Katharinen-Gymnasium Ingolstadt * Jahrgang 2004/2006

Facharbeit
aus dem Leistungskursfach
Mathematik

Thema: **Die mathematische Modellierung der Staubildung** -
Analysen mit Hilfe des Nagel-Schreckenberg-Modells zur Verkehrsflussoptimierung

Verfasser: **Christoph Settgast**
Leistungskurs: **M1**
Kursleiter: **Herr Wolfseher**
Abgabe am: **27.01.2006**

Bewertung der Arbeit: (schriftlicher Teil) _____ Punkte

_____ Unterschrift des Kursleiters

Inhalt

1. Einleitung 1

2. Beschreibung des Phänomens, Definitionen 1

3. Vorstellung des Nagel-Schreckenberg-Modells zur Simulation eines Verkehrsflusses 2
3.1 Theoretischer Ansatz 2
3.2 Beispiel 3
3.3 Kenngrößen des Verkehrs 4
3.3.1 Dichte ρ 4
3.3.2 Verkehrsfluss Φ 5
3.3.3 Fundamentaldiagramm 5
3.4 Empirische Überprüfung des Modells 5

4. Simulation des Nagel-Schreckenberg-Modells am PC 10
4.1 Grundlegendes Konzept 10
4.2 Vorstellung des Programms 12

5. Erweiterung des Nagel-Schreckenberg-Modells 13
5.1 VDR-Modell 13
5.2 Zwei- und dreispurige Straßen 17

6. Erklärung des Phänomens „Stau aus dem Nichts“ 19
6.1 Abgrenzung und Erläuterung 19
6.2 Die Bedeutung des Trödelfaktors p 22
6.3 Mögliche Gründe für Trödelfaktor p im Straßenverkehr 24
6.3.1 Unaufmerksames Fahren 24
6.3.2 Überhöhte Geschwindigkeit und zu dichtes Auffahren 24
6.3.3 Spurwechsel bei dichtem Verkehr 24

7. Vorstellung einiger Lösungsansätze zur Vermeidung von Stau 25
7.1 Generelles Tempolimit 25
7.2 Verkehrsleitsysteme 27

8. Persönliches Fazit 29

9. Anhang 31
9.1 Abbildungsverzeichnis 31
9.2 Literaturverzeichnis 32
9.3 Listing des Simulationsprogramms 33
9.3.1 Datei CAmodel.java 33
9.3.2 Datei RoadCanvas.java 39
9.3.3 Datei Road.java 42
9.3.4 Datei MessageBox.java 49

1. Einleitung

"The volume of vehicular traffic in the past several years has rapidly outstripped the capacities of the nation's highways. It has become increasingly necessary to understand the dynamics of traffic flow and obtain a mathematical description of the process."[1]

H. Greenberg, 1959

Auch wenn dieser Satz bereits 47 Jahre alt ist, so hat er doch nichts von seiner Aktualität eingebüßt. Im Gegenteil: Es ist heute wichtiger denn je, die Dynamik des Verkehrsflusses zu verstehen, denn folgende Situation ist wahrscheinlich jedem Autofahrer bekannt: Man fährt auf der Autobahn Richtung Süden, es herrscht zwar dichter, aber dennoch fließender Verkehr. Plötzlich bremsen die vorausfahrenden Fahrzeuge, und, eh man sich versieht, steht man auch selbst. Kurzzeitig geht nichts mehr, vor und hinter dem eigenen Auto steht alles, und es stellt sich die Frage, woher der Stau kommt. Bis unmittelbar zuvor floss der Verkehr noch. Gibt es ein Hindernis, z.B. eine Baustelle? Irgendwann bewegen sich die vorausfahrenden Fahrzeuge wieder, langsam kommt Schwung in die Blechlawine, am Ende des Staus ist jedoch kein Hindernis zu erkennen, das den Stau verursacht haben könnte. Woher kam dieser Stau? Was war die Ursache? Wie kann dieser „Stau aus dem Nichts" verhindert werden? Das sind die Fragen, die in dieser Arbeit beantwortet werden sollen.

2. Beschreibung des Phänomens, Definitionen

Auch wenn das Phänomen „Stau" den meisten Menschen ein Begriff ist, muss es doch für eine wissenschaftliche Arbeit genauer definiert werden.

Unter einem „Stau" wird in dieser Analyse eine stehende Fahrzeugkolonne von mehr als drei Fahrzeugen verstanden. Der Stillstand kann für einen kurzen Zeitraum sein, er kann sich jedoch auch über einen längeren Zeitraum erstrecken. Immer, wenn auf einer Landstraße oder einer Autobahn mehr als drei Fahrzeuge zum Stillstand gekommen sind, ist eine merkliche Verkehrsbehinderung anzunehmen.

„Stau aus dem Nichts" lässt sich wie folgt beschreiben: Der schlagartige Zusammenbruch eines Verkehrsflusses ohne ein vorangegangenes Hindernis wie einer Baustelle, einem Unfall oder Ähnlichem. Dieser „Stau aus dem Nichts" löst sich eine gewisse Zeit nach seiner Entste-

[1] Helbing, Dirk: *Traffic and related self-driven many-particle systems*, Seite 2

hung meist selbst auf, des Weiteren sind die Standzeiten der einzelnen Fahrzeuge nicht besonders lange. Der Stau selbst kann jedoch über einen längeren Zeitraum bestehen, bis er sich wieder vollständig auflöst.

3. Vorstellung des Nagel-Schreckenberg-Modells zur Simulation eines Verkehrsflusses

Um einen solchen „Stau aus dem Nichts“ zu verstehen, bietet es sich an, das Verkehrsgeschehen in einem Modell zu simulieren. Hierzu eignet sich das Zellularautomatenmodell von Kai Nagel und Michael Schreckenberg, das 1992 an der Universität Köln entwickelt wurde.[2]

3.1 Theoretischer Ansatz

Die allgemeine Definition eines Zellularautomatenmodells lautet: Zellularautomaten oder zelluläre Automaten sind diskrete Systeme, also Systeme, die mit fest definierten Zuständen arbeiten. Zellularautomaten trennen ein komplexes Bewegungssystem in abgetrennt definierte Steckenabschnitte von gleicher Länge und weisen einzelnen sich bewegenden Objekten jeweils zu einer genau bestimmten Zeit einen genau bestimmten Ort und eine entsprechende Geschwindigkeit zu. Des Weiteren reagieren die einzelnen Objekte des Zellularautomaten auf die Position der anderen Objekte im System. Zellularautomaten eignen sich sehr gut für eine Übertragung in ein Computersimulationsprogramm.[3]

Das Nagel-Schreckenberg-Modell ist nun wie folgt aufgebaut: Die Straße wird in gleich lange Zellen von je 7,5 m Länge unterteilt, da dies die realistisch dichteste Packung von Fahrzeugen in einem Stau inklusive geringem Sicherheitsabstand darstellt. Jede Zelle kann nun entweder den Zustand „besetzt“ oder „frei“ annehmen. Ist die Zelle „besetzt“, so ist dem in der Zelle befindlichen Fahrzeug eine definierte Geschwindigkeit zugewiesen,[4] vereinfacht als ganze Zahl zwischen 0 und 7. Die Zahl 0 repräsentiert hier ein stehendes Fahrzeug, die Zahl 7 mit linearer Proportionalität die Geschwindigkeit v = 189 km/h.

Das Nagel-Schreckenberg-Modell arbeitet mit Simulationsabschnitten (auch Runden genannt), pro Runde werden für jedes Fahrzeug eine Geschwindigkeitsberechnung und eine Ortsberechnung ausgeführt.[5]

---

[2] Nagel, K., Schreckenberg, M.: *A cellular automaton model for freeway traffic*
[3] Schadschneider, Andreas: *Verkehrsmodelle* (WWW-Dokument)
[4] Schadschneider, Andreas: *Verkehrsmodelle* (WWW-Dokument)
[5] Nagel, K., Schreckenberg, M.: *A cellular automaton model for freeway traffic*, Seite 2

Geschwindigkeitsberechnung:

- → **Beschleunigungsregel**: Wenn ein Fahrzeug noch nicht die Maximalgeschwindigkeit $v_{max} = 7$ erreicht hat, so erhöht sich die Geschwindigkeit v des Fahrzeugs um 1.
- → **Sicherheitsregel**: Hat das Fahrzeug eine höhere Geschwindigkeit v als Zellen vor ihm frei sind, so wird die Geschwindigkeit v auf die Zahl der freien Zellen vor dem Fahrzeug (*gap*) gesetzt.
- → **Trödeln**: Mit einer gewissen Wahrscheinlichkeit p verringert sich die Geschwindigkeit des Fahrzeugs v um 1.

Ortsberechnung:

Alle Fahrzeuge werden um genau so viele Felder nach vorne versetzt, wie ihre Geschwindigkeit v nach den Regeln der Geschwindigkeitsberechnung beträgt.[6]

3.2 Beispiel

Zur Veranschaulichung des Modells soll nun der Verlauf einer Runde an folgendem Beispiel gezeigt werden.

Zu Beginn liegt folgende Situation vor: In der ersten Zelle steht Fahrzeug A mit $v = 0$, in der zweiten Zelle befindet sich Fahrzeug B mit der Geschwindigkeit $v = 2$, in der vierten Zelle befindet sich Fahrzeug C mit der Geschwindigkeit $v = 2$ und in der siebten Zelle befindet sich Fahrzeug D mit der Geschwindigkeit $v = 1$. Alle Fahrzeuge bewegen sich von links nach rechts vorwärts.

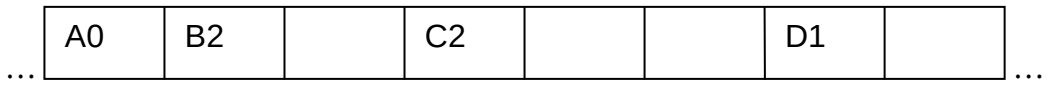

Abb. 1: Beispielhafte Erklärung, Ausgangssituation, eigene Darstellung

Nun werden die vier Regeln ausgeführt, exemplarisch zeige ich dies an Fahrzeug C:

Beschleunigen: Da Fahrzeug C noch nicht die Maximalgeschwindigkeit $v_{max} = 7$ erreicht hat, erhöht sich seine Geschwindigkeit v um 1 auf $v = 3$. Diese Regel des Modells beschreibt also den Wunsch der Autofahrer, sich so schnell wie möglich fortzubewegen.

Sicherheit: Da nur zwei freie Zellen vor Fahrzeug C sind, verringert sich die Geschwindigkeit v wieder auf $v = 2$. Hier wird der Sicherheitsgedanke beim Autofahren berücksichtigt, Auffahrunfälle werden so verhindert.

Trödelfaktor: Mit einer Wahrscheinlichkeit p verringert sich die Geschwindigkeit v des Fahrzeugs C um 1 auf $v = 1$. Diese Regel ist sehr wichtig, da sie die Unvollkommenheit der Autofahrer beschreibt, die manchmal zu träge reagieren, manchmal mehr bremsen als not-

[6] Schadschneider, Andreas: *Verkehrsmodelle*

wendig oder ihre Reisegeschwindigkeit aus anderen Gründen während der Fahrt leicht vermindern.

Ortsberechnung:

Alle Fahrzeuge werden entsprechend ihrer Geschwindigkeit nach vorne versetzt, bei Fahrzeug A ist dies 0, bei Fahrzeug B 1, bei Fahrzeug C 1, wie oben gezeigt, bei Fahrzeug D 2.

Daraufhin ergibt sich folgendes Bild:

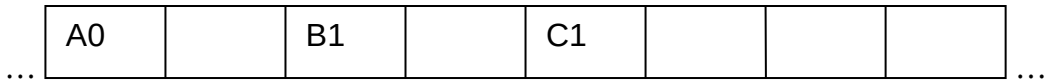

Abb. 2: Beispielhafte Erklärung, nach einer Runde, eigene Darstellung

Wenn nun eine weitere Runde abläuft, so ergibt sich nach den Regeln des Modells folgendes Bild:

Abb. 3: Beispielhafte Erklärung, nach Runde 2, eigene Darstellung

An diesem Beispiel sieht man, dass alle Regeln des Modells Sinn ergeben und gleichzeitig das Modell minimal[7] ist, denn wenn eine der Regeln, z. B. die Sicherheitsregel, weggelassen wird, so kann man nicht mehr von einer Verkehrssimulation sprechen.

3.3 Kenngrößen des Verkehrs

Um eine wissenschaftliche Vergleichbarkeit der Simulationsergebnisse zu erreichen, müssen fest definierte Kenngrößen eingeführt werden. Diese sind die Dichte ρ des Verkehrs, der Verkehrsfluss Φ und zur besseren Vergleichbarkeit das Fundamentaldiagramm, welches die beiden Größen ρ und Φ in Relation stellt.

3.3.1 Dichte ρ

Die Dichte ρ beschreibt die Zahl der Fahrzeuge pro Zelle. Sie kann also maximal den Wert 1 annehmen. Im obigen Beispiel betrug die Dichte ρ zu Beginn 0,5, da jede zweite Zelle besetzt war.

[7] Schadschneider, Andreas: *Verkehrsmodelle*

3.3.2 Verkehrsfluss Φ

Die Größe des Verkehrsflusses Φ beschreibt die Anzahl der Fahrzeuge pro Zeiteinheit an einer definierten Messstelle. Ein hoher Verkehrsfluss Φ wird im Allgemeinen als gute Auslastung der Straße empfunden, da dann besonders viele Fahrzeuge pro Zeiteinheit eine Stelle passieren können.

3.3.3 Fundamentaldiagramm

Um das Modell gerade in der kritischen Phase der Entstehung des Staus noch zu verfeinern, ist die Einführung eines so genannten Fundamentaldiagramms sinnvoll. Das Fundamentaldiagramm beschreibt in dieser Arbeit grafisch die Abhängigkeit des Verkehrsflusses Φ von der Dichte ρ.

3.4 Empirische Überprüfung des Modells

Um zu zeigen, dass das Nagel-Schreckenberg-Modell eine Verkehrssimulation darstellt, wird es mit empirisch gewonnenen Verkehrsmessungen verglichen.
Folgende in den USA aus Luftaufnahmen gewonnene Grafik[8] beschreibt mit jeder Linie die Position eines Fahrzeugs auf der rechten Spur einer Straße in Abhängigkeit von der Zeit. Plötzlich endende oder neu entstehende Linien sind durch Spurwechsel bedingt.

[8] Treiterer, J.: *Ohio State Technical Report*

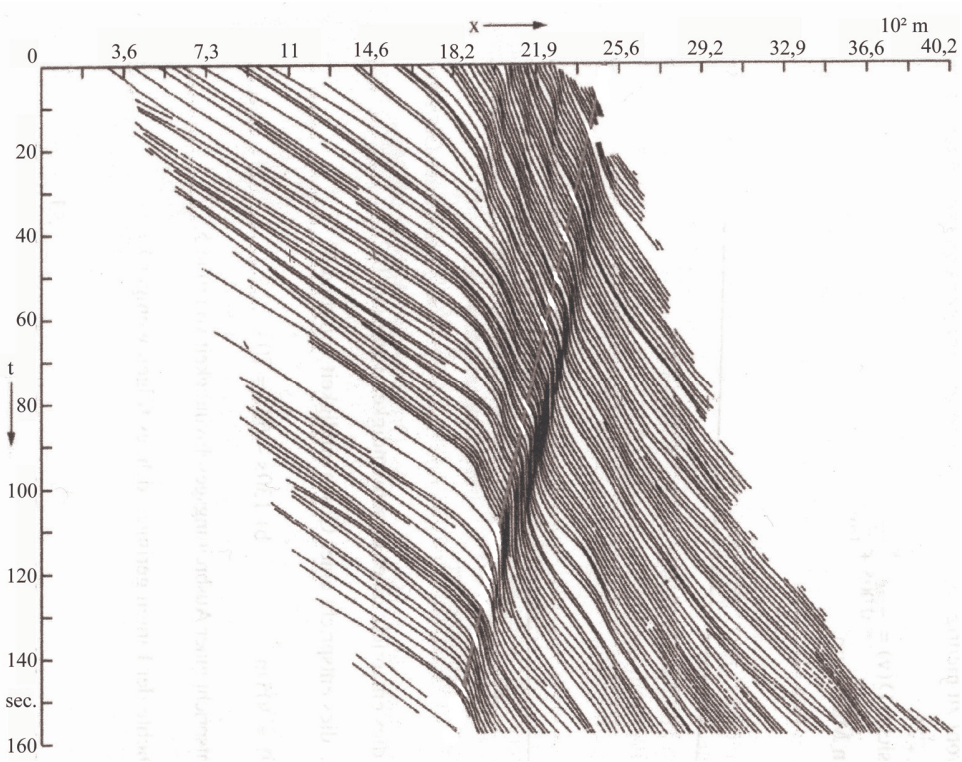

Abb. 4: Empirische Verkehrsmessung
aus J. Treiterer: Ohio State Technical Report, eigene Bearbeitung

Man sieht hier die Entstehung eines kurzen „Staus aus dem Nichts“, der rund 110 Sekunden existiert und sich entgegen der Fahrtrichtung bewegt.

Um das Nagel-Schreckenberg-Modell mit den empirischen Daten vergleichen zu können, muss die vorliegende Dichte ρ und das Tempolimit von rund 105 km/h berücksichtigt werden. Die Dichte ρ lässt sich aus der Grafik mit $\rho = 0{,}15$ errechnen, das Tempolimit entspricht einer Höchstgeschwindigkeit $v_{max} = 4$ Schritten pro Zeiteinheit.

Mit diesen Daten ergibt sich folgendes Bild:

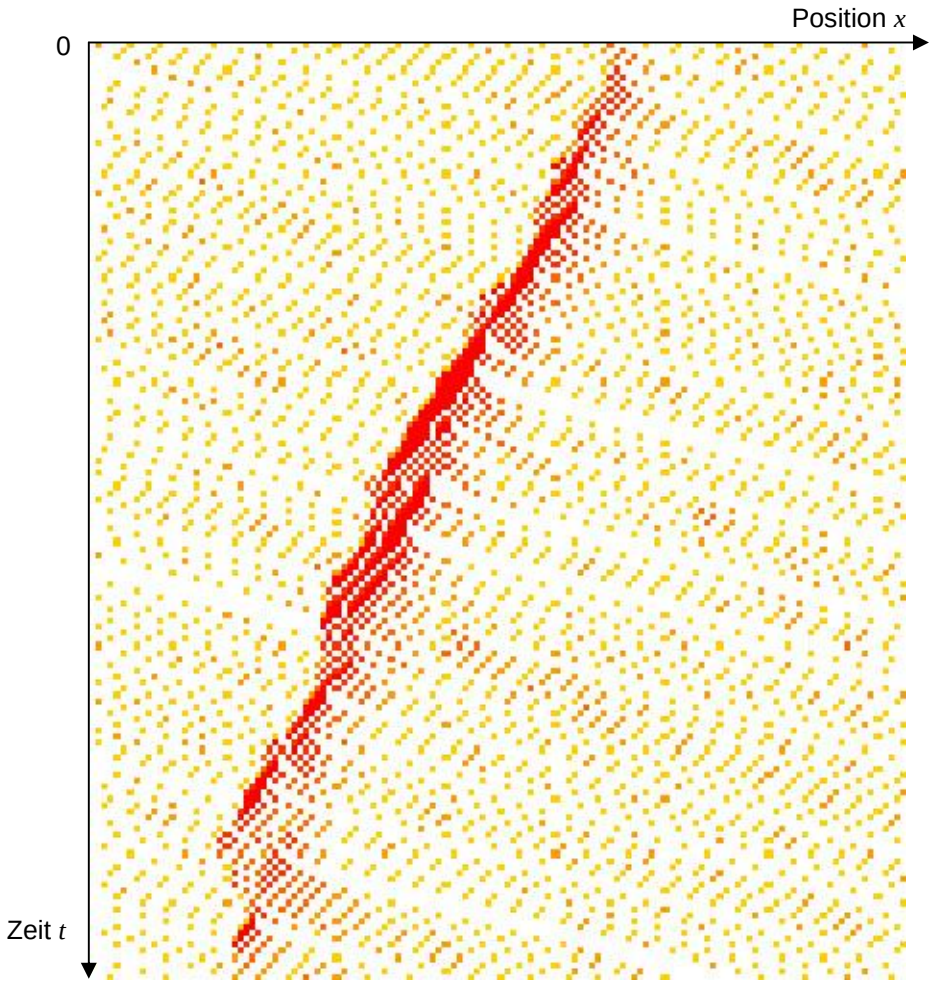

Abb. 5: Empirische Daten im Nagel-Schreckenberg-Modell, eigene Darstellung

In Abb. 5 ist zu erkennen, dass sich die empirische und die simulierte Situation stark ähneln. In der Simulation entsteht ebenfalls ein „Stau aus dem Nichts“, die Staulänge ist mit rund sieben Fahrzeugen vergleichbar mit der empirischen Untersuchung, in der maximal rund zehn Fahrzeuge zum Stillstand kamen. Die Zeit, die der Stau existiert, ist ebenfalls vergleichbar. Im Nagel-Schreckenberg-Modellversuch dauerte es vom Entstehen bis zur Auflösung des Staus rund 120 Sekunden, aus den empirischen Daten in Abb. 4 lässt sich eine Dauer von 110 Sekunden ablesen. Man kann also sagen, dass das Nagel-Schreckenberg-Modell bereits eine sehr genaue Verkehrssimulation ermöglicht.

