Gymnasium Puchheim
Bürgermeister-Ertl-Str. 11
82178 Puchheim

Schuljahr 2017/19

Seminararbeit

im W-Seminar „Chaostheorie und Fraktale“

Phantomstau – Der Stau aus dem Nichts

von
Tobias Felix Zillmann

Betreuende Lehrkraft: Herr Roschy

Abgabetermin: 06.11.2018

Punktezahl für die abgelieferte Arbeit: …….

Punktezahl für Präsentation mit Prüfungsgespräch: …….

Gesamtleistung: …….

_____
(Unterschrift des Kursleiters)

Inhaltsverzeichnis

1 Einleitung

Seit dem Jahr 1960 hat sich der Bestand der Personenkraftwagen von einer Anzahl von rund 4,5 Millionen auf fast 47 Millionen mehr als verzehnfacht. Zusätzlich dazu kommen mehrere Millionen Kraftfahrräder, Lastkraftwagen und sonstige Kraftfahrzeuge.[1] Hingegen dessen veränderte sich jedoch die Gesamtlänge des deutschen Straßennetzes innerhalb der letzten Jahre nur sehr geringfügig.[2]

Durch die stetig steigende Anzahl an Fahrzeugen geraten die Straßen der Bundesrepublik immer mehr an die Grenzen ihrer Kapazität. Somit ist es, und wird es auch immer wahrscheinlicher für den individuellen Menschen auf der Reise in den Urlaub oder auch nur zur Arbeit in einen Verkehrsstau zu geraten. Die häufigste Ursache für einen solchen stockenden Verkehr ist kein Hindernis wie beispielsweise eine Baustelle, sondern ein zu hohes Verkehrsaufkommen. Hierbei spricht man auch von einem „Phantomstau“ beziehungsweise von einem „Stau aus dem Nichts“. Um diese Art von Verkehrsstaus zu verhindern, ist es vorerst nötig, ein allgemeines Verständnis des Straßenverkehrs zu erlangen.

Ziel dieser Seminararbeit ist, bei dem Leser ein solches Verständnis aufzubauen, indem mithilfe einer auf einem mathematischen Modell basierten Simulation Daten ausgewertet und mit empirischen Mess-ergebnissen verglichen werden.

[1] Kraftfahr-Bundesamtes (2018)
[2] Statistisches Bundesamt

2 Definition eines Staus

Obwohl jedem die Begriffsbedeutung des Wortes „Staus“ aus dem Alltag bekannt sein sollte, muss diese für eine wissenschaftliche Arbeit genauer definiert werden. In dieser Arbeit wird ein Stau als ein Zusammentreffen von mindestens drei Autos verstanden, deren Geschwindigkeit für eine unbestimmte Zeit 0 km/h beträgt.

3 Verschiedene Messgrößen

3.1 Verkehrsfluss φ

Der Verkehrsfluss φ beschreibt die Anzahl der Fahrzeuge N, die innerhalb des Zeitintervalls Δt einen Ort x überqueren:

$$\varphi(\Delta \mathrm{N}, \Delta \mathrm{t}) = \frac{N}{\Delta \mathrm{t}}$$

Alternativ lässt sich φ auch nach der hydrodynamischen Formel bestimmen, die die Berechnung in der nachfolgenden Simulation deutlich vereinfacht:

$$\varphi(\rho, \mathrm{V}) = \rho \cdot \mathrm{V}$$

ρ entspricht hierbei der Dichte und V der durchschnittlichen Geschwindigkeit aller gemessenen Fahrzeuge.[3]

3.2 Dichte ρ

Die Dichte ρ beschreibt die Anzahl der Fahrzeuge N der Länge l in einem bestimmten Streckenabschnitt Δs. Vereinfacht lässt sie sich berechnen durch

[3] Vgl. Dr. Treiber, M. (2018), S. 13f

$$\rho = \frac{N \cdot l}{\Delta s}$$

oder in der nachfolgenden Simulation durch die Anzahl der Zellen n mit [4]

$$\rho = \frac{N}{n} .$$

4 Das Nagel-Schreckenberg-Modell

Um einen Stau am Computer simulieren zu können, werden Regeln benötigt, die besagen wie sich der Verkehrsstau verhält. Hierfür gibt es einerseits mikroskopische Modelle und andererseits makroskopische Modelle. In mikroskopischen Modellen wird jedes einzelne Fahrzeug unterschieden und unter Betrachtung der Wechselwirkung zu den anderen Fahrzeugen beschrieben, wie es sich fortbewegt. Gegensätzlich dazu wird in makroskopischen Modellen der Verkehr als zusammenhängendes „Fluid“ angenommen und die Bewegung des einzelnen Fahrzeugs außer Acht gelassen.[5] Das einfachste mikroskopische Modell ist das Nagel-Schreckenberg-Modell, welches 1992 von K. Nagel und M. Schreckenberg entworfen wurde.[6]

4.1 Vorstellung des Modells

Das Nagel-Schreckenberg-Modell ist ein sogenannter Zellularautomat, d.h. es ist diskret, also fest definiert, in Raum, Zeit und seinen Zustandsvariablen. Die stochastische Dynamik resultiert aus dem Trödel-parameter p.

Grundlegend wird eine in eine Richtung einspurig verlaufende Straße in gleichgroße Straßenabschnitte beziehungsweise Zellen unterteilt. In der

[4] Vgl. Neubert. , S. 9
[5] Vgl. Schadschneider, A. (2004), S. 2
[6] Vgl. Nagel, K./ Schreckenberg, M. (1992)

Realität entspräche die Länge dieser Zelle mit Berücksichtigung des Abstandes zwischen den Fahrzeugen von 2,5 Metern ungefähr 7,5 Metern. Jede dieser Zellen ist entweder leer oder kann von maximal einem Fahrzeug besetzt sein, welches höchstens die Geschwindigkeit v_{max} annehmen kann. Die Position beziehungsweise Zelle, in welcher sich das Auto nach einer Runde $t+1$ befindet, errechnet sich abhängig von der Geschwindigkeit v_n des jeweiligen Fahrzeugs n nach den folgenden Regeln[7]:

