gängigen Windows-, Macintosh- und Linux-Versionen (Download über Soft-Link). Vorausgesetzt wird lediglich eine Java Virtual Machine (JVM) ab Version 1.1 oder höher. Beim Download von NetLogo für Windows hat der Nutzer die Wahl, ob er es ohne oder mit Java VM herunterladen will.

Eine rege NetLogo-Community sorgt für Unterstützung. Über die User-Group kann man Fragen, Ideen und Beispiele diskutieren und austauschen (http://groups.yahoo.com/group/netlogo-users/). Wer sein Modell vorstellen will oder an einem Problem knabbert, ist hier richtig. Die Educators Group ist für Lehrer gedacht, die NetLogo im Unterricht einsetzen wollen (http://groups.yahoo.com/group/netlogo-educators/). Zudem sind alle NetLogo-Anwender aufgerufen, ihre Modelle der Community zur Verfügung zu stellen. Zu diesem Zweck wird für jedes Modell beim Upload eine eigene Website generiert. Analoge Diskussionen finden auch auf der StarLogo Design Discussion Area (http://education.mit.edu/dda) statt.

Mengen mit unterschiedlichen Eigenschaften und Verhaltensweisen definieren und elegant steuern. Hierzu muss vor den Prozeduren mit dem Schlüsselwort „breeds“ definiert werden, welche Gruppen unterschieden werden:

Die Setup-Prozedur könnte dann wie folgt aussehen:

```
breeds [gruene, blaue]
to setup
  ca
    cct-blaue number / 2
  [
    setxy (random screen-size-x)
          (random screen-size-y)
          set color blue
       if any other-turtles-here
     [ find-new-spot ]
  ]
  cct-gruene number / 2
  [
    setxy (random screen-size-x)
          (random screen-size-y)
          set color green
       if any other-turtles-here
     [ find-new-spot ]
  ]
end
```

In diesem Fall werden die zwei Agenten-Mengen „gruene“ und „blaue“ erzeugt. Sämtliche eingebauten Befehle und Variablen, die für Turtles allgemein gelten, lassen sich auf einzelne Breeds anwenden. `ask blaue [ forward 10 ]` lässt beispielsweise alle Turtles, die zum Stamm der blauen gehören zehn Schritte vorrücken. `blaue-here` liefert alle Turtles zurück, die sich auf dem anfragenden Patch befinden und zum Breed blaue gehören.

Update-turtles lässt sich mit Hilfe von Breeds variieren zu:

```
to update-turtles
  ask patches [ set gruene-nearby sum values-from
  neighbors [count gruene-here]
  set blaue-nearby sum values-from neighbors
  [count blaue-here]
  set total-nearby gruene-nearby + blaue-nearby  ]

     ask blaue
        [ set happy_blue? blaue-nearby >=
        ( %similar_wanted_blau * total-nearby / 100 ) ]
     ask gruene
      [ set happy_green? gruene-nearby >=
      ( %similar_wanted_gruen * total-nearby / 100 ) ]
end
```

Die aus der ersten Variante bekannte Variable `%similar-wanted` kann jetzt für beide Breeds unterschieden werden. Der experimentierfreudige Observer kann so testen, was für Muster sich ergeben, wenn beispielsweise die blaue Spezies „fremdenfeindlicher“ als die grüne ist, oder zunächst die eine Spezies umziehen lassen und dann die zweite. Der Neugier sind kaum Grenzen gesetzt. Das Segregationsmodell ist nur eines von vielen sozialwissenschaftlichen Modellen, die sich in der Library finden. Ebenfalls vorhanden sind die berühmten Gefangenendilemma-Simulationen von Robert Axelrodt zur „Evolution von Kooperation“ sowie mehrere Varianten davon [6]. Wer einmal angefangen hat, sich auf dezentrale Systeme und Selbstorganisation einzulassen, wird sich so schnell nicht langweilen. (wst)

Literatur

[1] http://el.www.media.mit.edu/groups/logo-foundation/index.html
[2] http://mindstorms.lego.com
[3] Weiter Multi-Agenten-Systeme: www.swarm.org, www.cpm.mmu.ac.uk/sdml, http://repast.sourceforge.net
[4] NetLogo Website: http://ccl.northwestern.edu/netlogo
[5] Thomas Schelling, Micromotives and Macrobehavior, Norton 1978
[6] Winfried Eggebrecht und Klaus Manhart, Fatale Logik, Egoismus oder Kooperation in der Computersimulation, c't 6/91, S. 144

Soft-Link 0325232