Auch das Fundamentaldiagramm des Nagel-Schreckenberg-Modells lässt sich mit empirisch gewonnenen Daten vergleichen.

Folgendes Diagramm (Abb. 6), welches die Abhängigkeit des Verkehrsflusses Φ von der Dichte ρ darstellt, wurde auf dem „Queen Elizabeth Way“ in Ontario (Kanada) mit Hilfe lokaler Sensoren im Boden aufgenommen. Jeder Punkt im Diagramm stellt den Durchschnittswert des Verkehrsflusses Φ für die Zeitspanne von 5 Minuten dar.[9]

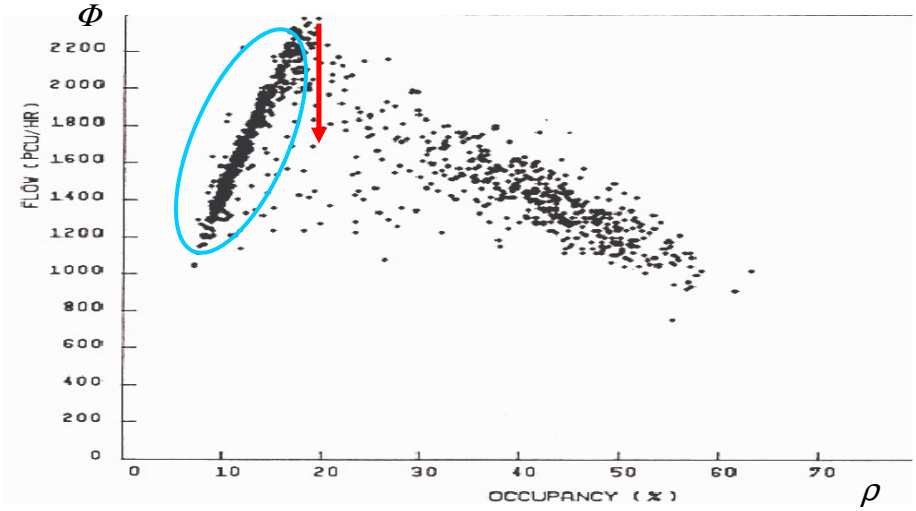

Abb. 6: Empirisches Fundamentaldiagramm
aus Hall, F. L., Allen, B. L., Gunter, M. A.: Empirical analysis of freeway flow-density relationships

Wie man in Abb. 6 erkennen kann, ist der Verkehrsfluss Φ bei geringer Dichte ρ noch linear abhängig (blaue Markierung), man spricht von frei fließendem Verkehr, auch „free-flow“[10] genannt. Erst ab einem bestimmten Wert für die Dichte ρ ändert sich dies schlagartig und es kommt zu einem starken Abfall des Verkehrsflusses Φ (roter Pfeil), ein so genannter „capacity drop“[11]. Jetzt fällt der Verkehrsfluss Φ mit zunehmender Dichte, eine deutliche Tendenz des Verkehrsflusses Φ ist erst ab einer Dichte $\rho \approx 35\%$ wieder erkennbar.

[9] Chowdhury, D., Santen, L., Schadschneider, A.: *Statistical Physics of Vehicular Traffic and Some Related Systems*, Seite 15
[10] Chowdhury, D., Santen, L., Schadschneider, A.: *Statistical Physics of Vehicular Traffic and Some Related Systems*, Seite 16
[11] Schadschneider, Andreas: *Grundlagen der Verkehrsdynamik*, Folie 23

Folgendes Diagramm wurde mit Hilfe des Simulationsprogramms und folgenden Einstellungen gewonnen: Modell: „Standard CA“; $v_{max} = 5$ (≈ 135 km/h)

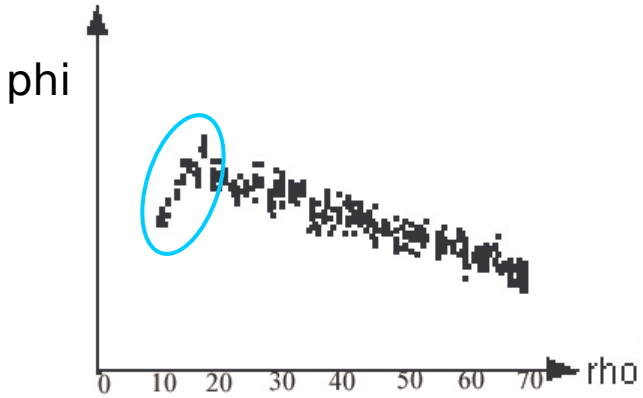

Abb. 7: Fundamentaldiagramm Nagel-Schreckenberg-Modell, eigene Darstellung

Wie man in Abb. 7 erkennen kann, stimmt die grobe Form des Fundamentaldiagramms mit den empirischen Daten aus Abb. 6 überein. Es existiert ebenfalls eine „free-flow“ – Phase (blaue Markierung), in der der Verkehrsfluss Φ mit zunehmender Dichte ρ proportional ansteigt, bei einer Dichte $\rho \approx 15\%$ folgt ein Knick und danach fällt der Verkehrsfluss Φ mit zunehmender Diche ρ immer weiter ab. Jedoch gerade der „capacity drop“ ist im Nagel-Schreckenberg-Modell noch nicht so stark ausgebildet. Dies wird in anderen Verkehrsmodellen besser berücksichtigt (siehe dazu Kapitel 5.1).

Um klarere Aussagen treffen zu können, ist es üblich, das Fundamentaldiagramm zu glätten und zu interpolieren, dabei besitzt das Nagel-Schreckenberg-Modell folgendes allgemeines Fundamentaldiagramm.

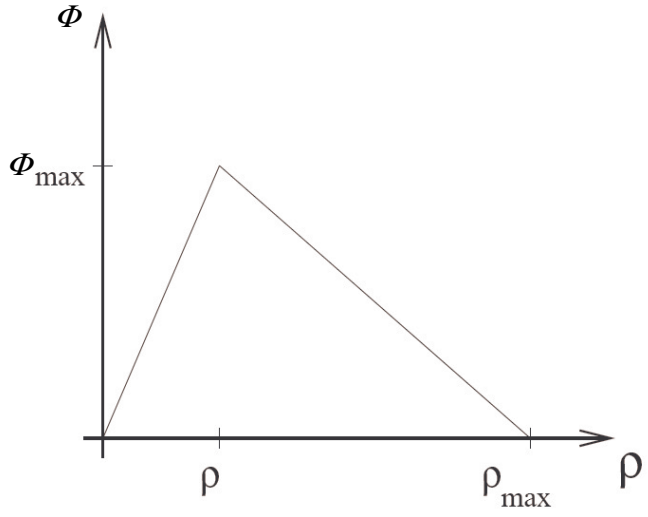

Abb. 8: Schematisches Fundamentaldiagramm Nagel-Schreckenberg-Modell aus Schadschneider, Andreas: Physik des Straßenverkehrs, Seite 11

Das Nagel-Schreckenberg-Modell stellt aufgrund der anschaulichen Übereinstimmungen bei Staulänge und Staudauer und des ähnlichen Fundamentaldiagramms bereits eine relativ genaue Verkehrssimulation dar. Mit Hilfe des Nagel-Schreckenberg-Modells kann die Dynamik des Verkehrsflusses in der Realität bereits so gut simuliert werden, dass es sinnvoll erscheint, das Modell in eine Computersimulation zu übertragen.

4. Simulation des Nagel-Schreckenberg-Modells am PC

4.1 Grundlegendes Konzept

Da die Erkenntnisgewinnung über den empirischen Weg, also über Luftaufnahmen oder Zählschleifen im Boden, sehr mühsam und langwierig ist, muss eine Alternative hierzu gefunden werden. Das Nagel-Schreckenberg-Modell eignet sich hervorragend für die Umsetzung in ein Computerprogramm[12], dass das Verkehrsgeschehen einschließlich eines Verkehrsstaus simuliert.

Das Programm besitzt folgendes Grundprinzip:

```
run() {
    Canvas.update(…) {
       if (buffer == null){…}; // Achsen zeichnen
        freeway.update(){
        //Autos zählen, gegebenenfalls hinzufügen, abziehen
        //Zellularautomat auf speed[i] anwenden
                Geschwindigkeit bestimmen
                Entsprechend vorrücken
        }
       freeway.paint(…){…}; //Ort-Zeit-Diagramm Zeichnen
       freeway.indypaint(…){…}; //Straße zeichnen
       freeway.measure(){…}; //Dichte ρ, Verkehrsfluss Φ messen
       freeway.diagram(…){…}; //Diagramme zeichnen
    }
}
```

Abb. 9: Schema des Kerns des Simulationsprogramms, eigene Darstellung

[12] Eisenblätter, B., Neubert, L., Wahle, J.: *CA_einspur.tgz* , http://www.traffic.uni-duisburg.de/homepage/model/model.html, 1998

Der Programmteil `run()` läuft dauerhaft ab, der eigentliche Simulationskern beschränkt sich jedoch auf den `freeway.update()` – Teil, in dem die Regeln des Nagel-Schreckenberg-Modells auf die simulierten Autos angewendet werden. Die Straße wird in dem Simulationsprogramm als eindimensionales Datenfeld aus Geschwindigkeiten (`speed[i]`) zusammengesetzt. Die Geschwindigkeit $v = -1$ bedeutet, dass sich in dieser Zelle kein Fahrzeug befindet, $v = 0$ bis $v = v_{max}$ entspricht einem Fahrzeug mit eben jener Geschwindigkeit.

Die Anwendung der Regeln des Nagel-Schreckenberg-Modells ist auf eine relativ einfache Art realisiert. Das Geschwindigkeitsarray wird von Anfang bis Ende durchlaufen und die Beschleunigungsregel, die Sicherheitsregel und der Trödelfaktor p werden auf jedes simulierte Fahrzeug angewendet. Danach wird das Geschwindigkeitsarray dementsprechend aktualisiert, sodass jedes simulierte Fahrzeug um die entsprechende Zahl von Zellen nach vorne versetzt wird.

Es folgen Programmteile, in denen die Dichte ρ und der Verkehrsfluss Φ der simulierten Straße bestimmt werden und in ein entsprechendes Fundamentaldiagramm gezeichnet werden.

Das Simulationsprogramm wurde an der Universität Duisburg entwickelt, für diese Arbeit jedoch um folgende Diagramme ergänzt: Entwicklung des Verkehrsflusses Φ in Abhängigkeit von der Zeit und Darstellung des aktuellen Verkehrsflusses Φ in Fahrzeugen/Minute. Außerdem wurde es auf deutsche Verhältnisse optimiert und das maximale Tempolimit im gesamten Programm von $v_{max} = 4$ auf $v_{max} = 7$ erhöht.

4.2 Vorstellung des Programms

Das Simulationsprogramm, das unter Kapitel 4.1 eingeführt wurde, zeigt bei der Ausführung Folgendes an.

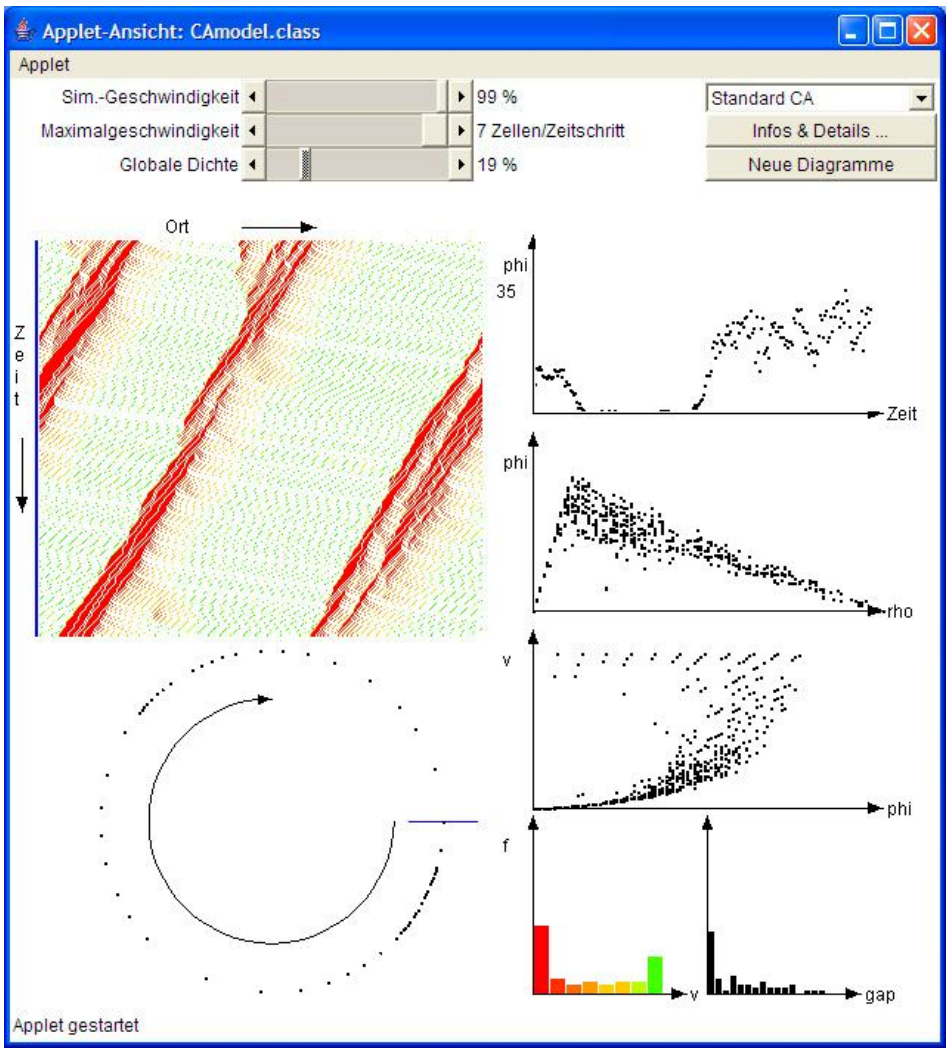

Abb. 10: Simulationsprogramm CA_einspur, eigene Darstellung

Wie in Abb. 10 ersichtlich, zeigt das Simulationsprogramm eine kreisförmige Straße, die in 300 Zellen unterteilt ist. „Das Fenster darüber zeigt die Trajektorien der einzelnen Fahrzeuge [also ihre Position in Abhängigkeit von der Zeit]. Der linke Rand (blaue Linie) entspricht dabei dem blauen Strich (bei 3 Uhr) in der Darstellung links unten. Die Fahrzeuggeschwindigkeiten werden durch unterschiedliche Farben gekennzeichnet.“[13] Des Weiteren werden in dem Simulationsprogramm die Dichte ρ und der Verkehrsfluss Φ in einem Bereich von 100 Zellen in Fahrtrichtung nach der blauen Markierung gemessen und diese dann in Diagrammen dargestellt. In dem Diagramm rechts oben sieht man die Entwicklung des Verkehrsflusses Φ in

[13] Schadschneider, Andreas: *Verkehrsmodelle*

Abhängigkeit von der Zeit. Darunter wird das in Kapitel 3.3.3 vorgestellte Fundamentaldiagramm dargestellt, also die Abhängigkeit des Verkehrsflusses Φ von der Dichte ρ. Das dritte Diagramm zeigt die Abhängigkeit der Durchschnittsgeschwindigkeit *v* der Fahrzeuge von der Dichte ρ. In den beiden kleinen Diagrammen rechts unten wird die Geschwindigkeitsverteilung der Fahrzeuge und die Verteilung der Abstände zum jeweils vorausfahrenden Fahrzeug (*gap*) angezeigt.

Das Programm biete die Möglichkeit, vier Verkehrsmodelle zu simulieren, in dieser Arbeit wird jedoch nur das „Standard CA", also das Nagel-Schreckenberg-Modell, und das „VDR-Modell" (siehe Kapitel 5.1) verwendet.

Pro Sekunde laufen im Simulationsprogramm bei der Simulationsgeschwindigkeit 99% rund 200 simulierte Sekunden ab. Pro Sekunde werden 20 Messungen der Dichte ρ und des Verkehrsflusses Φ durchgeführt und in die Diagramme übertragen.

5. Erweiterung des Nagel-Schreckenberg-Modells

5.1 VDR-Modell

Wie schon in Kapitel 3.4 bemerkt, kann das Nagel-Schreckenberg-Modell noch verbessert werden. Hierzu lassen sich in empirischen Untersuchungen[14,15] Hysterese-Effekte feststellen, also Effekte, die auch nach Wegfall der Ursache weiter existieren. Konkret bedeutet das beim Stau, dass sich phasenseparierte Zustände herausbilden, d.h. ein Teil der Fahrzeuge steht entweder in einem relativ großen Stau oder hat freie Fahrt. Der Stau selbst bleibt aber bestehen und löst sich nicht im Laufe der Zeit auf.

Des Weiteren lässt sich in den empirischen Untersuchungen[16] feststellen, dass sich metastabile Hochflussphasen bilden können. Darunter versteht man besonders hohen Verkehrsfluss Φ, aus dem sich jedoch schnell ein Stau bilden kann. Bei einer metastabilen Hochflussphase hat das geglättete Fundamentaldiagramm (Abb. 11) bei Dichten zwischen ρ_1 und ρ_2 zwei Äste.

[14] Kerner, B. S., Rehborn, H.: *Experimental Properties of Phase Transitions in Traffic Flow*
[15] Helbing, Dirk: *Empirical traffic data and their implications for traffic modelling*
[16] Barlovic, R., Santen, L., Schadschneider, A., Schreckenberg, M.: *Metastable states in cellular automata for traffic flow*, Seite 3

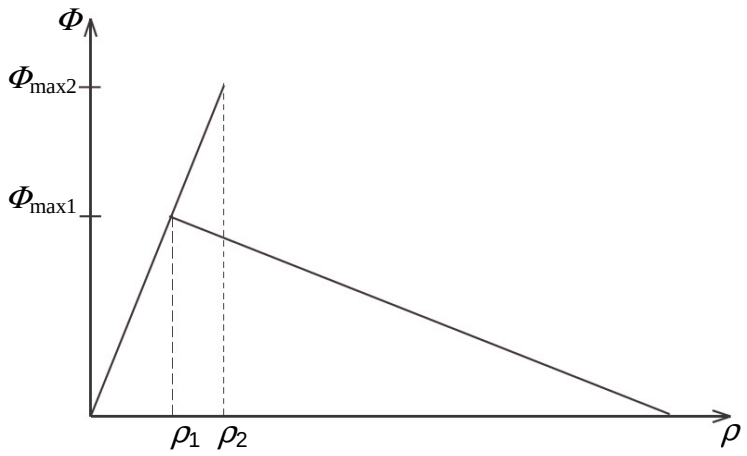

Abb. 11: Fundamentaldiagramm mit metastabilen Hochflusszuständen aus Schadschneider, Andreas: Physik des Straßenverkehrs, Seite 11

Zwischen der Dichte ρ_1 und ρ_2 kann man also aus dem Fundamentaldiagramm nicht direkt den zugehörigen Verkehrsfluss Φ ablesen. Da in diesem Bereich jedoch auch der höchste Verkehrsfluss Φ_{max2} möglich ist, ist dieser Bereich von besonderer Bedeutung.