Regel 1: Beschleunigung

Ist v_n kleiner als v_{max}, wird die Geschwindigkeit des n-ten Fahrzeugs um eins erhöht:

$$v_n \rightarrow v'_n := \min(v_n + 1;\ v_{max})$$

Regel 2: Sicherheitsabstand

Ist vor einem Fahrzeug n die Anzahl freier Zellen d_n kleiner als v'_n, so wird die Geschwindigkeit auf d_n reduziert:

$$v'_n \rightarrow v''_n := \min(v'_n;\ d_n)$$

Regel 3: Zufälliges Trödeln

Ist $v''_n > 0$, so wird die Geschwindigkeit des Fahrzeugs n mit zufälliger Wahrscheinlichkeit p um eins reduziert:

$$v''_n \rightarrow v'''_n := \begin{cases} \max(v''_n - 1;\ 0) & \text{mit Wahrscheinlichkeit } p \\ v''_n & \text{mit Wahrscheinlichkeit } 1-p \end{cases}$$

Regel 4: Fahren

Das Fahrzeug n bewegt sich um v'''_n Zellen weiter:

$$x_n(t+1) = x_n(t) + v'''_n$$

[7] Vgl. Schadschneider, A. (2004), S. 63f

4.2 Veranschaulichung anhand eines Beispiels

Um das Ganze anschaulich darzustellen, wird der Verlauf mehrerer Runden am folgenden Beispiel gezeigt. Die ausgehende Start-konfiguration sieht hier folgendermaßen aus:
In der ersten Zelle befindet sich Fahrzeug A mit der Geschwindigkeit $v = 1$, in der dritten Zelle steht Fahrzeug B mit $v = 3$ und in der sechsten Zelle Fahrzeug C mit $v = 0$. Die maximale Geschwindigkeit beträgt $v_{max} = 3$.

Abb. 1: Beispielhafte Erklärung, Startkonfiguration, eigene Darstellung

Nun werden die in 4.1 beschriebenen Regeln auf alle Autos einzeln angewandt:

Regel 1: Die Geschwindigkeit von Fahrzeug A bzw. C erhöhen sich um 1 auf $v = 2$ bzw. $v = 1$. Fahrzeug B behält seine Geschwindigkeit bei, da sie sonst v_{max} überschreiten würde.

Abb. 2: Beispielhafte Erklärung, Konfiguration nach Regel 1, eigene Darstellung

Regel 2: Da vor Fahrzeug A nur eine Zelle frei ist, verringert sich seine Geschwindigkeit auf $v = 1$. Ebenso bei Fahrzeug B, welches zwei freie Zellen vor sich hat und somit $v = 2$ annimmt. Fahrzeug C hat eine freie Zelle vor sich, somit verändert sich seine Geschwindigkeit $v = 1$ nicht.

Abb. 3: Beispielhafte Erklärung, Konfiguration nach Regel 2, eigene Darstellung

Regel 3: In diesem Fall reduziert sich zufällig die Geschwindigkeit von Fahrzeug B um 1 auf $v = 1$.

Abb. 5: Beispielhafte Erklärung, Konfiguration nach Regel 3, eigene Darstellung

Regel 4: Alle Fahrzeuge bewegen sich um 1 weiter.

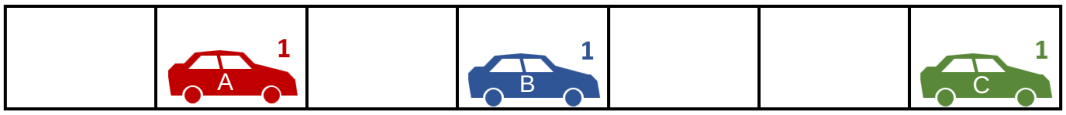

Abb. 4: Beispielhafte Erklärung, Konfiguration nach Runde 1, eigene Darstellung

Nach einer weiteren Runde wäre folgende Konstellation möglich:

Abb. 6: Beispielhafte Erklärung, Konfiguration nach Runde 2, eigene Darstellung

Da der Streckenabschnitt nur einen Ausschnitt aus einer deutlich längeren Straße darstellt, kann wie in diesem Beispiel (Abb. 6) das nachfolgende Fahrzeuge D von links aufschließen.

4.3 Die Bedeutung der Trödelparameters

Um die Regeln des Nagel-Schreckenberg-Modells realitätsbezogen zu erläutern, lässt sich folgendes festhalten: Die vierte Regel des Nagel-Schreckenberg-Modells entspricht offensichtlich der Fortbewegung. Die Beschleunigungsregel stellt den Drang des Fahrers schneller zu fahren dar. Durch die Regel des Sicherheitsabstandes wird die natürliche Vermeidung von Unfällen berücksichtigt. Der Trödelparameter p modelliert natürliche Schwankungen im Fahrverhalten eines Fahrers. Zusammen mit der zuvor genannten Regel führt das dazu, dass ein schnelleres Fahrzeug von einem langsameren, trödelnden abgebremst wird. Durch zusätzliches Trödeln des ursprünglich schnelleren Fahrzeugs können die hinter diesem Fahrzeug folgenden Fahrzeuge noch stärker abgebremst werden. Fortgeführt ist somit deutlich zu erkennen, dass diese vier Regeln theoretisch ausreichen würden, um einen Verkehrsstau nachzustellen.

Daher ist das Modell ein sehr minimalistisches Modell und kann noch weiter ausgebaut werden (4.4).[8]

Dass der Trödelparameter für eine sinnvolle Stausimulation nicht wegzudenken ist, wird an Abb. 7 ersichtlich. Ohne ein zufälliges Trödeln wäre es jedem Fahrzeugführer ohne Einschränkung möglich dauerhaft die Maximalgeschwindigkeit oder die Geschwindigkeit, die der Anzahl an freien Zellen entspricht zu fahren.