Wenn man die Zeitkomponente hinzufügt, so lässt sich die Situation wie folgt beschreiben: Von links kommend wird bei größer werdender Dichte ρ der obere Ast beschritten, der Verkehrsfluss Φ steigt also bis zum Maximalwert Φ_{max2}, behält diesen Zustand auch für eine gewisse Zeit, verringert sich dann jedoch schlagartig auf den geringeren Wert an der Stelle ρ_2 („capacity drop“).[17] Daher nennt man den Bereich zwischen ρ_1 und ρ_2 metastabile Hochflussphase, da hier ein überdurchschnittlich hoher Verkehrsfluss Φ_{max2} vorliegen kann, der jedoch üblicherweise nicht von langer Dauer ist. Wenn die Dichte ρ wieder auf ρ_1 verringert wird, so erreicht der Verkehrsfluss Φ trotzdem nicht mehr den Wert Φ_{max2}.[18]

[17] Schadschneider, Andreas: *Physik des Straßenverkehrs*, Seite 10
[18] Schadschneider, Andreas: *Physik des Straßenverkehrs*, Seite 10

Folgendes empirisch gewonnene Diagramm zeigt ebenfalls eine solche metastabile Hochflussphase.

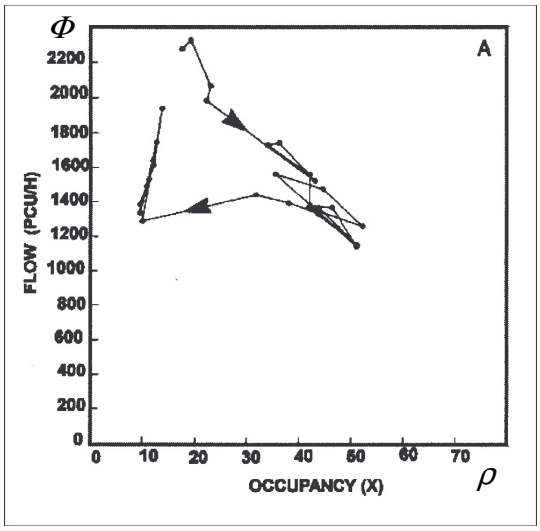

Abb. 12: empirisch bestimmte metastabile Hochflussphase und "capacity drop" aus Schadschneider, Andreas: Physik des Straßenverkehrs, Seite 12

Wie man hier erkennt, ist der Verkehrsfluss Φ zu Beginn der Messung bei seinem Maximum, bricht dann jedoch stark ein und erreicht nicht wieder den Hochflusszustand vom Beginn.
Um solche metastabilen Hochflusszustände zu erreichen, muss der Trödelfaktor p von der Geschwindigkeit des Fahrzeuges abhängig gemacht werden.[19] Im Englischen wird dies als „velocity-dependent randomization“ bezeichnet, daher der Name „VDR-Modell“. Die Abhängigkeit des Trödelfaktors p von der Geschwindigkeit v lässt sich wie folgt darstellen:

$$p(v) = \begin{cases} p_1 & v = 1 \\ p_2 & v = 2 \\ \dots & \dots \\ p_k & v = v_{\max} \end{cases}$$

Damit die gewünschten Hysterese-Effekte und die metastabilen Hochflusszustände erreicht werden, muss $p_1 > p_2 > \dots > p_k$ gewählt werden.[20] Anschaulich bedeutet dies im Stau, dass die Fahrer besonders beim Herausfahren aus einem Stau zum „Trödeln“ neigen, also nicht so schnell beschleunigen, wie sie könnten. Dies kann verschiedene Gründe haben: Zum einen weiß der Autofahrer nicht, ob der Stau wirklich schon zu Ende ist. Zum anderen erfordert die langsame Geschwindigkeit in einem Stau häufig nicht so starke Konzentration, wie im

[19] Barlovic, R., Santen, L., Schadschneider, A., Schreckenberg, M.: *Metastable States in cellular automata for traffic flow*, Seite 7
[20] Barlovic, R., Santen, L., Schadschneider, A., Schreckenberg, M.: *Metastable States in cellular automata for traffic flow*, Seite 8

fließenden Verkehr, so dass man am Ende eines Staus leicht zu Unaufmerksamkeit und Trödelverhalten neigt.

Folgende Aufnahme wurde mit Hilfe des VDR-Modells erstellt. Der Trödelparameter p wurde mit folgender Zuordnung verwendet:

$$p(v) = \begin{cases} 0{,}20 & v = 1 \\ 0{,}16 & v = 2 \\ 0{,}13 & v = 3 \\ 0{,}11 & v = 4 \\ 0{,}08 & v = 5 \\ 0{,}06 & v = 6 \\ 0{,}05 & v = 7 \end{cases}$$

Außerdem wurde $v_{max} = 7$ eingestellt, daraus ergibt sich folgendes Bild (Abb. 13):

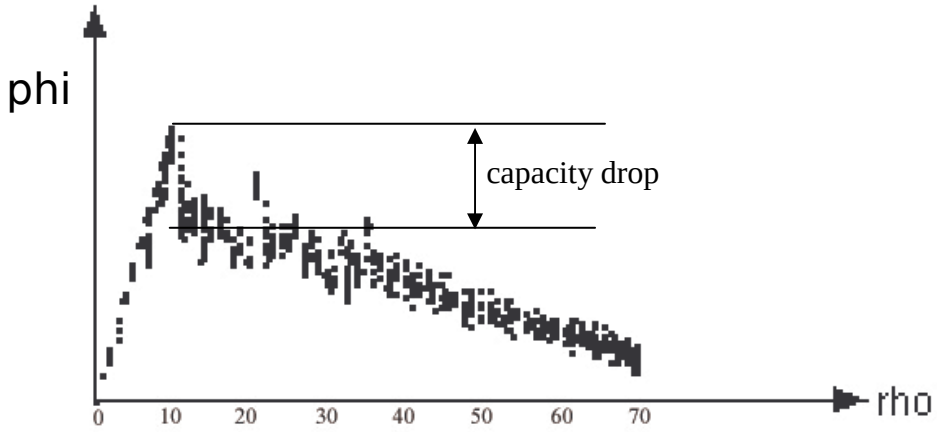

Abb. 13: VDR-Modell mit metastabiler Hochflussphase und "capacity drop", eigene Darstellung

Im Vergleich von Abb. 13 mit Abb. 11 wird deutlich, dass im Bereich um $\rho \approx 0{,}1$ eine metastabile Hochflussphase herrscht, und bei weiterer Erhöhung der Dichte ρ der Verkehrsfluss Φ schlagartig im „capacity drop“ zusammenbricht und danach gemittelt nur noch ⅔ Φ_{max} beträgt.

Der phasenseparierte Zustand, der auftritt, wenn nach einer metastabilen Hochflussphase der Verkehrsfluss Φ zusammenbricht, ist sehr gut in der folgenden Abbildung zu erkennen, die die prozentuale Verteilung der Fahrzeuge auf die Geschwindigkeiten v_0 bis v_7 von links nach rechts zeigt:

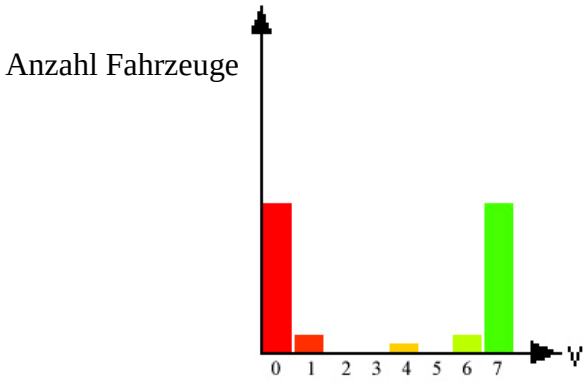

Abb. 14: Phasenseparierter Zustand im VDR-Modell, eigene Darstellung

Es ist offensichtlich, dass sich fest definierte Phasen herausgebildet haben, fast alle Fahrzeuge besitzen entweder die Geschwindigkeit $v = 0$ oder $v = v_{max} = 7$.

Man sieht also, dass es genügt, den Trödelparameter p von der Geschwindigkeit v abhängig zu machen, um die Effekte, die man in empirischen Untersuchungen (vgl. Abb. 6 und Abb. 12) festgestellt hat, im Simulationsmodell nachzubilden. Sowohl die gewünschten metastabilen Hochflusszustände, der „capacity drop“ und auch der phasenseparierte Zustand, der in Hysterese fortbesteht, sind allein auf die Abhängigkeit des Trödelfaktors p von der Geschwindigkeit v zurückzuführen.

5.2 Zwei- und dreispurige Straßen

Bisher wurde im Nagel-Schreckenberg-Modell nur der einspurige Verkehr betrachtet. Die Erweiterung zur Mehrspurigkeit liegt jedoch auf der Hand, allein zur Simulation von Autobahnen. Dazu muss man das Nagel-Schreckenberg-Modell bzw. das realistischere VDR-Modell um „Wechselregeln“[21] erweitern, die bestimmen, wann ein Fahrzeug die Spur wechselt und wann nicht. Da sich diese Arbeit auf Deutschland bezieht, muss auch das Rechtsfahrgebot und das Verbot, rechts zu überholen, in diese Wechselregeln mit aufgenommen werden. Die Simulation wird dann in Spurwechselregeln und Einspurdynamik nach VDR-Modell unterteilt. Damit kann das VDR-Modell weiter benutzt werden, es wird lediglich um eine Dimension erweitert.

[21] Schadschneider, Andreas: *Physik des Straßenverkehrs*, Seite 114

Die Spurwechselregeln setzten sich aus folgenden Aspekten zusammen: Der Wunsch, die Spur zu wechseln, da die eigene Spur als ungünstig empfunden wird, und der Sicherheitsgedanke beim Spurwechsel, so dass dieser ohne Kollision abläuft.

Eine typische Umsetzung der Spurwechselregeln ist der folgende Entwurf[22], Variablen mit Index 0 beziehen sich auf die Zielspur, alle anderen auf die eigene Spur:

- Die eigene Geschwindigkeit v ist höher als die eines vorausfahrenden Fahrzeugs auf der anderen Spur ($v < v_{0,front}$)
- Der Abstand zum vorausfahrenden Fahrzeug (gap) muss gering sein ($gap < l$)
- Der Abstand zum vorausfahrenden Fahrzeug auf der anderen Spur ($gap_{0,front}$) muss groß genug sein ($gap_{0,front} > l_{0,front}$)
- Der Abstand zum rückwärtigen Fahrzeug auf der anderen Spur ($gap_{0,back}$) muss groß genug sein ($gap_{0,back} > l_{0,back}$)

Wenn diese Regeln immer ausgeführt werden, kann dies dazu führen, dass sich ganze Fahrzeuggruppen bilden, die bei jedem Simulationsschritt als Einheit die Spur wechseln, da immer die andere Spur als günstiger angesehen wird.[23] Um dies zu Vermeiden, wird die so genannte Wechselwahrscheinlichkeit p_w eingeführt. Ein Spurwechsel wird dann, sobald die oben genannten Regeln erfüllt sind, mit dieser Wechselwahrscheinlichkeit p_w ausgeführt.
Folgende Grafik fasst die Vorgänge beim Spurwechsel zusammen, für das mit X markierte Fahrzeug sind die zugehörigen Abstände eingezeichnet:

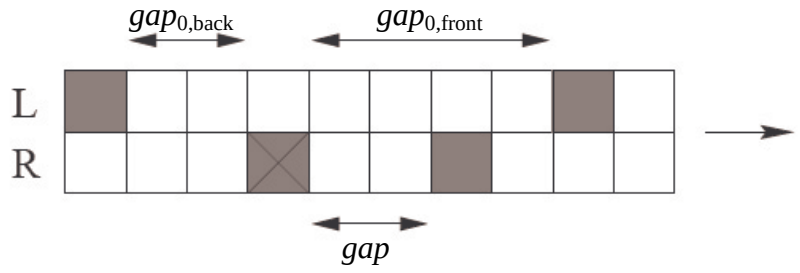

Abb. 15: Spurwechselregeln aus Schadschneider, Andreas: Physik des Straßenverkehrs, Seite 115

Trotz dieser Regeln ist er sehr schwer, ein „realistisches Verhalten“[24] mit den genannten Parameter l, $l_{0,front}$, und $l_{0,\ back}$ zu erreichen. Häufig ist starke „Pfropfenbildung“[25] zu beobachten,

[22] Rickert, M., Nagel, K., Schreckenberg, M., Latour, A.: *Two Lane Traffic Simulations Using Cellular Automata*, Seite 5
[23] Rickert, M., Nagel, K., Schreckenberg, M., Latour, A.: *Two Lane Traffic Simulations Using Cellular Automata*, Seite 4
[24] Schadschneider, Andreas: *Physik des Straßenverkehrs*, Seite 115
[25] Schadschneider, Andreas: *Physik des Straßenverkehrs*, Seite 115

so dass sich schon bei einem langsameren Fahrzeug viele andere Fahrzeuge sammeln und dieses langsamere Fahrzeug den ganzen Verkehrsfluss Φ bestimmt. Die realistische Simulation von mehrspurigem Verkehr ist also trotz der relativ einfachen Regeln nicht so leicht wie im Einspurverkehr zu erreichen.

6. Erklärung des Phänomens „Stau aus dem Nichts“

6.1 Abgrenzung und Erläuterung

Es gibt im Alltag vielerlei Arten von Stauauslösern, zum Beispiel Ampeln, Baustellen oder andere Engstellen. Das wissenschaftlich interessanteste Phänomen ist jedoch ein Stau, der ohne Außenfaktoren auf einer Landstraße oder Autobahn entsteht. Dieses Phänomen wird als „Stau aus dem Nichts“ bezeichnet.

Es lässt sich an folgendem Beispiel beschreiben: Eine Fahrzeugkolonne fährt mit relativ gleichmäßiger Geschwindigkeit, der Einfachkeit halber, auf einer Landstraße mit einer Spur. An einer Kreuzung möchte ein Verkehrteilnehmer abbiegen und verringert seine Geschwindigkeit. Die nachfolgenden Fahrzeuge werden dadurch ebenfalls zum Bremsen gezwungen, um den Sicherheitsabstand einhalten zu können. Dadurch wird eine Kettenreaktion ausgelöst, sodass die Fahrzeuge immer stärker bremsen müssen. Bei diesem Beispiel lässt sich nun beobachten, dass selbst nachdem das auslösende Fahrzeug abgebogen ist, es eine gewisse Zeit dauert, bis sich der Verkehrsfluss Φ wieder normalisiert hat. Die Zeit, in der das eigentliche Hindernis (der abbiegende Autofahrer) nicht mehr die Fahrbahn behindert, der Verkehrsfluss Φ aber dennoch nicht optimal ist, bezeichnet man als „Stau aus dem Nichts“.

Bei hohem Verkehrsaufkommen kann es passieren, dass in dem oben gewählten Beispiel der abbiegende Autofahrer vielleicht schon vor einer Viertelstunde abgebogen ist, an der Kreuzung auch keine Verkehrsbehinderung mehr vorliegt, sich dafür 3 km vor der Kreuzung ein Stau gebildet hat. Dieser Stau existiert dann, obwohl keine Behinderung mehr sichtbar ist. Er löst sich jedoch nicht auf, solange sich nicht weniger Fahrzeuge von hinten an den Stau anreihen, als vorne wieder losfahren. Man kann also von einer „Verselbstständigung“ eines Staus sprechen, der ohne eigentliche Ursache existieren kann.

Das eben beschriebene Szenario lässt sich auch im Simulationsprogramm beobachten:

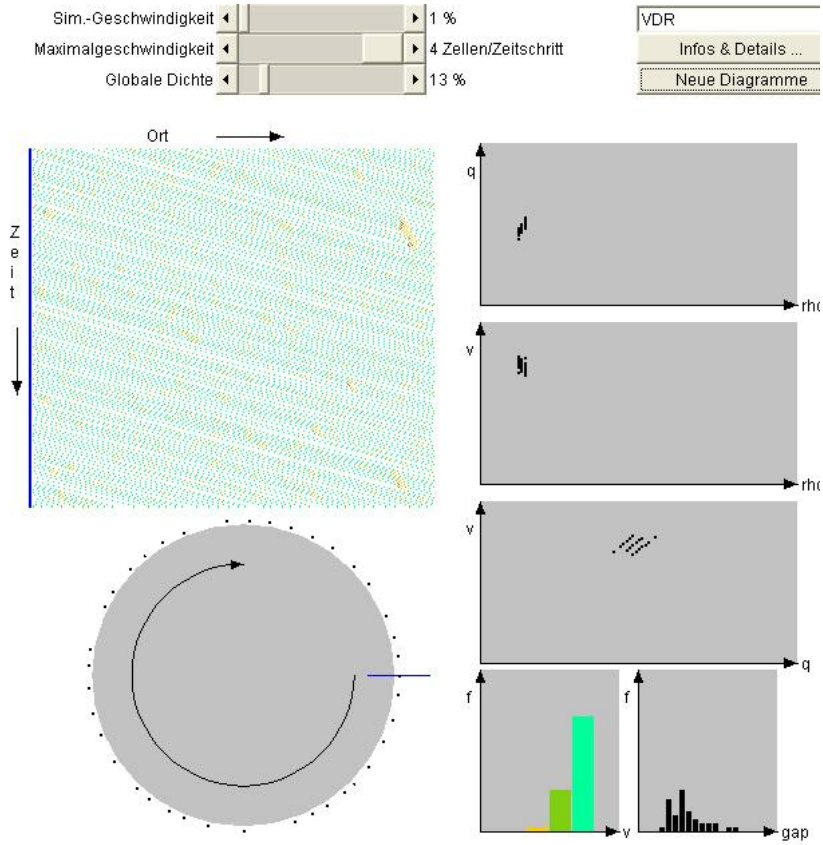

Abb. 16: Ungehinderter Verkehrsfluss, eigene Darstellung

Zu Beginn der Simulation in Abb. 16 fließt der Verkehrsfluss ungehindert. Dies kann man auch im ersten Diagramm rechts oben ablesen: Der Verkehrsfluss Φ ist recht hoch, auch die Geschwindigkeit v der einzelnen Fahrzeuge ist relativ hoch, wie im zweiten Diagramm deutlich wird.

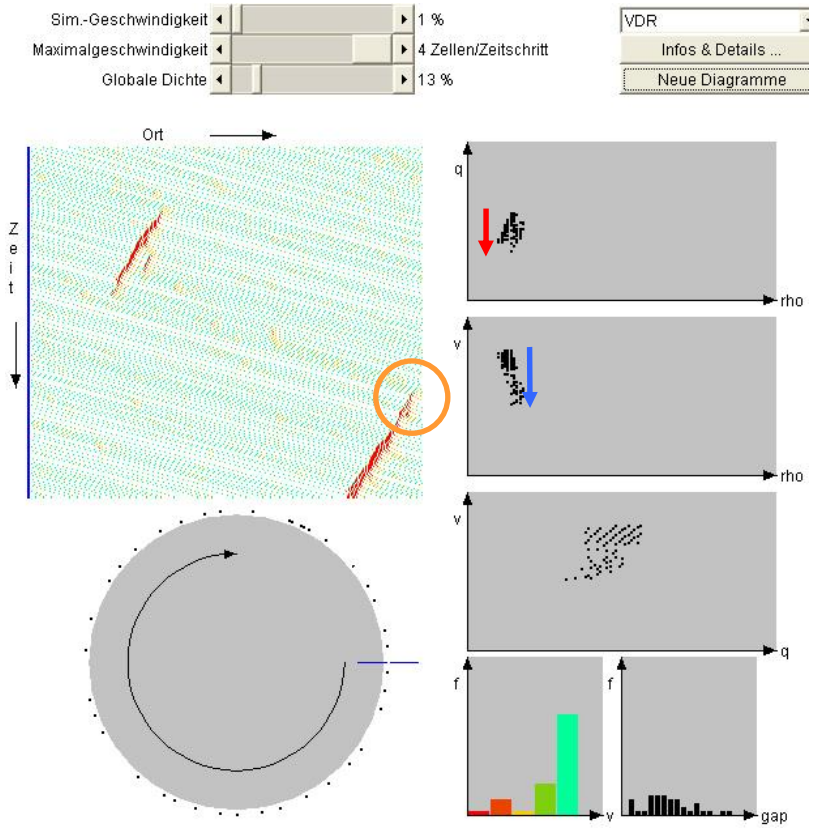

Abb. 17: Bildung eines Staus, eigene Darstellung

In Abb. 17 ist zu erkennen, wie sich der Rückstau bereits gebildet hat (orange Markierung), in dem oben genannten Beispiel ausgelöst durch das Abbremsen eines abbiegenden Fahrzeugs. Man kann auch im ersten Diagramm oben rechts erkennen, dass der Verkehrsfluss Φ nun bei Weitem nicht mehr so gut ist (roter Pfeil). Auch im zweiten Diagramm wird deutlich, dass sich die Geschwindigkeit v der Fahrzeuge fast halbiert hat (blauer Pfeil).

In Abb. 18 hat sich das Verkehrsgeschehen wieder normalisiert. Man sieht im Diagramm unten links, dass alle Fahrzeuge wieder mit hoher Geschwindigkeit fahren. Im Simulationsprogramm wird außerdem deutlich, dass die Dichte ρ und die Geschwindigkeit v wieder ähnlich zu denen in Abb. 16 sind.

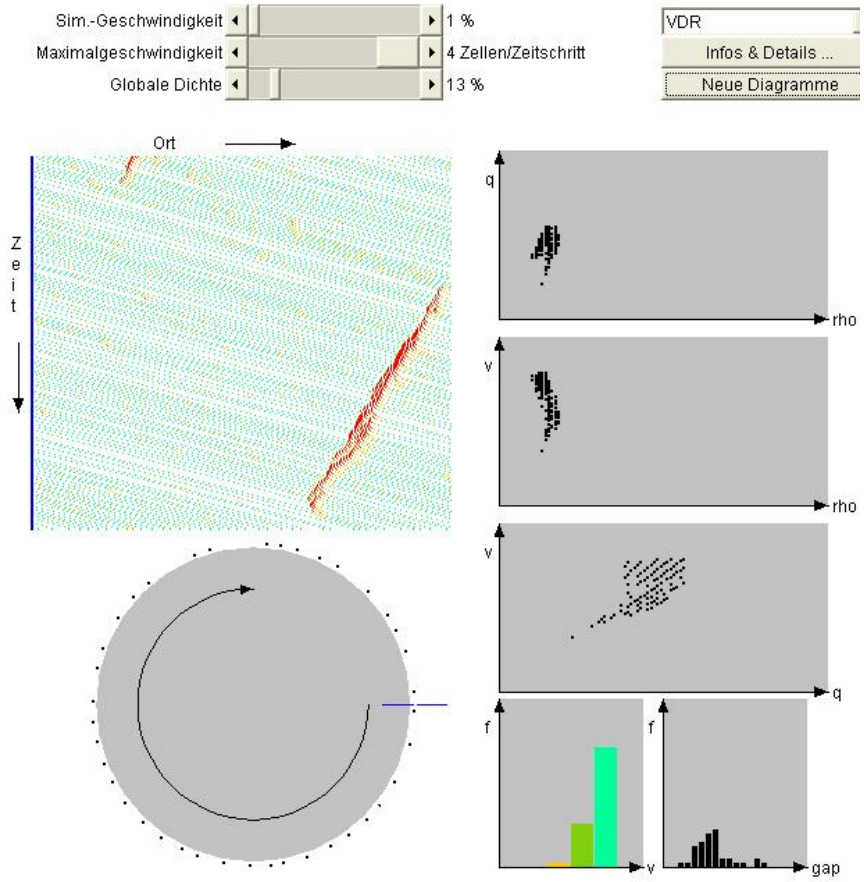

Abb. 18: Kompletter Stau mit Entstehung und Auflösung, eigene Darstellung

6.2 Die Bedeutung des Trödelfaktors *p*

Um die auslösenden Effekte, wie in dem obigen Beispiel das Verlangsamen und Abbiegen eines Fahrzeugs, zu berücksichtigen, wird im Nagel-Schreckenberg-Modell der Trödelfaktor *p* eingeführt. Unter dem Trödelfaktor *p* versteht man die Wahrscheinlichkeit, mit der ein Verkehrsteilnehmer seine Geschwindigkeit „grundlos“, also nicht aufgrund der Sicherheitsregel, verringert. Auch der Trödelfaktor *p* ist zwingend notwenig. Wenn er weggelassen würde, so könnte das Nagel-Schreckenberg-Modell nicht mehr als Verkehrssimulation einer Straße beschrieben werden.

In der folgenden Abbildung wird deutlich, warum der Trödelfaktor *p* Teil einer Simulation sein muss.

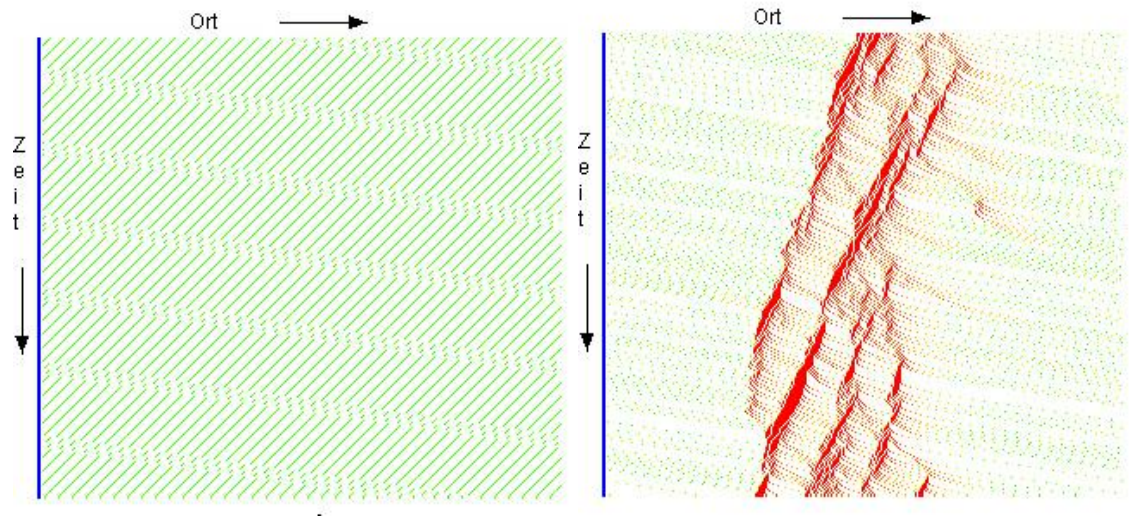

Abb. 19: Nagel-Schreckenberg-Modell mit v_{max} = 7, ρ = 0,12; links mit p = 0, rechts mit p = 0,4, eigene Darstellung

In Abb. 19 ist deutlich zu erkennen, dass das Modell ohne den Trödelfaktor *p* nicht mehr eine Simulation für den Straßenverkehr darstellt. Im Fall $p = 0$ (linkes Bild), was dem Vernachlässigen des Trödelfaktors *p* des Nagel-Schreckenberg-Modells entspricht, ist auch bei hoher Dichte ρ keine spontane Stauentstehung festzustellen.[26] Da empirische Untersuchungen das Gegenteil zeigen, ist der Trödelfaktor *p* also unerlässlich für die Simulation. Er ist jedoch gleichzeitig auch maßgeblich für die Entstehung von Staus verantwortlich (rechtes Bild).
Mit steigendem Trödelfaktor *p* verringert sich der Verkehrsfluss Φ immer weiter. Folgende Grafik zeigt die Abhängigkeit des Verkehrsflusses Φ vom Trödelfaktor *p*.

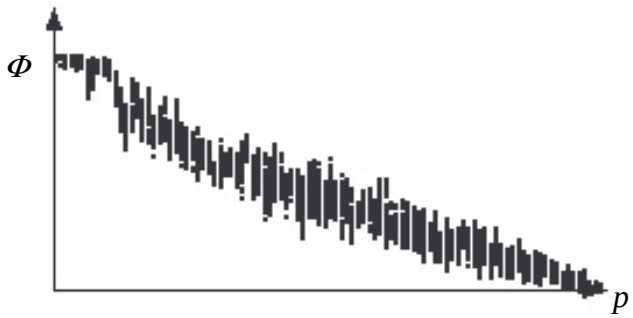

Abb. 20: Abhängigkeit von Verkehrsfluss und Trödelfaktor, eigene Darstellung

Zusammenfassend lässt sich also sagen, dass der Trödelfaktor *p* der entscheidende Auslöser für den „Staus aus dem Nichts“ ist.

[26] Schadschneider, Andreas: *Physik des Straßenverkehrs*, Seite 66

6.3 Mögliche Gründe für Trödelfaktor p im Straßenverkehr

Je größer die Neigung von Autofahrern zu einer ungleichmäßigen Fahrweise ist, desto höher ist ihr Trödelfaktor *p*. Im realen Straßenverkehr gibt es nun mehrere Beispiele, die zeigen, welches menschliche Verhalten der Trödelfaktor *p* enthält. Da der Trödelfaktor *p* für das Auslösen von Staus verantwortlich ist, sind dies Beispiele, wie in der Realität ein „Stau aus dem Nichts“ entsteht.

6.3.1 Unaufmerksames Fahren

Der einfachste Grund, warum man mit einem Trödelfaktor *p* bei Autofahrern rechnen muss, ist unaufmerksames Fahren. So kann es passieren, dass ein Verkehrsteilnehmer ohne Grund seine Geschwindigkeit verringert, also im wörtlichen Sinne „trödelt“, oder dass er aufgrund anderer Ablenkungen im Fahrzeug, wie anderer Fahrzeuginsassen, Radio oder Handy, seine Geschwindigkeit verringert.

6.3.2 Überhöhte Geschwindigkeit und zu dichtes Auffahren

Ein weiteres Phänomen, das durch den Trödelfaktor *p* abgedeckt ist, ist zu dichtes Auffahren. Wenn ein Verkehrsteilnehmer zu dicht auf das vor ihm fahrende Fahrzeug auffährt, muss er stärker als gewöhnlich bremsen und weit langsamer fahren, als das vor ihm fahrende Fahrzeug, um wieder ausreichend Sicherheitsabstand zu erreichen. Dadurch werden nachfolgende Fahrzeuge ebenfalls zum Bremsen gezwungen. Dieses langsame Fahren wird auch durch den Trödelfaktor *p* abgedeckt.

6.3.3 Spurwechsel bei dichtem Verkehr

Auch ein unbedachter Spurwechsel ist im Trödelfaktor *p* enthalten, zur Verdeutlichung wird folgende Situation beschrieben:
Eine zweispurige Straße ist bereits so ausgelastet, dass alle Fahrzeuge nur noch mit Sicherheitsabstand fahren (Abb. 21). Sobald jetzt ein Verkehrsteilnehmer (Fahrzeug C) die Spur wechselt, zum Beispiel um zu überholen, muss das nachfolgende Fahrzeug (Fahrzeug A) in der Spur, in die der Verkehrsteilnehmer gewechselt ist, abbremsen, um wieder den Sicherheitsabstand zu dem Fahrzeug C zu erreichen, das soeben die Spur gewechselt hat. Dadurch wird das nachfolgende Fahrzeug A zum „Trödeln“ gezwungen.

Eine Grafik veranschaulicht den Prozess:

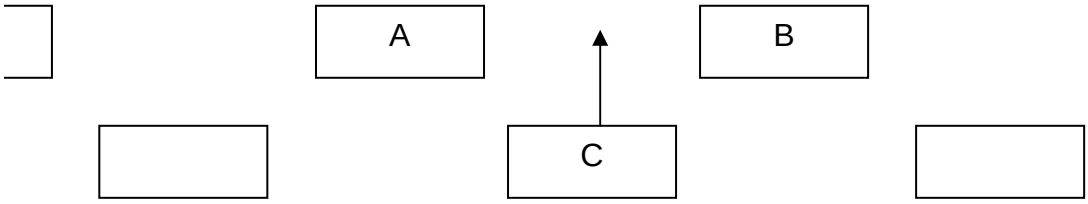

Abb. 21: Stark ausgelastete Straße, eigene Darstellung

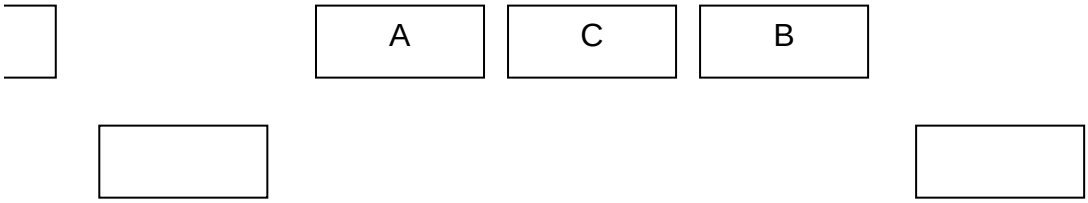

Abb. 22: Stark ausgelastete Straße nach dem Spurwechsel, eigene Darstellung

Somit ist es nicht immer zwingend notwendig, dass das Fahrzeug, welches seine Geschwindigkeit verringert (Fahrzeug A), auch ursächlich für den entstehenden Stau verantwortlich ist.

7. Vorstellung einiger Lösungsansätze zur Vermeidung von Stau

Nach der theoretischen Abhandlung des Nagel-Schreckenberg-Modells folgt nun die Vorstellung zweier anwendungsbezogener Lösungsansätze. Es wird versucht, die Wirkung eines generellen Tempolimit und elektronischer Verkehrsleitsysteme auf deutschen Autobahnen zur Stauvermeidung und zur Verkehrsflussoptimierung zu beurteilen.

7.1 Generelles Tempolimit

In der Politik wird immer wieder über die Einführung eines generellen Tempolimits auf deutschen Autobahnen von 100 km/h, vergleichbar mit den Vereinigten Staaten, oder 130 km/h, vergleichbar mit Italien und Frankreich, diskutiert. Dies soll vor allem die Sicherheit des Straßenverkehrs erhöhen und die Wahrscheinlichkeit von Staus verringern. Mit Hilfe des Simulationsprogramms lassen sich nun wissenschaftliche Aussagen über den Sinn und die Effektivität eines solchen Tempolimits hinsichtlich der Stauvermeidung treffen. Dazu wird die Entwicklung des Verkehrsflusses Φ über einen längeren Zeitraum einmal ohne Tempolimit, also

$v_{max} = 7$, d.h. ca. 200 km/h, mit dem Verkehrsfluss Φ mit dem Tempolimit 130 km/h, also $v_{max} = 5$, und Tempolimit 100 km/h, also $v_{max} = 4$ verglichen. Als Dichte ρ wurde $\rho = 0{,}11$ gewählt, der Trödelfaktor p wurde im VDR-Modell mit folgender Zuordnung verwendet: $p_0 = 0{,}30$, $p_1 = 0{,}24$, $p_2 = 0{,}19$, $p_3 = 0{,}15$, $p_4 = 0{,}11$, $p_5 = 0{,}12$, $p_6 = 0{,}15$, $p_7 = 0{,}20$;

Diese Zuordnung ist willkürlich festgelegt, beruht jedoch auf folgenden einfachen Regeln:

- → Autofahrer neigen bei langsamen Geschwindigkeiten eher zu Trödelverhalten.
- → Selbst wenn kein Tempolimit vorhanden ist, neigen vorsichtige Autofahrer trotzdem dazu, nicht mehr mit maximal möglicher Geschwindigkeit v_{max} zu fahren, daher $p_7 = 0{,}2$.

Die Verkehrssimulation zeigt folgendes Ergebnis:

v_{max}	4	5	6	7
Φ	23	32	26	26

Abb. 23: Verkehrsfluss Φ in Abhängigkeit von v_{max}, eigene Darstellung

Bei $v_{max} = 4$ stellt sich ein nahezu konstanter Verkehrsfluss $\Phi = 23$ Fahrzeuge/Minute ein. Dieser ist sehr beständig, es stellen sich keine Staus ein. Künstlich hervorgerufene Staus lösen sich schon nach kurzer Zeit wieder auf.

Bei $v_{max} = 5$ zeigt sich ein Verkehrsfluss $\Phi = 32$ Fahrzeuge/Minute, jedoch treten manchmal kleinere Staus auf, die mit rund 2,5 simulierten Minuten doppelt so viel Zeit wie bei $v_{max} = 4$ benötigen, bis sie sich auflösen.

Schon bei $v_{max} = 6$ stellt sich nach längerer Zeit immer wieder der Verkehrsfluss $\Phi = 26$ Autos/Minute ein. Empirische Untersuchungen ergeben, dass sich nach rund 30 bis 60 simulierten Minuten ein phasenseparierter Zustand herausbildet, in dem ein dauerhafter Stau den Verkehrsfluss Φ stark limitiert. Dieser Stau löst sich selbst nach vier bis fünf simulierten Stunden nicht auf.

Bei $v_{max} = 7$ stellt sich der phasenseparierte Zustand sogar in noch kürzerer Zeit ein. Ein optimaler Verkehrsfluss Φ ist bei gegebener Dichte $\rho = 0{,}11$ nur bis $v_{max} = 5$ möglich.

Da der Verkehrsfluss ohne Tempolimit häufiger zusammenbricht, ist es im Hinblick auf die Stauvermeidung durchaus sinnvoll, generelle Tempolimits einzuführen. So kann ein viel konstanterer und auf Dauer höherer Verkehrsfluss Φ garantiert werden. Selbst wenn ohne Tempolimits kurzzeitig, während metastabilen Hochflussphasen, ein größerer Verkehrsfluss Φ erreicht wird, bricht dieser schnell zusammen und ein dauerhaft phasenseparierter Zustand mit Stau entsteht. Im Hinblick auf die Stauvermeidung und Verkehrsoptimierung der Straßen stellt ein Tempolimit von 130 km/h eine sinnvolle Maßnahme dar.

7.2 Verkehrsleitsysteme

Ein weiterer Ansatz, den Verkehrsfluss Φ einer Straße zu verbessern, ist die Installation von Verkehrsleitsystemen. Die Idee ist, die metastabilen Hochflusszustände zu nutzen und gleichzeitig die Entstehung von dauerhaft phasenseparierten Zuständen zu vermeiden. Verkehrsleitsysteme können zu diesem Zweck das gültige Tempolimit variabel festlegen und ermöglichen somit einen noch höheren Verkehrsfluss Φ. Ein simples Beispiel ist hierfür der Lincoln-Tunnel zwischen Manhattan und New Jersey, bei dem „das hohe Verkehrsaufkommen […] eine weitere Tunnelröhre notwendig gemacht [hätte]. Stattdessen sind Verkehrsingenieure auf die Idee gekommen, den Zusammenbruch des Hochflußastes durch Erzeugung von Fahrzeugpaketen (‘Platoons’) zu verhindern. Die einzelnen Pakete sind dabei durch größere Lücken getrennt. Dies führt zu einem Abschneiden der Kettenreaktionen, die typischerweise einen Stau entstehen lassen.“[27] In dem genannten Tunnel wurde dies mit Ampeln realisiert und eine Verbesserung des Verkehrsflusses Φ um 20%[28] konnte festgestellt werden. Somit können die Hochflusszustände aufrechterhalten werden.