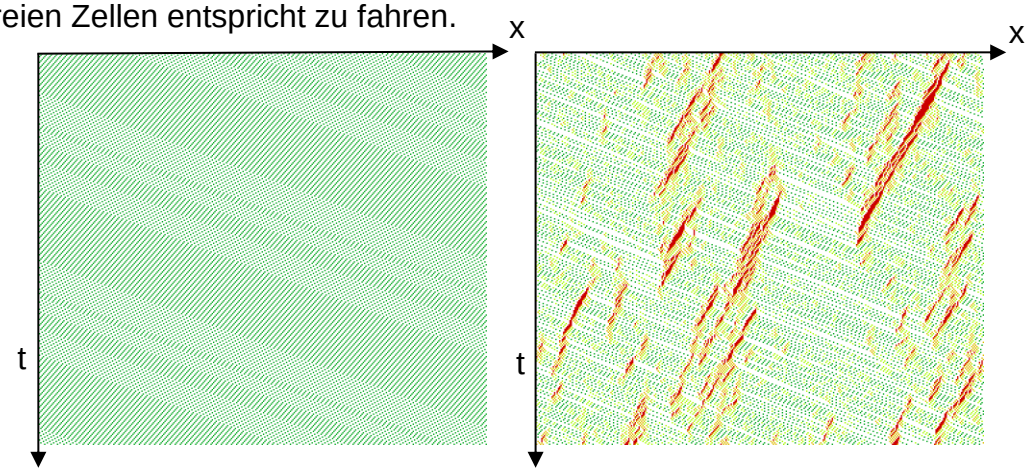

Abb. 7: Verkehrssimulation, links ohne Trödelparameter, rechts p = 0,25, eigene Darstellung

4.4 Das VDR-Modell als Erweiterung

Eine simple Erweiterung des Nagel-Schreckenberg-Modells ist das „Velocity-Dependent Randomization-“, kurz VDR-Modell. Während der Trödelparameter im Nagel-Schreckenberg-Modell für alle Fahrzeuge gleich ist, variiert dieser im VDR-Modell. Für stehende Fahrzeuge gilt der Trödelparameter p_2, der wesentlich höher ist als p. Das Nagel-Schreckenberg-Modell wird somit um eine Regel erweitert, welche vor den restlichen vier Regeln auszuführen ist[9]:

Schritt 0: Bestimmung des Trödelparameters *p*(*v*)

Für Fahrzeuge mit $v_n = 0$ gilt p_2, für fahrende Fahrzeuge ($v_n > 0$) gilt p:

$$p_n(v) := \begin{cases} p & \text{wenn } v_n > 0 \\ p_2 & \text{wenn } v_n = 0 \end{cases}$$

[8] Vgl. Schadschneider, A. (2004), S. 66
[9] Vgl. Schadschneider, A. (2000)

5 Computer-Simulation

5.1 Vorstellung des Programms

Das Nagel-Schreckenberg-Modell eignet sich äußerst gut dafür, um es an einem Computer zu simulieren. Dafür wurde für diese Arbeit mit dem Programm Greenfoot[10] eine Simulation erstellt, welche im Folgenden beschrieben wird. Sobald das Programm geöffnet wurde erscheint folgender Bildschirm:

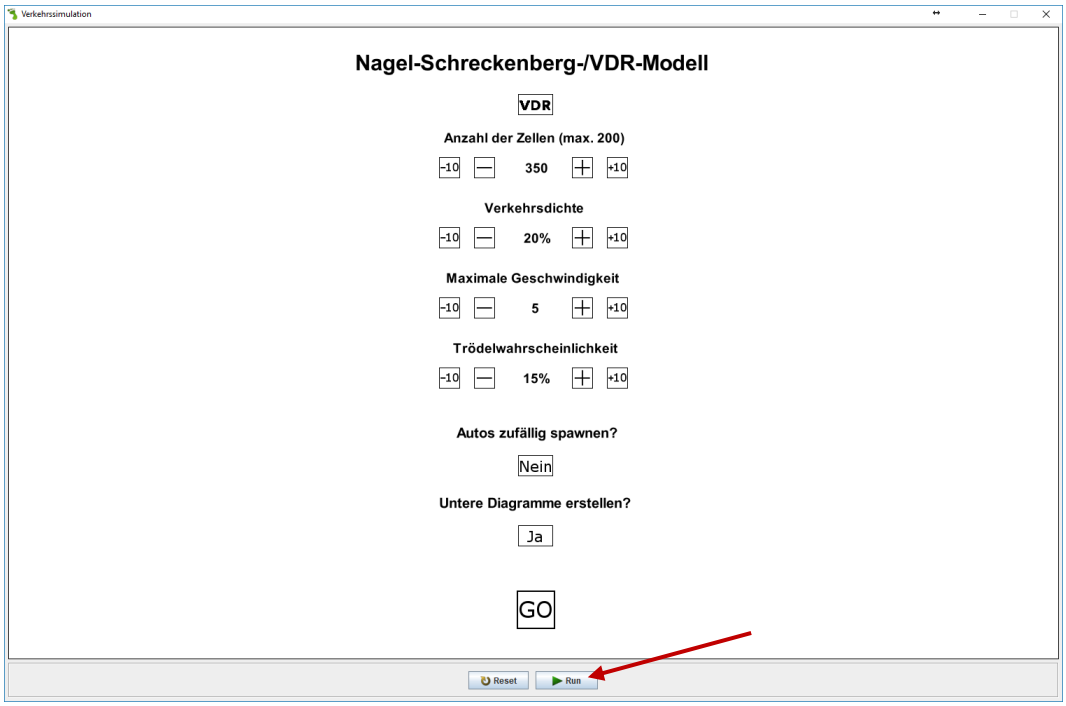

Abb. 8: Simulation, Startbildschirm, eigene Darstellung

Um das Programm selbst zu starten muss zunächst die mit rotem Pfeil gekennzeichnete Taste betätigt werden. Nun können über die darüberliegenden Tasten die Startvariablen „Anzahl der Zellen“, „Verkehrsdichte“, „Maximale Geschwindigkeit“ (v_{max}) und der Trödelparameter p („Trödelwahrscheinlichkeit“) verändert werden. Außerdem kann über die obere Taste das Nagel-Schreckenberg-Modell oder das VDR-Modell ausgewählt werden. Wenn bei „Untere Diagramme erstellen?“ „Nein“ ausgewählt ist, werden die eigentlich auf dem späteren Bildschirm rechts unten gezeichneten Diagramme nicht angezeigt und das Zeit-Ort-Diagramm

[10] https://www.greenfoot.org/home

bedeckt die ganze Fläche. Mit Betätigung der Taste „GO“ gelangt man zur eigentlichen Simulation (Abb. 8). In der linken Hälfte des Programms