Dieses Vorgehen lässt sich auch auf größere Verkehrsleitsysteme übertragen, wie sie z. T. schon in Deutschland installiert sind. Hierbei wird nicht eine Ampel auf Rot geschaltet, sondern der Verkehr wird im Ganzen auf 100 km/h oder 80 km/h abgebremst, sobald ein Stau entsteht.

Auch im Simulationsprogramm lässt sich dieses Vorgehen nachstellen. Startet man das VDR-Modell mit $\rho = 0{,}11$, $v_{max} = 6$ und folgender Zuordnung für den Trödelparameter p: $p_0 = 0{,}30$, $p_1 = 0{,}24$, $p_2 = 0{,}19$, $p_3 = 0{,}15$, $p_4 = 0{,}11$, $p_5 = 0{,}12$, $p_6 = 0{,}15$, $p_7 = 0{,}20$, so entsteht sich nach einiger Zeit des metastabilen Hochflusszustandes ein phasenseparierter Zustand mit Stau und Freifluss-Phase. Verringert man nun das Tempolimit $v_{max} = 7$ auf $v_{max} = 4$, so löst sich der Stau sofort auf, der Verkehrsfluss Φ stabilisiert sich und der Verkehr normalisiert sich. Selbst, wenn dann das Tempolimit v_{max} wieder angehoben wird, auf $v_{max} = 6$ oder $v_{max} = 7$, entsteht nicht sofort wieder ein neuer Stau, sondern es herrscht ein Hochflusszustand.

[27] Schadschneider, Andreas: *Physik des Straßenverkehrs*, Seite 110
[28] Schadschneider, Andreas: *Physik des Straßenverkehrs*, Seite 110

Die folgende Abb. 24 zeigt den metastabilen Hochflusszustand (orange Markierung), den „capacity drop“ und den Beginn eines Staus (rote Markierung), den Zeitpunkt des Einsetzen des Tempolimits $v_{max} = 4$ (blaue Markierung) und die folgende Auflösung des Staus.

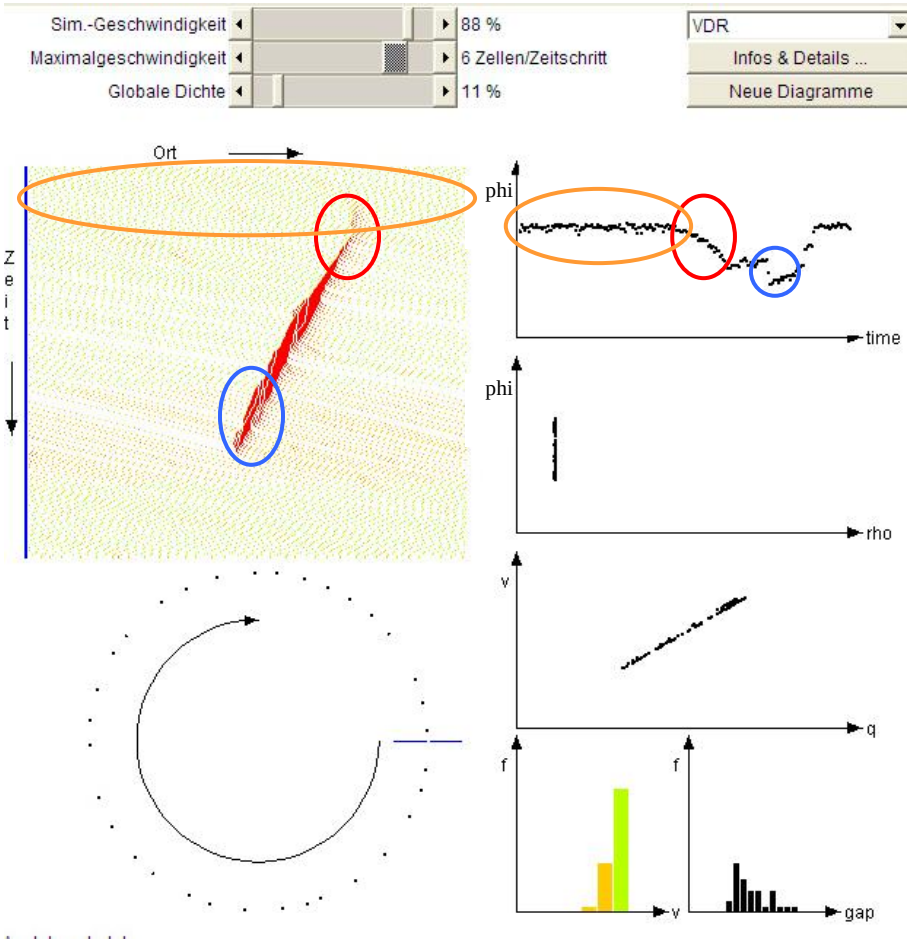

Abb. 24: Aufrechterhaltung von Hochflussphasen, eigene Darstellung

Über einen längeren Zeitraum hinweg betrachtet lassen sich mit Verkehrsleitsystemen also die metastabilen Hochflusszustände nutzen, ohne den Nachteil der Staubildung und der Hysterese des phasenseparierten Zustands zu erhalten.

8. Persönliches Fazit

Zusammenfassend lässt sich also feststellen, dass vor allem menschliche Unzulänglichkeiten, wie Unaufmerksamkeit oder Ablenkung, den „Stau aus dem Nichts“ verursachen. Dieses Verhalten lässt sich dennoch realistisch mit Hilfe des Trödelparameters p beschreiben und simulieren. Diese Methode wird unter anderem von der Uni Duisburg dazu genutzt, im Projekt „autobahn.nrw“[29] eine Stauprognose für die kommenden 60 Minuten zu erstellen.

Wie sich anhand der Simulationsergebnisse gezeigt hat, ist die Einführung eines Tempolimits von 130 km/h ein einfaches und probates Mittel zu Stauvermeidung. Da durch das Tempolimit die Dynamik des Verkehrsgeschehens reduziert wird, wird in gleichem Maße auch das Risiko von Staus reduziert.

Verkehrsleitsysteme stellen eine weitergehend ausgelegte und anpassungsfähigere Lösungsmöglichkeit für den „Stau aus dem Nichts“ dar, da hier das Tempolimit genau auf das Verkehrsgeschehen abgestimmt werden kann. Damit wird noch effektiver die Staubildung verhindert und gleichzeitig werden die metastabilen Hochflusszustände genutzt, um den Verkehrsfluss Φ zu steigern.

Die ideale Stauvermeidung wäre, wenn man den Trödelfaktor p der Menschen verringern oder abschaffen könnte. Dies wird tatsächlich in einem Forschungsprojekt versucht. Allerdings versucht man natürlich nicht, allen Menschen die Unaufmerksamkeit abzutrainieren, sondern die Autos intelligenter zu gestalten, so dass über Kommunikation von Auto zu Auto Informationen über die aktuelle Verkehrslage weitergegeben werden (z.B. per Funk). Diese werden dann vom einzelnen Fahrzeug genutzt, um einen harmonischeren Verkehrsfluss Φ herzustellen.

Ein weitergehender Ansatz ergibt sich aus folgender Fragestellung: Warum gibt es im Bahnnetz keine Staus? Ein erster Gedanke wäre, dass nicht so viele Menschen Züge als Verkehrsmittel benutzen. Doch bei weiteren Überlegungen fällt auf, dass im Schienennetz der Sicherheitsabstand wegfällt, zumindest zu einem großen Teil. Im Bahnverkehr muss schließlich nur zwischen den einzelnen Zügen Sicherheitsabstand gehalten werden, nicht zwischen jedem Abteil. Dadurch hat man pro Person, die das Verkehrsmittel nutzt, einen viel geringeren Sicherheitsabstand und kann so einen höheren Verkehrsfluss Φ insgesamt erreichen.

[29] Schreckenberg, Michael et. al.: *BAB NRW,* Seite „Profil“

Eine Überlegung, diesen Ansatz zu nutzen, ist die Bildung von so genannten „Adhoc-Zügen“. Man kann es sich ungefähr so vorstellen: Wenn zu Beginn der Sommerferien viele Autofahrer in den Süden über den Brennerpass fahren wollen, so fahren sie zumindest einen großen Teil ihres Weges parallel. Wenn man in dieser Zeit, in der die Fahrzeuge parallel fahren, Fahrzeugpakete aus zehn Autos bildet, so spart man 9/10 des Sicherheitsabstandes und kann den Verkehrsfluss Φ dramatisch erhöhen.

Doch wie genau sollen die „Adhoc-Züge“ aufgebaut sein? Folgende Aspekte sind bei der Konzeption wichtig: Der Zug muss sich schnell bilden können, damit keine zeitlichen Nachteile für die Verkehrsteilnehmer entstehen. Er muss sehr flexibel sein, um den individuellem Wünschen der Verkehrsteilnehmer gerecht zu werden. Er muss so konzipiert sein, dass den Verkehrsteilnehmern keine Nachteile hinsichtlich ihrer Sicherheit entstehen und er darf keine Umrüstung bei den Verkehrsteilnehmern erwarten, da ansonsten die breite Akzeptanz, die wichtig für den Erfolg ist, nicht möglich ist. Gerade die ersten beiden Aspekte sind wichtig, da diese die entscheidenden Unterschiede zu den herkömmlichen Autozügen darstellen.

Denkbar ist z.B. Folgendes: Der ganze Adhoc-Zug wird von einer LKW-Zugmaschine gezogen, die einzelnen Autos sind dann an den Vorderrädern an den Zug angehängt. Dadurch ist bei den Fahrzeugen keine Umrüstung notwendig, außerdem kann man die Reifen der Fahrzeuge nutzen und muss nicht eine Transportkonstruktion wie bei einem Autotransporter für Neuwagen bauen. Folgende einfache Skizze zeigt einen möglichen Adhoc-Zug:

Abb. 25: grobe Skizze eines Adhoc-Zuges, eigene Darstellung

Ein zukünftiger Lösungsansatz schlägt eine elektronische Kopplung der Verkehrsteilnehmer vor. Die mechanische Verbindung der Adhoc-Züge wird durch eine rein elektronische Kommunikation zwischen den Fahrzeugen ersetzt, so dass diese dann mit einem Abstand von einem Meter hintereinander fahren können. Dadurch wird ebenso der Sicherheitsabstand eingespart und der Verkehrsfluss Φ gesteigert.

9. Anhang

9.1 Abbildungsverzeichnis

Abb. 1: Beispielhafte Erklärung, Ausgangssituation, eigene Darstellung 3

Abb. 2: Beispielhafte Erklärung, nach einer Runde, eigene Darstellung 4

Abb. 3: Beispielhafte Erklärung, nach Runde 2, eigene Darstellung 4

Abb. 4: Empirische Verkehrsmessung aus J. Treiterer: Ohio State Technical Report, eigene Bearbeitung 6

Abb. 5: Empirische Daten im Nagel-Schreckenberg-Modell, eigene Darstellung 7

Abb. 6: Empirisches Fundamentaldiagramm aus Hall, F. L., Allen, B. L., Gunter, M. A.: Empirical analysis of freeway flow-density relationships 8

Abb. 7: Fundamentaldiagramm Nagel-Schreckenberg-Modell, eigene Darstellung 9

Abb. 8: Schematisches Fundamentaldiagramm Nagel-Schreckenberg-Modell aus Schadschneider, Andreas: Physik des Straßenverkehrs, Seite 11 9

Abb. 9: Schema des Kerns des Simulationsprogramms, eigene Darstellung 10

Abb. 10: Simulationsprogramm CA_einspur, eigene Darstellung 12

Abb. 11: Fundamentaldiagramm mit metastabilen Hochflusszuständen aus Schadschneider, Andreas: Physik des Straßenverkehrs, Seite 11 14

Abb. 12: empirisch bestimmte metastabile Hochflussphase und "capacity drop" aus Schadschneider, Andreas: Physik des Straßenverkehrs, Seite 12 15

Abb. 13: VDR-Modell mit metastabiler Hochflussphase und "capacity drop", eigene Darstellung 16

Abb. 14: Phasenseparierter Zustand im VDR-Modell, eigene Darstellung 17

Abb. 15: Spurwechselregeln aus Schadschneider, Andreas: Physik des Straßenverkehrs, Seite 115 18

Abb. 16: Ungehinderter Verkehrsfluss, eigene Darstellung 20

Abb. 17: Bildung eines Staus, eigene Darstellung 21

Abb. 18: Kompletter Stau mit Entstehung und Auflösung, eigene Darstellung 22

Abb. 19: Nagel-Schreckenberg-Modell mit $v_{max} = 7$, $\rho = 0{,}12$; links mit $p = 0$, rechts mit $p = 0{,}4$, eigene Darstellung 23

Abb. 20: Abhängigkeit von Verkehrsfluss und Trödelfaktor, eigene Darstellung 23

Abb. 21: Stark ausgelastete Straße, eigene Darstellung 25

Abb. 22: Stark ausgelastete Straße nach dem Spurwechsel, eigene Darstellung 25

Abb. 23: Verkehrsfluss Φ in Abhängigkeit von v_{max}, eigene Darstellung 26

Abb. 24: Aufrechterhaltung von Hochflussphasen, eigene Darstellung 28

Abb. 25: grobe Skizze eines Adhoc-Zuges, eigene Darstellung 30

9.2 Literaturverzeichnis

Barlovic, R., Santen, L., Schadschneider, A., Schreckenberg, M. (1998): *Metastable states in cellular automata for traffic flow*, Eur. Phys. J. B **5**, Seite 793 (1998)

Chowdhury, D., Santen, L., Schadschneider, A., (2000): *Statistical Physics of Vehicular Traffic and Some Related Systems*, Physics Reports 329, Seite 199 (2000)

Hall, F. L., Allen, B. L., Gunter, M A. (1986): *Empirical analysis of freeway flow-density relationships*, Transp. Res. A20, Seite 197 (1986)

Helbing, Dirk (1997): *Empirical traffic data and their implications for traffic modelling*, Phys. Rev. E55, R25 (Issue 1 – January 1997)

Helbing, Dirk (2001): *Traffic and related self-driven many-particle systems*, Reviews in Modern Physics 73, Seite 1067 (2001)

Kerner, B. S., Rehborn, H. (1997): *Experimental Properties of Phase Transitions in Traffic Flow*, Phys. Rev. Lett. 79, Seite 4030 (Issue 20 – 17 November 1997)

Nagel, K., Schreckenberg, M. (1992): *A cellular automaton model for freeway traffic*, J. Physique I **2**, Seite 2221 (1992)

Rickert, M., Nagel, K., Schreckenberg, M., Latour, A. (1996): *Two Lane Traffic Simulations Using Cellular Automata*, Physica A 231, Seite 534 (1996)

Schadschneider, Andreas (2000): *Verkehrsmodelle*, WWW-Dokument vom 14. Januar 2006, (http://www.thp.uni-koeln.de/~as/Mypage/verkehr.html)

Schadschneider, Andreas (2004): *Physik des Straßenverkehrs*, Vorlesungsscript Sommersemester 2004, WWW-Dokument vom 15. Januar 2005, (http://www.thp.uni-koeln.de/~as/Mypage/PSfiles/verkehr.pdf)

Schadschneider, Andreas (2005): *Grundlagen der Verkehrsdynamik*, Gastvortrag Universität Heidelberg, 2005

Schreckenberg, Michael et. al. (2005): *BAB-NRW project*, WWW-Dokument vom 21. Januar 2005, (http://www.traffic.uni-duisburg.de/homepage/babnrw/profile.html)

Treiterer, J. (1975): *Ohio State Technical Report* No. PB 246 094 (1975)

9.3 Listing des Simulationsprogramms

9.3.1 Datei CAmodel.java

```
import java.awt.*;
import java.applet.*;
import java.util.*;

////////////////////////////////////////////////////////
// Application of the standard CA Nagel Schreckenberg model, VDR, T² et. al.
// Simulation of traffic flow
////////////////////////////////////////////////////////

// B. Eisenblätter, L. Neubert and J. Wahle 1998
// C. Settgast 2005/2006
// last modified 01/23/2006

public class CAmodel extends Applet implements Runnable{

  // language for displaying
  String used_language;
  public int language;
  public final int MAXSPEED = 8;

  // Init values of the scrollbars
  private final int MAXSPEED_INIT = 8;
  private final int DENS_INIT = 20;
  private final int SPEED_INIT = 100;

  // Init values of the deceleration probabilities
  private final double P_DEC_INC_STS = 0.5;
  private final double P_DEC_INC_TSQR = 0.5;
  private final double P_DEC = 0.2;

  // Dinstance between two updates of the diagrams
  private final int UPDATE_DIAGRAM = 10;

  // old values for comparision
  private int maxspeed_old = 0;
  private int density_old = 0;
  private int simspeed_old = 1;

  // The panel for displaying everything
  private Panel p;

  // Labels of the scrollbars
  private Label label_maxspeed_name;
  private Label label_density_name;
  private Label label_simspeed_name;
  private Label label_maxspeed;
  private Label label_density;
  private Label label_simspeed;

  // The scrollbars themselves
  private Scrollbar scrollbar_maxspeed;
  private Scrollbar scrollbar_density;
  private Scrollbar scrollbar_simspeed;

  // Two buttons
  private Button clear;
  private Button vdr;

  // The pull-down-menu
  private Choice model;

  // The thread
  private Thread runner;
```

```
  // The canvas for the painting
  private RoadCanvas canvas;

  // Different types of the CA model
  // 0=Standard
  // 1=T²
  // 2=VDR
  // 3=Fukui-Ishibashi
  public int modeltype;

  // The message boxes for the p_dec-tuning
  private MessageBox vdr_box_0;
  private MessageBox vdr_box_1;
  private MessageBox vdr_box_2;
  private MessageBox vdr_box_3;

  // The several deceleration prob's
  private double[] p_dec;
  public double global_p_dec;
  private double global_p_dec_inc_sts;
  private double global_p_dec_inc_tsqr;
  private int maxspeed;

  public static void main(String[] args){
  }

  ////////////////////////////////////
  // Initializing the applet

  public void init(){

    // find out the language to use
    used_language = getParameter("LANGUAGE");
    // Default language is English
    language = 0;
    if (used_language.equals("German"))
      language = 1;

    // Assign the deceleration prob's
    p_dec = new double[MAXSPEED+1];
    global_p_dec = P_DEC;
    global_p_dec_inc_sts = P_DEC_INC_STS;
    global_p_dec_inc_tsqr = P_DEC_INC_TSQR;
    p_dec[0] = 0.25;
    p_dec[1] = 0.20;
    p_dec[2] = 0.17;
    p_dec[3] = 0.14;
    p_dec[4] = 0.12;
    p_dec[5] = 0.10;
    p_dec[6] = 0.08;
    p_dec[7] = 0.07;
    p_dec[8] = 0.06;

    // for (int v=0;v<=MAXSPEED;v++)
    // p_dec[v] = global_p_dec;


    maxspeed = MAXSPEED_INIT;

    // scrollbars of simulation speed, maxspeed and global density
    scrollbar_simspeed = new Scrollbar(Scrollbar.HORIZONTAL,SPEED_INIT,1,1,100);
    scrollbar_maxspeed = new
Scrollbar(Scrollbar.HORIZONTAL,MAXSPEED_INIT,1,1,MAXSPEED);

    scrollbar_density = new Scrollbar(Scrollbar.HORIZONTAL,DENS_INIT,1,0,100);


    // Init the message boxes, the panel and the buttons
```

```
    vdr_box_0 = new MessageBox(global_p_dec,language);
    vdr_box_1 = new MessageBox(global_p_dec,global_p_dec_inc_tsqr,language);
    vdr_box_2 = new MessageBox(p_dec,maxspeed,MAXSPEED+1,language);
    vdr_box_3 = new MessageBox(global_p_dec,maxspeed,MAXSPEED+1,language);

    p = new Panel();
    switch(language){
    case 1:
      clear = new Button("Neue Diagramme");
      vdr = new Button("Infos & Details ...");
      break;
      // Default language is English
    default:
      clear = new Button("Clear Diagram");
      vdr = new Button("Info & Details ...");
      break;
    }

    // Set the model type, starting with the Standard-Nagel-Schreckenberg-
    // configuration
    modeltype = 0;

    // Pull-Down-menu and the filling of the panels
    model = new Choice();
    model.addItem("Standard CA");
    model.addItem("Takayasu");
    model.addItem("VDR");
    model.addItem("Fukui-Ishibashi");

    p.setLayout(new GridLayout(4,4));
    switch(language){
    case 1:
      label_simspeed_name = new Label("Sim.-Geschwindigkeit");
      label_maxspeed = new Label((new Integer(MAXSPEED_INIT).toString())+"
Zellen/Zeitschritt");
      label_maxspeed_name = new Label("Maximalgeschwindigkeit");
      label_density_name = new Label("Globale Dichte");
      break;
      // Default language is English
    default:
      label_simspeed_name = new Label("Simulation speed");
      label_maxspeed = new Label((new Integer(MAXSPEED_INIT).toString())+"
sites/timestep");
      label_maxspeed_name = new Label("Maximum velocity");
      label_density_name = new Label("Global Density");
      break;
    }

    p.add(label_simspeed_name);
    label_simspeed_name.setAlignment(Label.RIGHT);
    p.add(scrollbar_simspeed);
    label_simspeed = new Label((new Integer(SPEED_INIT).toString())+" %");
    p.add(label_simspeed);
    p.add(model);

    p.add(label_maxspeed_name);
    label_maxspeed_name.setAlignment(Label.RIGHT);
    p.add(scrollbar_maxspeed);
    p.add(label_maxspeed);
    p.add(vdr);

    p.add(label_density_name);
    label_density_name.setAlignment(Label.RIGHT);
    p.add(scrollbar_density);
    label_density = new Label((new Integer(DENS_INIT).toString())+" %");
    p.add(label_density);
    p.add(clear);

    for (int i=0;i<4;i++)
```

```
    p.add(new Label("  "));

  // Init the canvas
  canvas = new RoadCanvas(getFirstDensity(),language);

  // Design the table
  setLayout(new BorderLayout());
  add("North",p);
  add("Center",canvas);
}

//////////////////////////////////////
// Starting the applet

public void start(){
  if (runner == null){
    runner = new Thread(this);
    runner.start();
  }
  else
    if (runner.isAlive())
     runner.resume();
  clear.requestFocus();
}

public void stop(){
  runner.suspend();
}

public void destroy(){
}

//////////////////////////////////////
// Make the applet running

public void run(){
  // loop counter
  // indicates whether the diagrams have to be plotted
  int counter = 0;
  while (true){
    if (vdr_box_0.isRunning){
     if (global_p_dec != vdr_box_0.getlocal_p_dec()){
       global_p_dec = vdr_box_0.getlocal_p_dec();
       for (int v=0;v<=maxspeed;v++)
         p_dec[v] = global_p_dec;
     }
    }
    else{
      if (vdr_box_1.isRunning){
        if (global_p_dec != vdr_box_1.getlocal_p_dec()){
          global_p_dec = vdr_box_1.getlocal_p_dec();
          for (int v=0;v<=maxspeed;v++)
            p_dec[v] = global_p_dec;
        }
        if (global_p_dec_inc_tsqr != vdr_box_1.getlocal_p_dec_inc())
          global_p_dec_inc_tsqr = vdr_box_1.getlocal_p_dec_inc();
      }
      else{
        if (vdr_box_2.isRunning){
          for (int v=0;v<=maxspeed;v++)
            p_dec = vdr_box_2.getlocal_p_dec_array();
        }
        else{
          if (vdr_box_3.isRunning){
            if (global_p_dec != vdr_box_3.getlocal_p_dec()){
              global_p_dec = vdr_box_3.getlocal_p_dec();
              for (int v=0;v<=maxspeed;v++)
                p_dec[v] = 0.0;
              p_dec[maxspeed] = global_p_dec;
```

```
              }
            }
            else{
              model.enable(true);
              scrollbar_maxspeed.enable(true);
            }
          }
        }
      }
      maxspeed = getMaxspeed();

canvas.update(p_dec,getDensity(),counter,modeltype,global_p_dec_inc_sts,global_p_de
c_inc_tsqr,maxspeed);
      if (++counter >= UPDATE_DIAGRAM)
       counter = 0;
      try{
       Thread.sleep(getSimSpeed());
      }
      catch(InterruptedException e){}
    }
  }


  ////////////////////////////////////
  // Handling the events

  public boolean action(Event evt, Object arg){

    // Buttons
    if (evt.target instanceof Button){
      if (evt.target==clear){
       canvas.ClearDiagrams();
       return true;
      }
      if (evt.target==vdr){
       model.enable(false);
       // Model types:
       // 0: Standard model
       // 1: T²-model
       // 2: VDR-model
       // 3: Fukui-Ishibashi-model
       switch(modeltype){
       case 0:{
         vdr_box_0.show(global_p_dec,language);
         break;
       }
       case 1:{
         vdr_box_1.show(global_p_dec,global_p_dec_inc_tsqr,language);
         break;
       }
       case 2:{
         scrollbar_maxspeed.enable(false);
         vdr_box_2.show(p_dec,maxspeed,MAXSPEED+1,language);
         break;
       }
       case 3:{
         vdr_box_3.show(global_p_dec,maxspeed,MAXSPEED+1,language);
         break;
       }
       }
       return true;
      }
    }

    // Pull-Down Menu
    if (evt.target instanceof Choice){
      if ("Standard CA".equals(arg)){
       // Setting the model parameters:
       // p_dec <> p_dec(v)
```

```
        modeltype = 0;
        global_p_dec = P_DEC;
        for (int v=0;v<=maxspeed;v++)
          p_dec[v] = global_p_dec;
        return true;
        }
        if ("Takayasu".equals(arg)){
        // No modifications necessary, done in the update routine
        modeltype = 1;
        global_p_dec = P_DEC;
        global_p_dec_inc_tsqr = P_DEC_INC_TSQR;
        for (int v=0;v<=maxspeed;v++)
          p_dec[v] = global_p_dec;
        return true;
        }
        if ("VDR".equals(arg)){
        // p_dec = p_dec(v) (default)
        modeltype = 2;
        for (int v=0;v<=MAXSPEED;v++){
          p_dec[v] = P_DEC;
        }
        return true;
        }
        if ("Fukui-Ishibashi".equals(arg)){
        // p_dec = p_dec(v) (default)
        modeltype = 3;
        global_p_dec = P_DEC;
        for (int v=0;v<=maxspeed;v++)
          p_dec[v] = 0.0;
        p_dec[maxspeed] = global_p_dec;
        return true;
        }
      }
      return false;
    }

    ////////////////////////////////////
    // get the actual simulation speed

    public int getSimSpeed(){
      int s =  scrollbar_simspeed.getValue();
      if (s != simspeed_old){
        simspeed_old = s;
        label_simspeed.setText((new Integer(s).toString())+" %");
      }
      return (int)(5*(100-s));
    }

    ////////////////////////////////////
    // get the actual maxspeed

    public int getMaxspeed(){
      int maxs =  scrollbar_maxspeed.getValue();
      if (maxs != maxspeed_old){
        maxspeed_old = maxs;
        switch(language){
        case 1:
          label_maxspeed.setText((new Integer(maxs).toString())+"
Zellen/Zeitschritt");
          break;
          // Default language is English
        default:
          label_maxspeed.setText((new Integer(maxs).toString())+" sites/timestep");
          break;
        }
      }
      return maxs;
    }
```

```
    ////////////////////////////////////////////
    // get the global density for the first time

    public double getFirstDensity(){
      int dens =  scrollbar_density.getValue();
      if (dens != density_old)
        density_old = dens;
      return dens*0.01;
    }

    ////////////////////////////////////
    // get the actual global density value

    public double getDensity(){
      int dens =  scrollbar_density.getValue();
      if (dens != density_old){
        density_old = dens;
        label_density.setText((new  Integer(dens).toString())+" %");
      }
      return dens*0.01;
    }

  }  // End of "public class CAmodel extends Applet implements Runnable"
```

9.3.2 Datei RoadCanvas.java

```
import java.awt.*;
import java.applet.*;
import java.util.*;

class RoadCanvas extends Canvas{

  // The simulated road
  private Road freeway;

  // Buffer for painting the road
  private Image buffer;

  // diagram labels
  //private String SpaceLabel = "space";
  //private String[] TimeLabel = {"t","i","m","e"};
  private String SpaceLabel;
  private String[] TimeLabel;

  private final String x1="Zeit";
  private final String y1="phi";
  private final String x2="rho";
  private final String y2="phi";
  private final String x3="phi";
  private final String y3="v";
  private final String x4="v";
  private final String y4="f";
  private final String x5="gap";
  private final String y5="f";

  private final String[] xAxisLabel={x1,x2,x3,x4,x5};
  private final String[] yAxisLabel={y1,y2,y3,y4,y5};

  // dot sizes
  private final int DOTSIZE = 1;
  private final int XDOTDIST = 1;

  // counter of the rows in the space-time-plot
  private int row;

  // some properties of the several graphic outputs
  private int xsize;
```

```
  private int ysize;
  private int xsizeSTD;
  private int ysizeSTD;
  private int xsizeIndy;
  private int ysizeIndy;
  private int xsizeDiagram;
  private int ysizeDiagram;
  private int xsizeDiagramPart;
  private int ysizeDiagramPart;

  private final int xShift = 17;
  private final int yShift = 17;
  private final int bluebar = 3;

  // some properties of the indianapolis scenario
  private int radIn;
  private int radOut;
  private int radVeh;
  private int xIndyMid;
  private int yIndyMid;

  // some properties of the diagrams
  private int[] xDiagram;
  private int[] yDiagram;

  private int[] xOrigin;
  private int[] yOrigin;

  private double[] xDelta;
  private double[] yDelta;

  // Init the class
  RoadCanvas(double density,int language){
    freeway = new Road(density);
    row = 0;
    SpaceLabel = new String();
    TimeLabel = new String[4];
    switch(language){
    case 1:
      SpaceLabel = "Ort";
      TimeLabel[0] = "Z";
      TimeLabel[1] = "e";
      TimeLabel[2] = "i";
      TimeLabel[3] = "t";
      break;
    default:
      SpaceLabel = "space";
      TimeLabel[0] = "t";
      TimeLabel[1] = "i";
      TimeLabel[2] = "m";
      TimeLabel[3] = "e";
      break;
    }
  }

  ////////////////////////////////////////
  // Update routine: simulation and painting

public void update(double p_dec[], double density, int counter,int modeltype,double
   p_dec_inc_sts,double p_dec_inc_tsqr,int maxspeed){

    // No buffer available
    if (buffer == null){
      xsize = size().width;
      ysize = size().height;
      xsizeSTD = xsize/2;
      ysizeSTD = ysize/2;
      xsizeIndy = xsize/2;
      ysizeIndy = ysize/2;
```

```
   xsizeDiagram = xsize/2;
   ysizeDiagram = ysize;
   xsizeDiagramPart = xsizeDiagram/8;
   ysizeDiagramPart = ysize/40;

   radIn = (int)(0.36*xsizeIndy); //** size of the street
   radOut = (int)(0.38*xsizeIndy);
   radVeh = (int)(0.37*xsizeIndy);

   xIndyMid = xsizeIndy/2+xShift+bluebar; //** midpoint of circle
   yIndyMid = ysizeSTD+ysizeIndy/2+yShift/2;

   xDiagram = new int[5];
   yDiagram = new int[5];
   xOrigin = new int[5];
   yOrigin = new int[5];
   xDelta = new double[5];
   yDelta = new double[5];

   for (int i=0;i<3;i++){
xDiagram[i] = xsizeDiagram-2*xsizeDiagramPart;
yDiagram[i] = (ysizeDiagram-4*ysizeDiagramPart)/4;
xOrigin[i] = xsizeSTD+xsizeDiagramPart;
yOrigin[i] = (i+1)*(yDiagram[i]+ysizeDiagramPart);
xDelta[i] = 1.0;
yDelta[i] = 1.0;
   }

   for (int i=3;i<5;i++){
xDiagram[i] = (int)(0.44*xDiagram[0]);
yDiagram[i] = (ysizeDiagram-4*ysizeDiagramPart)/4;
xOrigin[i] = (i == 3)? xsizeSTD+xsizeDiagramPart: xsizeSTD+xsizeDiagram/2;
yOrigin[i] = 4*(yDiagram[i]+ysizeDiagramPart)-8;
xDelta[i] = 1.0;
yDelta[i] = 1.0;
   }

   // draw all diagrams
   buffer = createImage(xsize,ysize);
   Graphics bg = buffer.getGraphics();
   for (int i=0;i<5;i++){
freeway.diagramAxis(bg,xOrigin[i],yOrigin[i],xOrigin[i]+xDiagram[i],yOrigin[i]);
bg.drawString(xAxisLabel[i],xOrigin[i]+xDiagram[i]+3,yOrigin[i]+5);

freeway.diagramAxis(bg,xOrigin[i],yOrigin[i],xOrigin[i],yOrigin[i]-yDiagram[i]);
bg.drawString(yAxisLabel[i],xOrigin[i]-20,yOrigin[i]-yDiagram[i]+25);
   }
   freeway.street(bg,xIndyMid,yIndyMid,radOut,radIn);
   freeway.bluelines(bg,xShift,yShift,bluebar-1,ysizeSTD);

freeway.SpaceTimeLabels(bg,xsizeSTD/3,3*yShift/4,xsizeSTD/2,yShift/2,50,SpaceLabel
,xShift/2,yShift+ysizeSTD/4,xShift/2,yShift+ysizeSTD/2,50,TimeLabel);
 }

 // Update routine: simulation
 freeway.update(p_dec,density,modeltype,p_dec_inc_sts,p_dec_inc_tsqr,maxspeed);
 // update all diagrams
 Graphics bg = buffer.getGraphics();
 // update space-time-plot
 freeway.paint(bg,xShift+row,XDOTDIST,DOTSIZE,xShift+bluebar);
 // update indy-scenario
 freeway.indypaint(bg,XDOTDIST*2,DOTSIZE*2,xIndyMid,yIndyMid,radVeh);

 // if enabled, plot diagrams after the measurement

 if (counter == 0){
   freeway.measure(p_dec[1]);
   for (int i=0;i<5;i++)
```

```
    freeway.diagram(bg,xOrigin[i],yOrigin[i],xDiagram[i],yDiagram[i],xDelta[i],yDelta[
    i],i);
     bg.copyArea(xOrigin[0]+3,yShift,xDiagram[0],yDiagram[0]-5,-DOTSIZE,0);
     }


     // move space-time-plot upwards
     if (row < ysizeSTD-DOTSIZE)
       row+=DOTSIZE;
     else
       bg.copyArea(xShift+bluebar,DOTSIZE+yShift,xsizeSTD-xShift+bluebar,ysizeSTD-
    DOTSIZE,0,-DOTSIZE);


     bg.dispose();
     repaint();
   }

   ////////////////////////////////
   // Cleaning the diagrams

   public void ClearDiagrams(){
     Graphics bg = buffer.getGraphics();
     freeway.ClearDiagrams(bg,xOrigin,yOrigin,xDiagram,yDiagram);
     bg.dispose();
     repaint();
   }

   ////////////////////////////////
   // Draing the picture

   public void paint(Graphics g){
     if (buffer != null)
       g.drawImage(buffer, 0, 0, null);
   }

   ////////////////////////////////
   // Update the picture

   public void update(Graphics g){
     paint(g);
   }

}  // End of "class RoadCanvas extends Canvas"
```