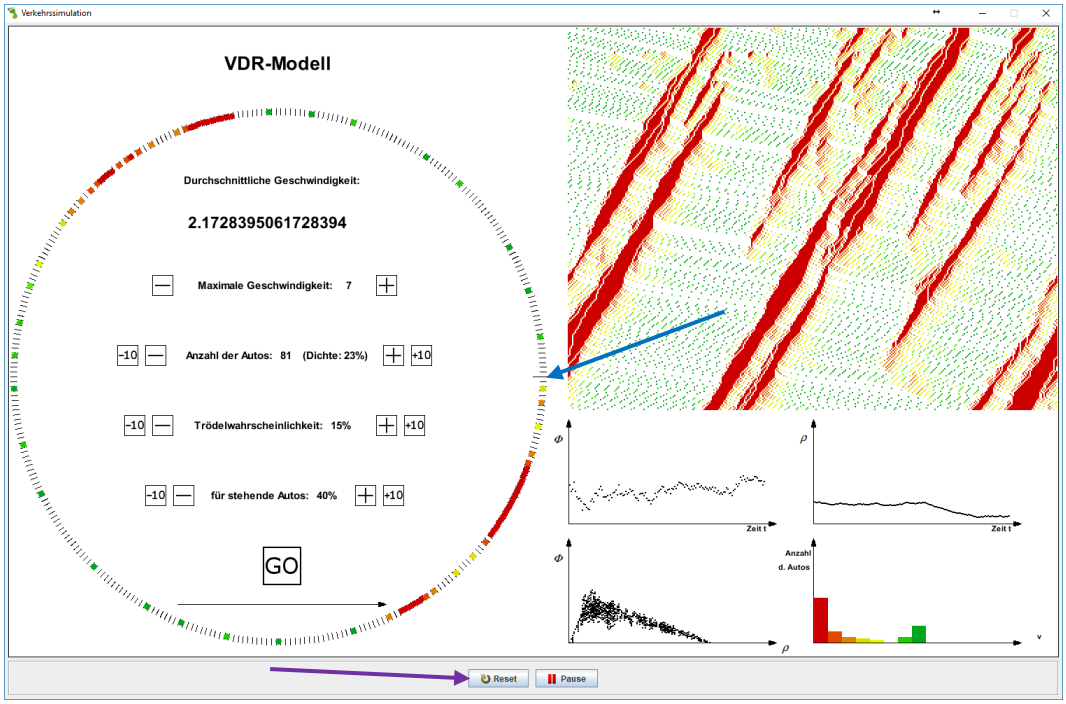

Abb. 9: Simulation, Simulationsbildschirm, eigene Darstellung

befindet sich eine kreisförmige Aneinanderreihung von standartmäßig 350 Zellen. Die Autos werden je nach Einstellung zufällig oder in gleichmäßigen Abständen auf der Kreisbahn platziert. Mit erneutem Klicken der Taste „GO“ startet die Simulation. Die Autos bewegen sich nun nach den Regeln des Nagel-Schreckenberg-/VDR-Modells entgegen dem Uhrzeigersinn fort. Innerhalb des Kreises können die zuvor festgelegten Startparameter auch während der Simulation verändert werden. Im rechten Teil werden die Messdaten jeder Runde direkt aufgezeichnet. Oben befindet sich ein Ort-Zeit-Diagramm, das die Trajektorien (also den Bewegungsverlauf) der einzelnen Fahrzeuge abbildet, wobei die y-Achse (nach unten zunehmend) die Runden beziehungsweise die Zeit darstellt und die x-Achse den Ort. Je weiter rechts sich ein Punkt befindet, desto weiter ist das zugehörige Auto auf dem Kreis von der mit blauem Pfeil markierten Zelle entfernt. Somit entspricht der Punkt, welcher im Diagramm am weitesten rechts ist dem Auto, das im Uhrzeigersinn gesehen der markierten Zelle am nächsten ist.

Die beiden darunterliegenden Diagramme darunter zeichnen im zeitlichen Verlauf jeweils den Verkehrsfluss und die Verkehrsdichte auf. Die Messungen erfolgen immer innerhalb der ersten 100 Zellen ab der markierten, ersten Zelle (gegen den Uhrzeigersinn). Darunter links befindet sich ein Fundamentaldiagramm, das die vorher genannten Größen in einem Diagramm vereint. Um ein vollständiges Fundamental-diagramm zu erhalten, bei dem alle Werte von $\rho = 0$ bis $\rho = 1$ enthalten sind, ist es nötig, die besagten Dichten über den manuellen Regler selbst herbeizuführen, da diese sonst nicht erreicht werden. Das rechts davon liegende Diagramm zeigt eine globale Geschwindigkeitsverteilung der Fahrzeuge. Bei Betätigung der violett gekennzeichneten Taste wird das Programm neu gestartet.

5.2 Funktionsweise

Die Oberklassen „World“ und „Actor“ sind in Greenfoot bereits integriert. Somit müssen Klassen wie zum Beispiel „Frame“, die beschreiben, wie die Oberfläche angezeigt wird, nicht mehr geschrieben werden. Sobald der Benutzer das Programm startet, wird ein Objekt der Klasse „MyWorld“ erstellt. Dies ist standartmäßig so in Greenfoot festgelegt. In dieser Klasse wird definiert, was auf dem Bildschirm zu sehen sei soll. Außerdem werden die Startvariablen gespeichert, um diese später an das Objekt der Klasse „Simulation“ übergeben zu können. Mithilfe der Methode act() wird überprüft, ob sich eine Variable verändert und visuell darauf reagiert. Beim Wechseln des Bildschirms zur Simulationsoberfläche werden die zuvor eingestellten Variablen an den Konstruktor des neuen Objekts der Klasse Simulation weitergegeben. Daraufhin läuft das Programm nach folgendem Schema, welches kontinuierlich wiederholt wird:

```
public void act(){
    if (Programm pausiert){
        checkButtons();      //überprüft, ob Programm gestartet wird
    } else {
        move();                          //bewegt die Autos weiter
        checkButtons();      //überprüft und reagiert auf Betätigung
                                                        einer Taste
        NaSchRegeln();          //berechnet v für die nächste Runde
        setColors();                     //ändert Farben der Autos
        drawDiagramms();            //zeichnet Diagrammdaten ein
        avgSpeedRefresh();          // aktualisiert Durchschnitts-
                                                    geschwindigkeitsanzeige
    }
}
```

Abb. 10: Schematischer Ablauf des Computer-Programms, eigene Darstellung

5.3 Empirische Überprüfung der Simulation

5.3.1 Ort-Zeit-Diagramm

Um zu zeigen, dass das Modell einen Verkehrsstau realitätsnah nachstellt, werden nun Daten der Simulation mit empirischen Daten verglichen.