9.3.3 Datei Road.java

```
import java.awt.*;
import java.applet.*;
import java.util.*;

class Road{

  // length of the road
  public final int LENGTH = 300;
  // Maximum speed
  public final int MAXSPEED = 8;

  // point of measurement
  public final int mp =LENGTH-1;
  // length of the measurement
  public final int ml = LENGTH-200;
  // dummy for infinity
  public final double DBL_INF = 99999.99;

  // my private colors
  private final Color streetcolor = Color.white;
```

```
private final Color background = Color.white;
private final Color foreground = Color.black;
private final Color linecolor = Color.blue;

// colors for different velocities
private final Color free = Color.white;
private final Color v0color = Color.red;
private final Color v1color = new Color(255,51,0);
private final Color v2color = new Color(255,102,0);
private final Color v3color = new Color(255,153,0);
private final Color v4color = new Color(255,204,0);
private final Color v5color = Color.orange;
private final Color v6color = new Color(207,255,0);
private final Color v7color = new Color(153,255,0);
private final Color v8color = new Color(0,255,0);
private final Color
 vcolor[]={v0color,v1color,v2color,v3color,v4color,v5color,v6color,v7color,v8color}
 ;

// the road as an array of speeds
private int[] speed;
// number of vehicles on the road
private int cars;
// 1/number of vehicles
private double invcars;
// local density of vehicles on the road [veh/site]
private double dens;
// local mean velocity [sites/timestep]
private double v;
// local flow of vehicles [vehicles/time]
private double flow;
// deceleration probability
private double p_dec;
// global frequency of gaps
private double[] gapfreq;
// global frequency of velocities
private double[] vfreq;

////////////////////////////////
// Init the road for the simulation

public Road(double density){
  // the road as an array of speeds
  speed = new int[LENGTH];
  // frequency of the gaps
  gapfreq = new double[LENGTH];
  // frequency of the velocities
  vfreq = new double[MAXSPEED+1];

  // How many vehicles?
  cars = 0;
  int cars_abs = (int)(density*LENGTH);
  invcars = 1.0/((cars_abs > 0)? cars_abs: 1.0/DBL_INF);

  // clear road
  for (int i=0;i<LENGTH;i++)
    speed[i] = -1;

  // put vehicles on the road
  while (cars < cars_abs){
    int i = (int)(Math.random()*LENGTH);
    if (speed[i] == -1){
 speed[i] = 0;
 cars++;
    }
  }
}

////////////////////////////////
```

```
// update of the cellular automaton

public void update(double p_dec[], double prob_create,int modeltype,double
 p_dec_incr_sts,double p_dec_incr_tsqr,int maxspeed){

  // number of vehicles
  int car_local = (int)(prob_create*LENGTH);
  // the different between the actual and the requested number of vehicles
  int diff_cars = cars-car_local;

  for(int i=0;i<LENGTH;i++)
    gapfreq[i]=0.0;
  for(int i=0;i<=MAXSPEED;i++)
    vfreq[i]=0.0;

  // match the number of vehicles
  if (diff_cars > 0){
    // number of vehivles, that have to be killed
    int kill_cars = 0;
    while (kill_cars < diff_cars){
 int i = (int)(Math.random()*LENGTH);
 if (speed[i] != -1){
   speed[i] = -1;
   kill_cars++;
 }
    }
    cars = car_local;
    invcars  = 1.0/((cars > 0)? cars: 1.0/DBL_INF);
  }
  else
    if (diff_cars < 0){
 // number of vehivles, that have to be created
 int new_cars = 0;
 while (new_cars < Math.abs(diff_cars)){
   int i = (int)(Math.random()*LENGTH);
   if (speed[i] == -1){
     speed[i] = MAXSPEED;
     new_cars++;
   }
 }
 cars = car_local;
 invcars  = 1.0/((cars > 0)? cars: 1.0/DBL_INF);
    }

  // where is the 1st vehicle
  int i = -1;
  while((++i < LENGTH) && (speed[i] == -1));

  // go on until reaching the end of the lane
  while (i < LENGTH){
    // searching for the vehicle ahead
    int gap = i;
    while (speed[++gap%LENGTH] == -1);
    car_local++;

    // distance between two consecutive vehicles
    gap-=(i+1);
    gapfreq[gap]+=((invcars < DBL_INF)? invcars: 0);

    ////////////////////////////////////////
    // Update rules of the cellular automaton //
    ////////////////////////////////////////

    int speed_old = speed[i];

    // Acceleration
    if (gap > speed_old)
 speed[i] = Math.min(speed_old+1,maxspeed);
    // Slow down to prevent crashes
```

```
   else
speed[i] = gap;

   // Stochastic behavior
   // Slow down with prob p_dec
   // Different model types are applied! See Info...
   // 0=Standard
   // 1=T²
   //       All vehicles with gap<=1 are suffering from a
   //       increased deceleration probability
   // 2=VDR
   //       Comparable to modeltype 0, but with explicit
   //       p_dec as a function of speed p_dec = p_dec(v)
   // 3=Fukui-Ishibashi
   //       Full acceleration (unlike the Standard-CA with Delta_v=+1
   //       here v <- min(v_max,gap)
   //       p_dec only applied for v=v_max
   switch(modeltype){
   case 0:{
if ((speed[i] > 0) && (Math.random() <= p_dec[speed_old]))
  speed[i]--;
break;
   }
   case 1:{
if ((speed[i] > 0) && (Math.random() <= p_dec[speed_old]))
  speed[i]--;
break;
   }
   case 2:{
if (gap <= 1){
  if ((speed[i] > 0) && (Math.random() <=
Math.min(p_dec[speed[i]]+p_dec_incr_tsqr,1.0)))
    speed[i]--;
}
else
  if ((speed[i] > 0) && (Math.random() <= p_dec[speed[i]]))
    speed[i]--;
break;
   }
   case 3:{
if ((speed[i] == maxspeed) && (Math.random() <= p_dec[speed[i]]))
  speed[i]--;
break;
   }
   }

   vfreq[speed[i]]+=((invcars < DBL_INF)? invcars: 0);
   // next vehicle is on site j
   i+=(gap+1);
 }

 // Move the vehicles

 // Where is the 1st vehicle?
 car_local = 0;
 i = -1;
 while((++i < LENGTH) && (speed[i] == -1));

 // go on until reaching the end of the lane
 while (i < LENGTH){
   car_local++;
   int inext = i+speed[i];
   int j = inext%LENGTH;
   // exchange the positions
   if (i != j){
speed[j] = speed[i];
speed[i] = -1;
   }
   // searching for the vehicle ahead
```

```
    while (speed[++inext%LENGTH] == -1);
    i = inext;
  }
}

//////////////////
// space-time-plot

public void paint(Graphics g,int row,int dotdist,int dotsize,int xshift){
  int i;
  for (i = 0; i < LENGTH; i++){
    g.setColor(free);
    if (speed[i] >= 0) g.setColor(vcolor[speed[i]]);
    g.fillRect((xshift+i)*dotdist,row,dotsize,dotsize);
  }
  g.setColor(background);
}

////////////////////////////////
// plot the indianapolis-scenario

public void indypaint(Graphics g, int dotdist, int dotsize,int xm,int ym,int
 radius){

  // degrees per vehicle
  double f = 2.0*Math.PI/(double)LENGTH;
  for (int i = 0; i < LENGTH; i++){
    // angle of a vehicle
    double rad = f*i;
    // Erase a vehicle
    if (speed[i] == -1){
 g.setColor(streetcolor);
 g.fillRect((int)(xm+Math.cos(rad)*radius), (int)(ym+Math.sin(rad)*radius),
 dotsize, dotsize);
    }
    // plot a vehicle
    else{
 g.setColor(foreground);
 g.fillRect((int)(xm+Math.cos(rad)*radius), (int)(ym+Math.sin(rad)*radius),
 dotsize, dotsize);
    }
  }
}

////////////////////
// local measurement

public void measure(double pdec){
  int vsum=0;
  int rhoc=0;

  for(int i=mp;i>mp-ml;i--)
    if (speed[i] >= 0){
 vsum+=speed[i];
 rhoc++;
    }
  v = (double)(vsum)/(double)((rhoc > 0) ? rhoc : 1 );
  dens =(double)(rhoc)/(double)(ml);
  flow = v*dens;
  p_dec = pdec;
}

////////////////
// plot diagrams

void diagram(Graphics g, int x1, int y1, int intdx, int intdy, double dbldx, double
 dbldy, int index){
  double xValue;
  double yValue;
```

```
 g.setColor(foreground);
 // fundamental diagrams
 if (index < 3){
   switch(index){
   case 0:{
// local flow = local flow(local density)
xValue = 0.97; //original    dens
yValue = flow;
g.setColor(background);
g.fillRect(x1-25,y1-(int)(0.65/dbldy*intdy)-12,25,12);
g.setColor(foreground);
g.drawString(""+(int)(flow*60),x1-25,y1-(int)(0.65/dbldy*intdy));
break;
   }
   case 1:{
/*// local mean velocity = local mean velocity(local density)
xValue = dens;
yValue = v/MAXSPEED;*/
//
xValue = dens;
yValue = flow;
break;
   }
   default:{
// local mean velocity = local mean velocity(local flow)
xValue = flow;
yValue = v/MAXSPEED;
break;
   }
   }
   g.drawRect(x1+(int)(xValue/dbldx*intdx),y1-(int)(yValue/dbldy*intdy),1,1);
 }

 // bar charts
 else{
   // bar chart of the frequency of velocities
   if (index == 3){
// clear diagram
g.setColor(background);
g.fillRect(x1+1,y1-intdy,intdx-1,intdy);
drawArrow(g,x1,y1-intdy,10,8,270,true,foreground);
drawArrow(g,x1+intdx,y1,10,8,0,true,foreground);
// bar widht
int bar = intdx/(MAXSPEED+1);
g.setColor(foreground);
for (int i=0;i<=MAXSPEED;i++){
  g.setColor(vcolor[i]);
  int y = (int)(vfreq[i]/dbldy*intdy);
  g.fillRect(x1+i*bar+1,y1-y,bar-1,y);
}
   }
   // bar chart of the frequency of gaps
   else{
// clear diagram
g.setColor(background);
g.fillRect(x1+1,y1-intdy,intdx-1,intdy);
drawArrow(g,x1,y1-intdy,10,8,270,true,foreground);
drawArrow(g,x1+intdx,y1,10,8,0,true,foreground);
// number of bars
int xmax = 18;
// bar width
int bar = intdx/xmax;
g.setColor(foreground);
for (int i=0;i<xmax;i++){
  int y = (int)(gapfreq[i]/dbldy*intdy);
  g.fillRect(x1+i*bar+1,y1-y,bar-1,y);
}
   }
 }
```

```
}

///////////////
// diagram axes

public void diagramAxis(Graphics g, int xOr, int yOr, int xDiag, int yDiag){
  g.drawLine(xOr,yOr,xDiag,yDiag);
  drawArrow(g,xDiag,yDiag,10,8,(xOr == xDiag)? 270: 0,true,foreground);
}

/////////////////////////////////////////
// draw the road of the indianapolis-scenario

public void street(Graphics g, int xm, int ym, int r1, int r2){
  g.setColor(streetcolor);
  g.fillOval(xm-r1,ym-r1,2*r1,2*r1);
  g.setColor(background);
  g.fillOval(xm-r2,ym-r2,2*r2,2*r2);
  g.setColor(linecolor);
  g.drawLine(xm+r2-20,ym,xm+r1+20,ym);
  g.setColor(foreground);
  g.drawArc(xm-r2+30,ym-r2+30,2*r2-60,2*r2-60,0,-270);
  drawArrow(g, xm,ym-r2+30,10,8,355, true, Color.black);
}

////////////////
// clear diagrams

public void ClearDiagrams(Graphics bg,int xOrigin[],int yOrigin[],int
 xDiagram[],int yDiagram[]){
  bg.setColor(background);
  for (int i=0;i<5;i++){
    bg.fillRect(xOrigin[i]+1,yOrigin[i]-yDiagram[i],xDiagram[i]-1,yDiagram[i]);
    drawArrow(bg,xOrigin[i],yOrigin[i]-yDiagram[i],10,8,270,true,foreground);
    drawArrow(bg,xOrigin[i]+xDiagram[i],yOrigin[i],10,8,0,true,foreground);
  }
  bg.setColor(foreground);
}

//////////////////////////////
// blue line at the first site

public void bluelines(Graphics g,int x1,int y1,int x2,int y2){
  Color old_color = g.getColor();
  g.setColor(linecolor);
  g.fillRect(x1,y1,x2,y2);
  g.setColor(old_color);
}

/////////////////////////////////////////////////////
// Labels of the space-time-plot
// The letters of the ordinate are arranged vertically.

public void SpaceTimeLabels(Graphics g,int x1,int y1,int xA1,int yA1,int
 length1,String label1,int x2,int y2,int xA2,int yA2,int length2,String[] label2){
  Color color = foreground;
  Color old_color = g.getColor();
  g.setColor(color);
  g.drawString(label1,x1,y1);
  g.drawLine(xA1,yA1,xA1+length1,yA1);
  drawArrow(g,xA1+length1,yA1,10,8,0,true,color);
  for (int i=0;i<4;i++)
    g.drawString(label2[i],x2-5,y2+i*15);
  g.drawLine(xA2,yA2,xA2,yA2+length2);
  drawArrow(g,xA2,yA2+length2,10,8,90,true,color);
  g.setColor(old_color);
}

/////////////////////////////////////////////////////
```

```
  // plot an arrow with any properties like size, angle, ...

  public void drawArrow(Graphics g,int xA,int yA,int Length,int Width,int deg,boolean
    filled,Color color){

    double Angle=(double)deg/180.0*Math.PI;
    double sinA = Math.sin(Angle);
    double cosA = Math.cos(Angle);
    int x1 = xA-(int)(cosA*Length + sinA*Width/2);
    int y1 = yA-(int)(sinA*Length - cosA*Width/2);

    Polygon filledPolygon = new Polygon();
    filledPolygon.addPoint(xA, yA);
    filledPolygon.addPoint(x1,y1);
    filledPolygon.addPoint(x1+(int)(sinA*Width),y1-(int)(cosA*Width));
    Color old_color = g.getColor();
    g.setColor(color);
    if(filled)
      g.fillPolygon(filledPolygon);
    else
      g.drawPolygon(filledPolygon);
    g.setColor(old_color);
  }
}  // End of "class Road"
```