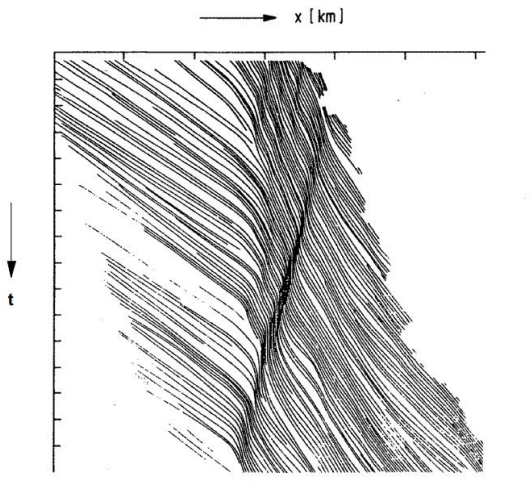

Abb. 11: Verkehrsdaten Schadschneider, Andreas: Physik des Straßenverkehrs

Abb. 11 zeigt die aus einer Serie von Luftaufnahmen abgeleiteten Trajektorien einzelner Fahrzeuge auf einer Spur einer mehrspurigen Straße. Eine Linie, die abrupt endet bzw. beginnt, gehört zu einem Fahrzeug, das auf seine Spur wechselt. [11]

Es ist deutlich zu sehen, dass sie die Fahrzeuge anfangs mit konstanter Geschwindigkeit bewegen und dann gezwungen werden abzubremsen und teilweise ganz zum Stehen kommen. Durch eine Ansammlung der Fahrzeuge kommt es zu einem sich entgegen der Fahrtrichtung bewegenden Stau mit einer Geschwindigkeit von ungefähr 15 km/h. Erwähnenswert ist außerdem, dass bei den Beobachtungen „kein äußerer Anlass für den Stau“ [12] festgestellt wurde. Demnach ist dies ein typisches Beispiel für einen Stau aus dem Nichts.

Folgende Grafik eines Staus wurde mit einer globalen Dichte $\rho = 0{,}14$, einer maximalen Geschwindigkeit von $v_{max} = 5$ und einer Trödelwahrscheinlichkeit von $p = 0{,}15$ erstellt:

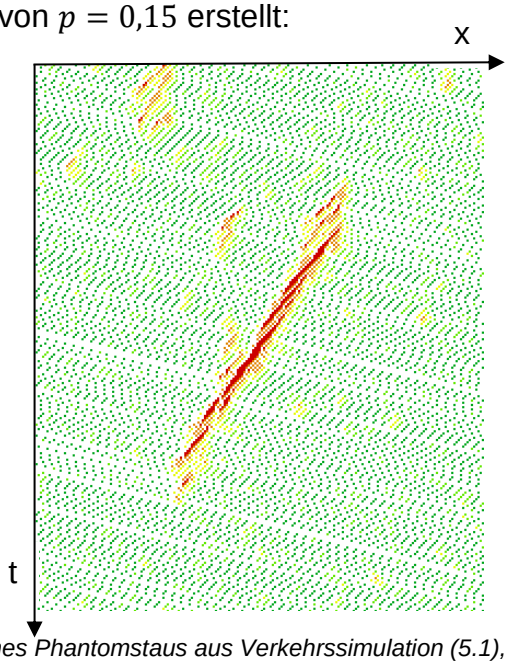

Abb. 12: Grafik eines Phantomstaus aus Verkehrssimulation (5.1), eigene Darstellung

Es wird ersichtlich, dass die Simulationsergebnisse mit den empirischen Daten übereinstimmen. Auch in Abb. 12 entsteht ein ähnlich lang andauernder Stau entgegen der Fahrtrichtung mit ähnlich vielen Autos, die durch den Stau zum Stillstand kommen. Insofern ist mit dem Nagel-

[11] Vgl. Schadschneider, A. (2004), S. 9
[12] ebd.

Schreckenberg-Modell tatsächlich bereits eine realitätsnahe Veranschaulichung eines Phantomstaus möglich.

5.3.2 Fundamentaldiagramm

Eine weitere Möglichkeit das Modell auf seine Echtheit zu überprüfen bietet das Fundmentaldiagramm, das die Größen Verkehrsfluss und Verkehrsdichte in Zusammenhang bringt.

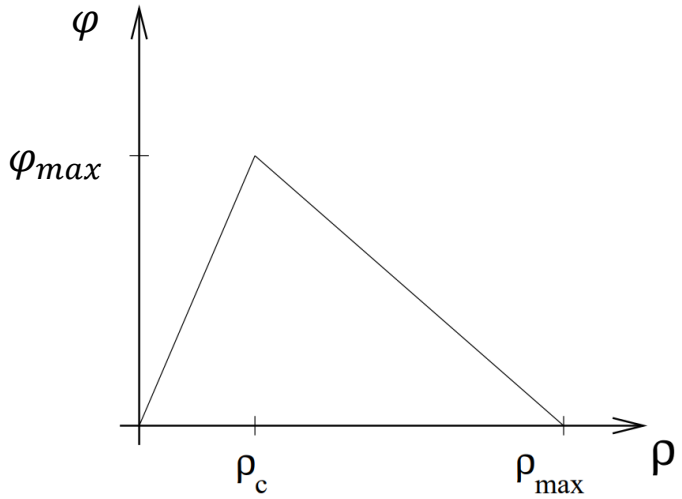

Abb. 13: Schematisches Fundamentaldiagramm des Nagel-Schreckenberg-Modells, A. Schadschneider: Physik des Straßenverkehrs, Seite 11

Abb. 13 zeigt die schematische Struktur des Fundamentaldiagramms des Nagel-Schreckenberg-Modells. Zwei verschiedene Äste sind deutlich zu erkennen: Den linken Ast mit positiver Steigung und den rechten mit negativer Steigung. Der linke sogenannte Freiflussast beschreibt Fahrer, die mit ihrer Wunschgeschwindigkeit, in diesem Fall v_{max} fahren können und nicht durch vorrausfahrende Fahrzeuge abgebremst werden. Je mehr Fahrzeuge sich mit der besagten Geschwindigkeit bewegen, desto höher ist der Verkehrsfluss nach $\varphi(\Delta N, \Delta t) = \frac{N}{\Delta t}$. Im rechten Ast ist die Dichte größer als ρ_c, die Wechselwirkungen zwischen den Fahrzeugen nimmt zu und es muss häufiger die Geschwindigkeit verringert werden. Je größer

die Dichte, umso langsamer bewegen sich die einzelnen Fahrzeuge bewegen langsam fort und der Verkehrsfluss sinkt. [13]