9.3.4 Datei MessageBox.java

```
import java.awt.*;
import java.applet.*;
import java.util.*;

////////////////////////////////////////////
// The message boxes for changing the p_dec's

class MessageBox extends Frame{

  // used language
  public int local_language;

  // Message box alive?
  public boolean isRunning;

  // Close button
  private Button close;

  // The scrollbars
  private Scrollbar p_dec_s;
  private Scrollbar p_dec_s_inc;
  private Scrollbar p_dec_s_0;
  private Scrollbar p_dec_s_1;
  private Scrollbar p_dec_s_2;
  private Scrollbar p_dec_s_3;
  private Scrollbar p_dec_s_4;
  private Scrollbar p_dec_s_5;
  private Scrollbar p_dec_s_6;
  private Scrollbar p_dec_s_7;
  private Scrollbar p_dec_s_8;
  private Scrollbar p_dec_s_9;

  // The labels
  private Label p_dec_s_name;
  private Label p_dec_s_inc_name;
  private Label p_dec_s_0_name;
  private Label p_dec_s_1_name;
  private Label p_dec_s_2_name;
  private Label p_dec_s_3_name;
  private Label p_dec_s_4_name;
```

```
private Label p_dec_s_5_name;
private Label p_dec_s_6_name;
private Label p_dec_s_7_name;
private Label p_dec_s_8_name;
private Label p_dec_s_9_name;

// ... and more labels
private Label p_dec_l;
private Label p_dec_l_inc;
private Label p_dec_l_0;
private Label p_dec_l_1;
private Label p_dec_l_2;
private Label p_dec_l_3;
private Label p_dec_l_4;
private Label p_dec_l_5;
private Label p_dec_l_6;
private Label p_dec_l_7;
private Label p_dec_l_8;
private Label p_dec_l_9;

// For T² the resulting deceleration prob.
private Label p_dec_result_name;
private Label p_dec_result_value;

// A textfield
private TextArea text;

// Some nice panels
private Panel panel_text;
private Panel panel_scroll;
private Panel panel_button;

// Local variables
// speed, deceleration prob's ...
private int local_maxspeed;
private double[] local_p_dec_array;
private double local_p_dec;
private int local_p_dec_ini;
private double local_p_dec_inc;
private int local_p_dec_inc_ini;

// Which model type we use?
private int modeltype;

/////////////////
// Standard-model

public MessageBox(double p_dec,int language){

  local_language = language;
  isRunning = false;
  modeltype = 0;
  local_p_dec = p_dec;
  local_p_dec_ini = (int)(local_p_dec*100);

  switch(local_language){
  case 1:
    close = new Button("Schließen");
    text = new TextArea("Standard-CA-Modell",20,50);
    p_dec_s_name = new Label("Wahrsch. P");
    break;
    // Default language is English
  default:
    close = new Button("Close");
    text = new TextArea("Standard-CA-model",20,50);
    p_dec_s_name = new Label("Prob. P");
    break;
  }
  panel_text = new Panel();
```

```
  panel_scroll = new Panel();
  panel_button = new Panel();

  panel_text.setLayout(new GridLayout(1,1));
  panel_scroll.setLayout(new GridLayout(2,3));
  panel_button.setLayout(new GridLayout(3,7));

  panel_text.add(text);

  p_dec_s_name.setAlignment(Label.RIGHT);
  p_dec_s = new Scrollbar(Scrollbar.HORIZONTAL,local_p_dec_ini,1,0,100);
  p_dec_l = new Label((new Integer(local_p_dec_ini).toString())+" %");
  for (int i=0;i<3;i++)
    panel_scroll.add(new Label("    "));
  panel_scroll.add(p_dec_s_name);
  panel_scroll.add(p_dec_s);
  panel_scroll.add(p_dec_l);

  for (int i=0;i<12;i++)
    panel_button.add(new Label("   "));
  panel_button.add(close);
  for (int i=0;i<8;i++)
    panel_button.add(new Label("   "));

  add("North",panel_text);
  add("Center",panel_scroll);
  add("South",panel_button);
}

////////////////
// Takayasu²-model
public MessageBox(double p_dec,double p_dec_inc,int language){

  local_language = language;
  isRunning = false;
  modeltype = 1;

  local_p_dec = p_dec;
  local_p_dec_ini = (int)(local_p_dec*100);
  local_p_dec_inc = p_dec_inc;
  local_p_dec_inc_ini = (int)(local_p_dec_inc*100);

  switch (local_language){
  case 1:
    close = new Button("Schließen");
    text = new TextArea("Takayasu²-Modell",20,50);
    break;
    // Default language is English
  default:
    close = new Button("Close");
    text = new TextArea("Takayasu²-model",20,50);
    break;
  }
  panel_text = new Panel();
  panel_scroll = new Panel();
  panel_button = new Panel();

  panel_text.setLayout(new GridLayout(1,1));
  panel_scroll.setLayout(new GridLayout(4,3));
  panel_button.setLayout(new GridLayout(3,7));

  panel_text.add(text);

  p_dec_s_name = new Label("    P(g>0)");
  p_dec_s_name.setAlignment(Label.RIGHT);
  p_dec_s = new Scrollbar(Scrollbar.HORIZONTAL,local_p_dec_ini,1,0,100);
  p_dec_l = new Label((new Integer(local_p_dec_ini).toString())+" %");
  p_dec_s_inc_name = new Label("   +P_add(g<=1)");
  p_dec_s_inc_name.setAlignment(Label.RIGHT);
```

```
  p_dec_s_inc = new Scrollbar(Scrollbar.HORIZONTAL,local_p_dec_inc_ini,1,0,100);
  p_dec_l_inc = new Label((new Integer(local_p_dec_inc_ini).toString())+" %");
  p_dec_result_name = new Label(" -> P(g<=1)");
  p_dec_result_name.setAlignment(Label.RIGHT);
  p_dec_result_value = new Label((new
 Integer(Math.min(local_p_dec_ini+local_p_dec_inc_ini,100)).toString())+" %");
  p_dec_result_value.setAlignment(Label.CENTER);

  for (int i=0;i<3;i++)
    panel_scroll.add(new Label("    "));
  panel_scroll.add(p_dec_s_name);
  panel_scroll.add(p_dec_s);
  panel_scroll.add(p_dec_l);
  panel_scroll.add(p_dec_s_inc_name);
  panel_scroll.add(p_dec_s_inc);
  panel_scroll.add(p_dec_l_inc);
  panel_scroll.add(p_dec_result_name);
  panel_scroll.add(p_dec_result_value);
  panel_scroll.add(new Label("    "));

  for (int i=0;i<12;i++)
    panel_button.add(new Label("   "));
  panel_button.add(close);
  for (int i=0;i<8;i++)
    panel_button.add(new Label("   "));

  add("North",panel_text);
  add("Center",panel_scroll);
  add("South",panel_button);
}

////////////////
// VDR-model

public MessageBox(double p_dec[],int maxspeed,int maxspeed_array,int language){

  local_p_dec_array = new double[maxspeed_array];
  local_language = language;
  isRunning = false;
  modeltype = 2;

  String s = new String();

  local_maxspeed = maxspeed;
  for (int v=0;v<maxspeed_array;v++)
    local_p_dec_array[v] = p_dec[v];

  switch (local_language){
  case 1:
    close = new Button("Schließen");
    text = new TextArea("VDR-Modell",20,50);
    break;
    // Default language is English
  default:
    close = new Button("Close");
    text = new TextArea("VDR-model",20,50);
    break;
  }

  panel_text = new Panel();
  panel_scroll = new Panel();
  panel_button = new Panel();

  panel_text.setLayout(new GridLayout(1,1));
  panel_scroll.setLayout(new GridLayout(maxspeed_array,3));
  panel_button.setLayout(new GridLayout(3,7));

  panel_text.add(text);
```

```
 for (int v=0;v<maxspeed_array;v++){
   local_p_dec_ini = (int)(100*p_dec[v]);
   s = "       P(v="+v+")";
   switch(v){
   case 0:{
p_dec_s_0_name = new Label(s);
p_dec_s_0 = new Scrollbar(Scrollbar.HORIZONTAL,local_p_dec_ini,1,0,100);
p_dec_l_0 = new Label((new Integer(local_p_dec_ini).toString())+" %");
panel_scroll.add(p_dec_s_0_name);
p_dec_s_0_name.setAlignment(Label.RIGHT);
panel_scroll.add(p_dec_s_0);
panel_scroll.add(p_dec_l_0);
break;
   }
   case 1:{
p_dec_s_1_name = new Label(s);
p_dec_s_1 = new Scrollbar(Scrollbar.HORIZONTAL,local_p_dec_ini,1,0,100);
p_dec_l_1 = new Label((new Integer(local_p_dec_ini).toString())+" %");
panel_scroll.add(p_dec_s_1_name);
p_dec_s_1_name.setAlignment(Label.RIGHT);
panel_scroll.add(p_dec_s_1);
panel_scroll.add(p_dec_l_1);
break;
   }
   case 2:{
p_dec_s_2_name = new Label(s);
p_dec_s_2 = new Scrollbar(Scrollbar.HORIZONTAL,local_p_dec_ini,1,0,100);
p_dec_l_2 = new Label((new Integer(local_p_dec_ini).toString())+" %");
panel_scroll.add(p_dec_s_2_name);
p_dec_s_2_name.setAlignment(Label.RIGHT);
panel_scroll.add(p_dec_s_2);
panel_scroll.add(p_dec_l_2);
break;
   }
   case 3:{
p_dec_s_3_name = new Label(s);
p_dec_s_3 = new Scrollbar(Scrollbar.HORIZONTAL,local_p_dec_ini,1,0,100);
p_dec_l_3 = new Label((new Integer(local_p_dec_ini).toString())+" %");
panel_scroll.add(p_dec_s_3_name);
p_dec_s_3_name.setAlignment(Label.RIGHT);
panel_scroll.add(p_dec_s_3);
panel_scroll.add(p_dec_l_3);
break;
   }
   case 4:{
p_dec_s_4_name = new Label(s);
p_dec_s_4 = new Scrollbar(Scrollbar.HORIZONTAL,local_p_dec_ini,1,0,100);
p_dec_l_4 = new Label((new Integer(local_p_dec_ini).toString())+" %");
panel_scroll.add(p_dec_s_4_name);
p_dec_s_4_name.setAlignment(Label.RIGHT);
panel_scroll.add(p_dec_s_4);
panel_scroll.add(p_dec_l_4);
break;
   }
   case 5:{
p_dec_s_5_name = new Label(s);
p_dec_s_5 = new Scrollbar(Scrollbar.HORIZONTAL,local_p_dec_ini,1,0,100);
p_dec_l_5 = new Label((new Integer(local_p_dec_ini).toString())+" %");
panel_scroll.add(p_dec_s_5_name);
p_dec_s_5_name.setAlignment(Label.RIGHT);
panel_scroll.add(p_dec_s_5);
panel_scroll.add(p_dec_l_5);
break;
   }
   case 6:{
p_dec_s_6_name = new Label(s);
p_dec_s_6 = new Scrollbar(Scrollbar.HORIZONTAL,local_p_dec_ini,1,0,100);
p_dec_l_6 = new Label((new Integer(local_p_dec_ini).toString())+" %");
panel_scroll.add(p_dec_s_6_name);
```

```
  p_dec_s_6_name.setAlignment(Label.RIGHT);
  panel_scroll.add(p_dec_s_6);
  panel_scroll.add(p_dec_l_6);
  break;
     }
     case 7:{
  p_dec_s_7_name = new Label(s);
  p_dec_s_7 = new Scrollbar(Scrollbar.HORIZONTAL,local_p_dec_ini,1,0,100);
  p_dec_l_7 = new Label((new Integer(local_p_dec_ini).toString())+" %");
  panel_scroll.add(p_dec_s_7_name);
  p_dec_s_7_name.setAlignment(Label.RIGHT);
  panel_scroll.add(p_dec_s_7);
  panel_scroll.add(p_dec_l_7);
  break;
     }
     case 8:{
  p_dec_s_8_name = new Label(s);
  p_dec_s_8 = new Scrollbar(Scrollbar.HORIZONTAL,local_p_dec_ini,1,0,100);
  p_dec_l_8 = new Label((new Integer(local_p_dec_ini).toString())+" %");
  panel_scroll.add(p_dec_s_8_name);
  p_dec_s_8_name.setAlignment(Label.RIGHT);
  panel_scroll.add(p_dec_s_8);
  panel_scroll.add(p_dec_l_8);
  break;
     }
     case 9:{
  p_dec_s_9_name = new Label(s);
  p_dec_s_9 = new Scrollbar(Scrollbar.HORIZONTAL,local_p_dec_ini,1,0,100);
  p_dec_l_9 = new Label((new Integer(local_p_dec_ini).toString())+" %");
  panel_scroll.add(p_dec_s_9_name);
  p_dec_s_9_name.setAlignment(Label.RIGHT);
  panel_scroll.add(p_dec_s_9);
  panel_scroll.add(p_dec_l_9);
  break;
     }
     }
   }

   for (int i=0;i<12;i++)
     panel_button.add(new Label("   "));
   panel_button.add(close);
   for (int i=0;i<8;i++)
     panel_button.add(new Label("   "));

   add("North",panel_text);
   add("Center",panel_scroll);
   add("South",panel_button);
}

////////////////
// Fukui-Ishibashi

public MessageBox(double p_dec,int maxspeed,int maxspeed_array,int language){

   local_language = language;
   isRunning = false;
   modeltype = 3;
   local_p_dec = p_dec;
   local_p_dec_ini = (int)(local_p_dec*100);
   local_maxspeed = maxspeed;

   switch (local_language){
   case 1:
     close = new Button("Schließen");
     text = new TextArea("Fukui-Ishibashi-Modell",20,50);
     break;
     // Default language is English
   default:
     close = new Button("Close");
```

```
    text = new TextArea("Fukui-Ishibashi-model",20,50);
    break;
  }

  panel_text = new Panel();
  panel_scroll = new Panel();
  panel_button = new Panel();

  panel_text.setLayout(new GridLayout(1,1));
  panel_scroll.setLayout(new GridLayout(2,3));
  panel_button.setLayout(new GridLayout(3,7));

  panel_text.add(text);

  p_dec_s_name = new Label("P(v_max)");
  p_dec_s_name.setAlignment(Label.RIGHT);
  p_dec_s = new Scrollbar(Scrollbar.HORIZONTAL,local_p_dec_ini,1,0,100);
  p_dec_l = new Label((new Integer(local_p_dec_ini).toString())+" %");
  for (int i=0;i<3;i++)
    panel_scroll.add(new Label("    "));
  panel_scroll.add(p_dec_s_name);
  panel_scroll.add(p_dec_s);
  panel_scroll.add(p_dec_l);

  for (int i=0;i<12;i++)
    panel_button.add(new Label("   "));
  panel_button.add(close);
  for (int i=0;i<8;i++)
    panel_button.add(new Label("   "));

  add("North",panel_text);
  add("Center",panel_scroll);
  add("South",panel_button);
}

// Close-Button

public boolean action(Event evt, Object arg){
  switch(local_language){
  case 1:
    if(arg.equals("Schließen")){
      isRunning = false;
      hide();
      return true;
    }
    break;
  default:
    if(arg.equals("Close")){
      isRunning = false;
      hide();
      return true;
    }
    break;
  }
  return false;
}

// Handle Window Events

public boolean handleEvent(Event e){

  switch(e.id){
  case Event.SCROLL_LINE_UP:
  case Event.SCROLL_LINE_DOWN:
  case Event.SCROLL_PAGE_UP:
  case Event.SCROLL_PAGE_DOWN:
  case Event.SCROLL_ABSOLUTE:{
    switch(modeltype){
    case 0:{
```

```
  local_p_dec_ini = p_dec_s.getValue();
  local_p_dec = (double)local_p_dec_ini/100;
  p_dec_l.setText((new Integer(local_p_dec_ini).toString())+" %");
  break;
     }
     case 1:{
  local_p_dec_ini = p_dec_s.getValue();
  local_p_dec = (double)local_p_dec_ini/100;
  p_dec_l.setText((new Integer(local_p_dec_ini).toString())+" %");
  local_p_dec_inc_ini = p_dec_s_inc.getValue();
  local_p_dec_inc = (double)local_p_dec_inc_ini/100;
  p_dec_l_inc.setText((new Integer(local_p_dec_inc_ini).toString())+" %");
  p_dec_result_value.setText(new
  Integer(Math.min(local_p_dec_ini+local_p_dec_inc_ini,100)).toString()+" %");
  break;
     }
     case 2:{
  p_dec_l_0.setText((new Integer(p_dec_s_0.getValue()).toString())+" %");
  local_p_dec_array[0] = (double)p_dec_s_0.getValue()/100;
  p_dec_l_1.setText((new Integer(p_dec_s_1.getValue()).toString())+" %");
  local_p_dec_array[1] = (double)p_dec_s_1.getValue()/100;
  p_dec_l_2.setText((new Integer(p_dec_s_2.getValue()).toString())+" %");
  local_p_dec_array[2] = (double)p_dec_s_2.getValue()/100;
  p_dec_l_3.setText((new Integer(p_dec_s_3.getValue()).toString())+" %");
  local_p_dec_array[3] = (double)p_dec_s_3.getValue()/100;
  p_dec_l_4.setText((new Integer(p_dec_s_4.getValue()).toString())+" %");
  local_p_dec_array[4] = (double)p_dec_s_4.getValue()/100;
  p_dec_l_5.setText((new Integer(p_dec_s_5.getValue()).toString())+" %");
  local_p_dec_array[5] = (double)p_dec_s_5.getValue()/100;
  p_dec_l_6.setText((new Integer(p_dec_s_6.getValue()).toString())+" %");
  local_p_dec_array[6] = (double)p_dec_s_6.getValue()/100;
  p_dec_l_7.setText((new Integer(p_dec_s_7.getValue()).toString())+" %");
  local_p_dec_array[7] = (double)p_dec_s_7.getValue()/100;
  p_dec_l_8.setText((new Integer(p_dec_s_8.getValue()).toString())+" %");
  local_p_dec_array[8] = (double)p_dec_s_8.getValue()/100;
  //p_dec_l_9.setText((new Integer(p_dec_s_9.getValue()).toString())+" %");
  //local_p_dec_array[9] = (double)p_dec_s_9.getValue()/100;
  break;
     }
     case 3:{
  local_p_dec_ini = p_dec_s.getValue();
  local_p_dec = (double)local_p_dec_ini/100;
  p_dec_l.setText((new Integer(local_p_dec_ini).toString())+" %");
  break;
     }
     }
  }
  }

  if ((e.id == Event.WINDOW_DESTROY) || (e.id == Event.KEY_PRESS && e.key==27)){
     isRunning = false;
     if (modeltype == 3){
  p_dec_s_0.enable(true);
  p_dec_s_1.enable(true);
  p_dec_s_2.enable(true);
  p_dec_s_3.enable(true);
  p_dec_s_4.enable(true);
  p_dec_s_5.enable(true);
  p_dec_s_6.enable(true);
  p_dec_s_7.enable(true);
  p_dec_s_8.enable(true);
     }
     hide();
     return true;
  }

  return super.handleEvent(e);
}
```

```
/////////////////
// Standard-model
public void show(double p_dec,int language){

  modeltype = 0;
  isRunning = true;

  switch(local_language){
  case 1:
    text.setText("          Standard-CA-Modell\n          Regeln fuer paralleles
 Update\n          ============================\n\n- Fahrzeugposition p\n- Geschw.
 v und max. Geschw. v_max\n- Luecke g = Zahl der leeren Zellen zum Vordermann\n\n1.
 Beschleunigung\n       v <- MIN(v+1,v_max,g)\n2. Troedeln\n       mit Wahrsch. P:
 v <- MAX(v-1,0)\n3. Bewegung\n       p <- p+v");
    super.setTitle("Infos & Details");
    break;
  default:
    text.setText("          Standard-CA-model\n          Rules for the parallel
 update\n          ======================\n\n- Vehicle position p\n- Velocity v and
 Maximum Velocity v_max\n- Gap g = Amount of empty cells ahead\n\n1. Acceleration
 with respect to g\n     v <- MIN(v+1,v_max,g)\n2. Randomization\n     with prob. P
 do v <- MAX(v-1,0)\n3. Movement\n     p <- p+v");
    super.setTitle("Info & Details");
    break;
  }

  local_p_dec = p_dec;
  local_p_dec_ini = (int)(100*p_dec);

  p_dec_s.setValue(local_p_dec_ini);
  p_dec_l.setText((new Integer(local_p_dec_ini).toString())+" %");

  super.pack();
  super.show();
  close.requestFocus();
}

/////////////////
// Takayasu²-model
public void show(double p_dec,double p_dec_inc,int language){

  modeltype = 1;
  isRunning = true;

  switch(local_language){
  case 1:
    text.setText("          Takayasu²-Modell\n          Regeln fuer paralleles
 Update\n          ============================\n\n- Fahrzeugposition p\n- Geschw.
 v und max. Geschw. v_max\n- Luecke g = Zahl der leeren Zellen zum Vordermann\n\n1.
 Beschleunigung\n       v <- MIN(v+1,v_max,g)\n2. Troedeln\n       mit Wahrsch.
 P(g): v <- MAX(v-1,0)\n       Ueblicherweise P(g<=1) > P(g>1)\n3. Bewegung\n
 p <- p+v");
    super.setTitle("Infos & Details");
    break;
  default:
    text.setText("          Takayasu²-model\n          Rules for the parallel
 update\n          ======================\n\n- Vehicle position p\n- Velocity v and
 Maximum Velocity v_max\n- Gap g = Amount of empty cells ahead\n\n1. Acceleration
 with respect to g\n     v <- MIN(v+1,v_max,g)\n2. Randomization\n     with prob.
 P(g) do v <- MAX(v-1,0)\n       Usually P(g<=1) > P(g>1)\n3. Movement\n     p <-
 p+v");
    super.setTitle("Info & Details");
    break;
  }

  local_p_dec = p_dec;
  local_p_dec_ini = (int)(100*p_dec);
  local_p_dec_inc = p_dec_inc;
  local_p_dec_inc_ini = (int)(100*local_p_dec_inc);
```

```
  p_dec_s.setValue(local_p_dec_ini);
  p_dec_l.setText((new Integer(local_p_dec_ini).toString())+" %");
  p_dec_s_inc.setValue(local_p_dec_inc_ini);
  p_dec_l_inc.setText((new Integer(local_p_dec_inc_ini).toString())+" %");
  p_dec_result_value.setText((new
 Integer(Math.min(local_p_dec_ini+local_p_dec_inc_ini,100)).toString())+" %");

  super.pack();
  super.show();
  close.requestFocus();
}

public double getlocal_p_dec()
  {
    return local_p_dec;
  }

public double getlocal_p_dec_inc()
  {
    return local_p_dec_inc;
  }

public double[] getlocal_p_dec_array()
  {
    return local_p_dec_array;
  }

/////////////////
// VDR-model
public void show(double p_dec[],int maxspeed,int maxspeed_array,int language){

  modeltype = 2;
  isRunning = true;

  switch(local_language){
  case 1:
    text.setText("          VDR-Modell\n          Regeln fuer paralleles Update\n
 ============================\n\n- Fahrzeugposition p\n- Geschw. v und max.
 Geschw. v_max\n- Luecke g = Zahl der leeren Zellen zum Vordermann\n\n1.
 Beschleunigung\n       v <- MIN(v+1,v_max,g)\n2. Troedeln\n       mit Wahrsch.
 P(v): v <- MAX(v-1,0)\n       (VDR=geschw.-abhaeng. Regeln)\n3. Bewegung\n       p
 <- p+v");      super.setTitle("Infos & Details");
    break;
  default:
    text.setText("          VDR-model\n          Rules for the parallel update\n
 ======================\n\n- Vehicle position p\n- Velocity v and Maximum Velocity
 v_max\n- Gap g = Amount of empty cells ahead\n\n1. Acceleration with respect to
 g\n     v <- MIN(v+1,v_max,g)\n2. Randomization\n     with prob. P(v) do v <-
 MAX(v-1,0)\n       (VDR=velocity-depending rules)\n3. Movement\n     p <- p+v");
    super.setTitle("Info & Details");
    break;
  }

  local_maxspeed = maxspeed;
  for (int v=0;v<maxspeed_array;v++){
    local_p_dec_array[v] = p_dec[v];
    local_p_dec_ini = (int)(100*local_p_dec_array[v]);
    switch(v){
    case 0:{
 p_dec_s_0.setValue(local_p_dec_ini);
 p_dec_l_0.setText((new Integer(local_p_dec_ini).toString())+" %");
 if (0 > maxspeed)
   p_dec_s_0.enable(false);
 break;
    }
    case 1:{
 p_dec_s_1.setValue(local_p_dec_ini);
 p_dec_l_1.setText((new Integer(local_p_dec_ini).toString())+" %");
```

```
 if (1 > maxspeed)
   p_dec_s_1.enable(false);
 break;
    }
    case 2:{
 p_dec_s_2.setValue(local_p_dec_ini);
 p_dec_l_2.setText((new Integer(local_p_dec_ini).toString())+" %");
 if (2 > maxspeed)
   p_dec_s_2.enable(false);
 break;
    }
    case 3:{
 p_dec_s_3.setValue(local_p_dec_ini);
 p_dec_l_3.setText((new Integer(local_p_dec_ini).toString())+" %");
 if (3 > maxspeed)
   p_dec_s_3.enable(false);
 break;
    }
    case 4:{
 p_dec_s_4.setValue(local_p_dec_ini);
 p_dec_l_4.setText((new Integer(local_p_dec_ini).toString())+" %");
 if (4 > maxspeed)
   p_dec_s_4.enable(false);
 break;
    }
    case 5:{
 p_dec_s_5.setValue(local_p_dec_ini);
 p_dec_l_5.setText((new Integer(local_p_dec_ini).toString())+" %");
 if (5 > maxspeed)
   p_dec_s_5.enable(false);
 break;
    }
    case 6:{
 p_dec_s_6.setValue(local_p_dec_ini);
 p_dec_l_6.setText((new Integer(local_p_dec_ini).toString())+" %");
 if (6 > maxspeed)
   p_dec_s_6.enable(false);
 break;
    }
    case 7:{
 p_dec_s_7.setValue(local_p_dec_ini);
 p_dec_l_7.setText((new Integer(local_p_dec_ini).toString())+" %");
 if (7 > maxspeed)
   p_dec_s_7.enable(false);
 break;
    }
    case 8:{
 p_dec_s_8.setValue(local_p_dec_ini);
 p_dec_l_8.setText((new Integer(local_p_dec_ini).toString())+" %");
 if (8 > maxspeed)
   p_dec_s_8.enable(false);
 break;
    }
    case 9:{
 p_dec_s_9.setValue(local_p_dec_ini);
 p_dec_l_9.setText((new Integer(local_p_dec_ini).toString())+" %");
 if (9 > maxspeed)
   p_dec_s_9.enable(false);
 break;
    }
    }
  }

  super.pack();
  super.show();
  close.requestFocus();
}

////////////////
```

```
  // Fukui-Ishibashi
  public void show(double p_dec,int maxspeed,int maxspeed_array,int language){

    modeltype = 3;
    isRunning = true;

    switch(local_language){
    case 1:
      text.setText("          Fukui-Ishibashi-Modell\n          Regeln fuer
    paralleles Update\n          ============================\n\n- Fahrzeugposition
    p\n- Geschw. v und max. Geschw. v_max\n- Luecke g = Zahl der leeren Zellen zum
    Vordermann\n\n1. Beschleunigung\n       v <- MAX(v_max,g)\n2. Troedeln\n
    Wenn v = v_max dann\n       mit Wahrsch. P(v_max): v <- max(v-1,0)\n3. Bewegung\n
    p <- p+v");      super.setTitle("Infos & Details");
      break;
    default:
      text.setText("          Fukui-Ishibashi-model\n          Rules for the parallel
    update\n          ======================\n\n- Vehicle position p\n- Velocity v and
    Maximum Velocity v_max\n- Gap g = Amount of empty cells ahead\n\n1. Acceleration
    with respect to g\n     v <- MIN(v_max,g)\n2. Randomization\n     if v = v_max
    then\n       with prob. P(v_max) do v <- MAX(v-1,0)\n3. Movement\n     p <- p+v");
      super.setTitle("Info & Details");
      break;
    }
    local_p_dec = p_dec;
    local_p_dec_ini = (int)(100*p_dec);
    local_maxspeed = maxspeed;

    p_dec_s.setValue(local_p_dec_ini);
    p_dec_l.setText((new Integer(local_p_dec_ini).toString())+" %");

    super.pack();
    super.show();
    close.requestFocus();
  }
}
```

Bestätigung über die selbstständige Anfertigung

Ich erkläre hiermit, dass ich die vorliegende Facharbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angegebenen Quellen und Hilfsmittel verwendet habe.

_____	_____
Ort, Datum	Unterschrift des Schülers