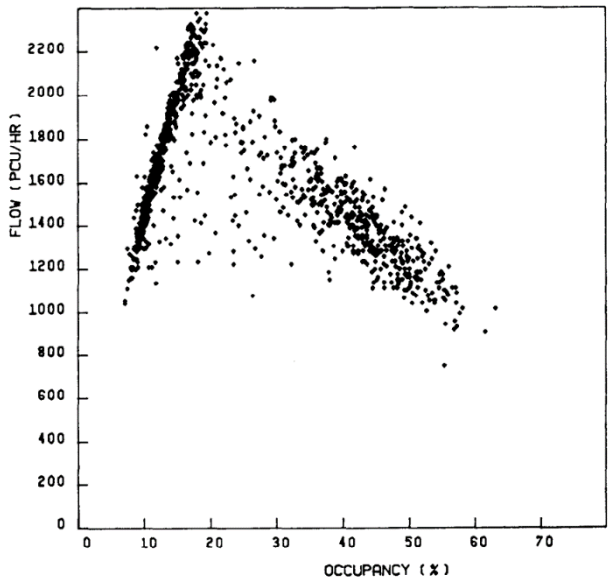

Abb. 14: Verkehrsdaten, Fred L. Hall, Brian L. Allen, Margot A. Gunter: Empirical analysis of freeway flow-density relationships, Seite 203

Abb. 14 zeigt ein Abbild eines aus empirischen Daten auf dem „Queen Elizabeth Way“ in Ontario generierten Fundamentaldiagramms. Es entspricht der schematischen Darstellung (Abb. 13). Um den Bereich des Maximums des Verkehrsflusses sieht man jedoch, dass die beiden Äste nicht in einem gemeinsamen Punkt enden. Vielmehr weitet sich der linke Ast aus, sobald er sich der Dichte $\rho \approx 0{,}2$ nähert. Abb. 15 ist daher eine genauere schematische Darstellung eines Fundamentaldiagramms.

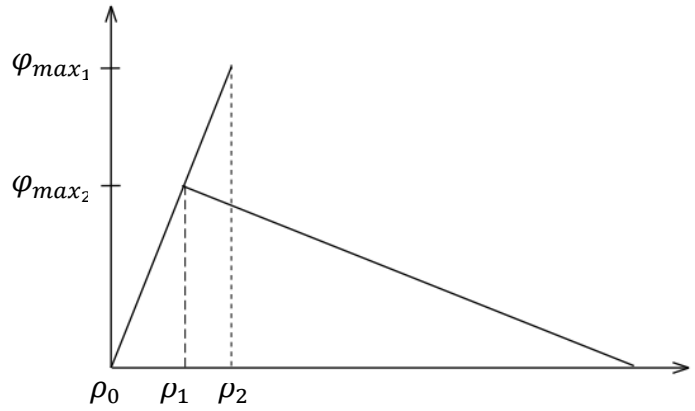

Abb. 15: Schematisches Fundamentaldiagramm des Nagel-Schreckenberg-Modells, A. Schadschneider: Physik des Straßenverkehrs, Seite 11

[13] Vgl. Schadschneider, A. (2004), S. 9

Die beiden φ_{max} zwischen den Dichten ρ_1 und ρ_2 lassen sich folgendermaßen erklären: Erhöht man die Dichte von ρ_0 bis zu ρ_2, steigt der Verkehrsfluss kontinuierlich an bis zu φ_{max_1}. Wenn die Dichte weiter erhöht wird, bricht der Verkehrsfluss zusammen und die Werte für φ liegen nun auf dem rechten Ast. Der Verkehrsfluss zwischen φ_{max_2} und φ_{max_1} gilt deshalb als metastabil. Wenn die Dichte nun wieder bis zu ρ_1 verringert wird, steigt der Verkehrsfluss an und bleibt dabei auf dem rechten Ast.[14, 15]

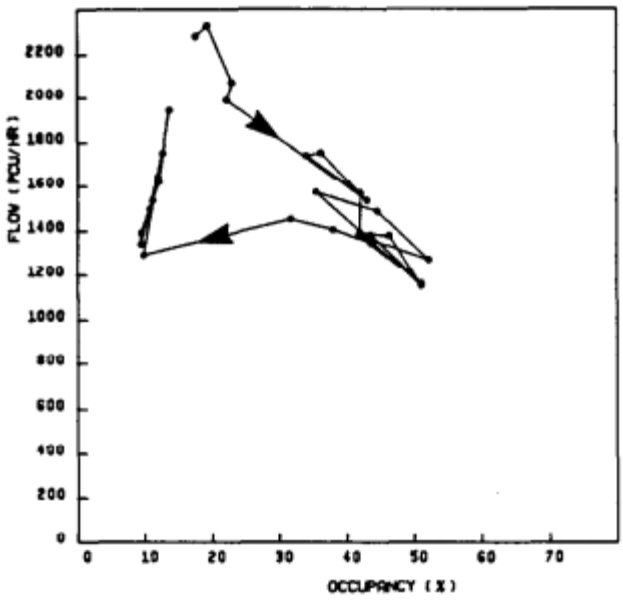

Abb. 16: Verkehrsdaten, Fred L. Hall, Brian L. Allen, Margot A. Gunter: Empirical analysis of freeway flow-density relationships, Seite 204

Abb. 15 zeigt die beschriebene Struktur anhand empirischer Daten durch den Pfeil, dessen Richtung dem zeitlichen Verlauf der Messungen entspricht deutlich auf. Mit zunehmender Dichte fällt der Verkehrsfluss ab. Später, wenn die Dichte selbst wieder abnimmt steigt der Verkehrsfluss wieder an, jedoch nicht wieder auf denselben Wert wie zuvor.

In der Simulation sind thematisierte Effekt ebenso zu beobachten. Folgende Grafiken wurden mit $v_{max} = 4$ und $p = 0{,}35$ erstellt. Die Dichte wurde dabei manuell verändert und die hinzugefügten Autos wurden stets in die letzte freie Zelle gesetzt. Die linke Grafik aus Abb. 17 entspricht hierbei Abb. 16 und die rechte entspricht Abb. 14 bzw. Abb. 15.

[14] Vgl. Schadschneider, A. (2000), S. 3f
[15] Vgl. Schadschneider, A. (2004), S. 9

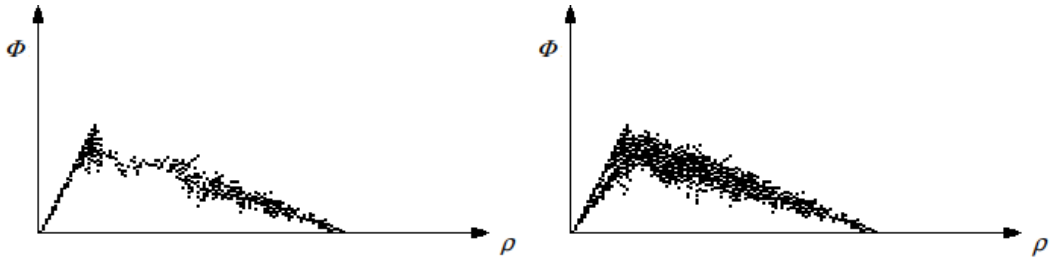

Abb. 17: Fundamentaldiagramme aus Verkehrssimulation (5.1), eigene Darstellung

6 Maßnahmen zur Stauminderung

Nachdem nun in dieser Arbeit erklärt wurde, wie es durch eine zu hohe Auslastung des Straßennetzes zu einem Stau aus dem Nichts kommen kann, stellt sich die Fragen, wie es möglich ist zumindest die Wahrscheinlichkeit dafür zu senken. Neben Mautstellen, chaotischer Ampelschaltung oder Carsharing-Systemen, die die Anzahl der angemeldeten Kraftfahrzeuge grundlegend verringern, werden im Folgenden zwei Lösungsansätze vorgestellt.

6.1 Tempolimits

Geschwindigkeitsbegrenzungen wie beispielsweise auf einer Autobahn in der Schweiz mit einer zulässigen Höchstgeschwindigkeit von 120 km/h oder in Frankreich mit maximal erlaubten 130 km/h reduzieren nicht nur die Unfallgefahr und sorgen somit für Sicherheit, sondern können auch zu einem gesteigerten Verkehrsfluss führen.

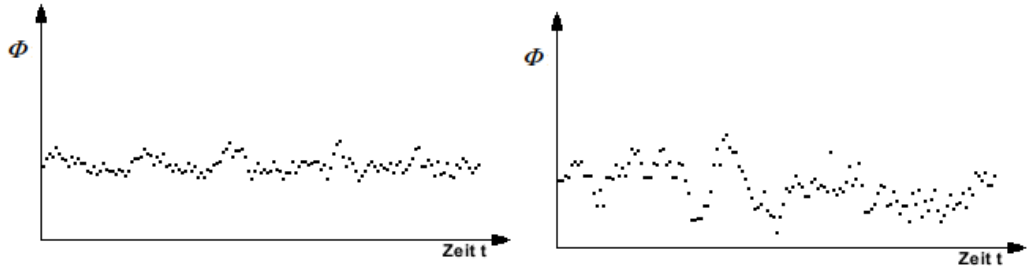

Abb. 18: Verkehrsflussdiagramme abhängig von v_{max} unter sonst gleichen Bedingungen, links: $v_{max} = 5$, rechts: $v_{max} = 9$

In Abb. 18 ist zu erkennen, dass der Verkehrsfluss bei einer maximalen Geschwindigkeit von $v_{max} = 5$ höher ist als bei $v_{max} = 9$ und außerdem konstant bleibt. Um umfassendere Ergebnisse zu erlangen wurde hier nur die Messung jeder dritten Runde im Diagramm eingezeichnet. Ausgehend von einer Größe einer Zelle von 7,5 Metern und einer vergangenen Sekunde pro Runde würde $v = 5$ einer realen Geschwindigkeit von 135 km/h und $v = 9$ würden 243 km/h entsprechen. Selbstverständlich sind nicht alle Fahrzeuge dazu fähig schneller als 200 km/h zu fahren, wodurch die schnelleren Fahrzeuge von denen, die langsamer fahren ausgebremst werden würden.

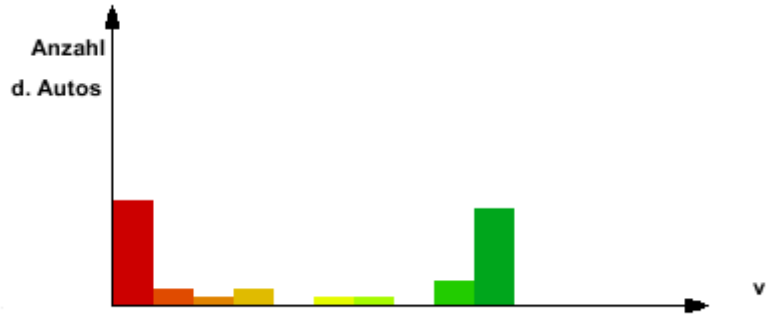

Abb. 19: Geschwindigkeitsverteilung der Autos, eigene Darstellung

Erwähnenswert ist außerdem, dass in der Simulation bei $v_{max} = 5$ nur selten ein Stau entsteht, während sich bei $v_{max} = 9$ eine Phasenseparation[16] herausbildet. Das bedeutet, dass ein Bereich mit hohem Verkehrsfluss existiert und ein Bereich eines weitgefächerten lang-andauernden Staus wie es in Abb. 19 ersichtlich wird.

Insofern wäre eine allgemeine Geschwindigkeitsbegrenzung sinnvoll und würde zu mehr Sicherheit und erhöhtem Verkehrsfluss führen. Diese Maßnahme würde zusätzlich sogar zum Umweltschutz beitragen, da mit niedrigeren Geschwindigkeiten auch die Abgasemissionen sinken.

[16] Vgl. Schadschneider, A. (2004), S. 10

6.2 Autonomes Fahren

Das autonome Fahren würde zusätzlich zur Stauminderung auch zu einem erhöhen Verkehrsfluss beitragen. Dadurch, dass das Fahrverhalten der Fahrer nicht mehr natürlichen Schwankungen unterliegt, sondern von einer Maschine gesteuert wird, würde ein Stau aus dem Nichts komplett ausgeschlossen werden. Wenn es nun aufgrund externer Einflüsse trotzdem zu einem stockenden Verkehr kommen sollte, könnte die künstliche Intelligenz des Fahrzeugs so optimiert werden, dass der Trödelparameter p_2, der beim Wiederanfahren des Fahrzeugs eine Rolle spielt durch eine wesentlich geringe Reaktionszeit minimiert wird.
Ein weiterer theoretisch denkbarer Aspekt ist die Kommunikation zwischen den selbstfahrenden Autos. Dadurch wären eine gleiche Geschwindigkeiten für alle Fahrzeuge auf derselben Spur erreichbar und langsamere Autos würden schnellere nicht ausbremsen. Es ist vergleichbar mit einem Zug, auf dem sich alle Autos befinden. Sobald ein Fahrzeug aufgrund äußerer Einflussfaktoren wie zum Beispiel einer Baustelle oder Witterungsverhältnissen seine Geschwindigkeit verringern muss, wird die Geschwindigkeit der restlichen Verkehrsteilnehmer sofort angepasst und ein Stau kann umgangen werden.

7 Fazit

Zusammenfassend lässt sich festhalten, dass das Nagel-Schreckenberg-Modell trotz seiner Minimalität sehr wohl für eine Beschreibung des Verkehrsflusses und insbesondere die Erklärung des Staus aus dem Nichts genutzt werden kann. Eine simple Erweiterung durch eine genauere Definition des Trödelparameters passt das Modell noch exakter an die Realität an.

Gut vergleichbar ist der Verkehr in unserer heutigen Welt mit einer Wettervorhersage. Auch wenn alle Ausgangsparameter bestmöglich bekannt sind, kann das Wetter trotzdem von den ursprünglichen

Berechnungen anhand der Parameter stark abweichen. Es ist fraglich, wie groß die Auswirkung eines Flügelschlags eines Schmetterlings in Florida auf das Wetter in Deutschland sein kann. Man spricht hier auch vom Schmetterlingseffekt, der das Beschriebene, also wie groß sich eine Minimale Abweichung der Anfangsbedingungen auf das Endergebnis auswirkt, thematisiert.

Überträgt man dies nun auf den Verkehr, ist dieser ebenso chaotisch wie das Wetter. Eine geringfügige Abweichung des Fahrverhaltens eines Fahrers kann durch eine Kettenreaktion zu einem Stau führen. Beispielsweise verzögert ein Autofahrer aufgrund von Müdigkeit und bremst damit das Fahrzeug hinter sich ebenso aus. Während der vordere Autofahrer wieder seine ursprüngliche Geschwindigkeit aufgenommen hat, kann der hintere Fahrer nur zeitverzögert reagieren. Wiederum hinter ihm muss ein weiteres Auto seine Geschwindigkeit noch stärker verringern, und kann ebenso erst kurze Zeit später wieder nach seinem Vordermann beschleunigen. Dieses Szenario setzt sich immer weiter nach hinten fort bis es zum Stillstand einiger Fahrzeuge kommt. Ein Phantomstau wurde geschaffen. Da diese Art von Staus eine der häufigsten sind und Staus im allgemeinen diverse Tätigkeiten verlangsamt, ist es nötig durch Maßnahmen wie Geschwindigkeitsbegrenzungen oder autonomes Fahren Staubildung zu verhindern.

8 Anhang

8.1 Abbildungsverzeichnis

8.2 Literaturverzeichnis

Dr. Treiber, Martin (2018): Verkehrsdynamik und -simulation. Skript zur Vorlesung. http://www.mtreiber.de/Vkmod_Skript/Vkmod_all.pdf [04.11.2018].

Fred L. Hall, Brian L. Allen, Margot A. Gunter: Empirical analysis of freeway flow-density relationships. Ontario 1985. S. 203f.

Kraftfahrt-Bundesamt (2018): Bestand in den Jahren 1960 bis 2018 nach Fahrzeugklassen. https://www.kba.de/DE/Statistik/Fahrzeuge/Bestand/FahrzeugklassenAufbauarten/b_fzkl_zeitreihe.html [04.11.2018].

Nagel, Kai/ Schreckenberg, Michael: A cellular automaton model for freeway traffic. Journal de Physique I, EDP Sciences, 1992, 2 (12), pp.2221-2229. <10.1051/jp1:1992277>. <jpa-00246697>.

Neubert, Lutz (o. A): Statistische Analyse von Verkehrsdaten und die Modellierung von Verkehrsfluss mittels zellularer Automaten. https://duepublico.uni-duisburg-essen.de/servlets/DerivateServlet/Derivate-5028/neubert.pdf [04.11.2018].

Schadschneider, Andreas: Physik des Straßenverkehrs. Köln 2004.

Schadschneider, Andreas: Statistical Physics of Traffic Flow. In: Physica A285, 101 (2000), S. 2.

Schadschneider, Andreas (2000): Verkehrsmodelle. http://www.thp.uni-koeln.de/~as/Mypage/verkehr.html [04.11.2018].

Statistisches Bundesamt (2013): Verkehr auf einen Blick. https://www.destatis.de/DE/Publikationen/Thematisch/TransportVerkehr/Querschnitt/BroschuereVerkehrBlick0080006139004.pdf?__blob=publicationFile [04.11.2018].

8.3 Programm-Code

Klasse MyWorld

```
import greenfoot.*;

public class MyWorld extends World
{
    private int anzahlZellen;
    private int anzahlAutos;
    private int vMax;
    private int p; //Trödelparameter
    private int dichte;
    private boolean spawnRandom;
    private boolean diagramme; //falls false, dann werden untere 4
Diagramme nicht erstellt
    private boolean nasch; //falls true, dann Nagel-Schreckenberg-
Modell; falls false VDR-Modell

    private Text anzahlZellen_text, anzahlAutos_text, vMax_text,
p_text;
    private Button zellen_less, zellen_more, autos_less,
autos_more;
    private Button vMax_less, vMax_more, p_less, p_more;

    private Button zellen_less_10, zellen_more_10, autos_less_10,
autos_more_10;
    private Button vMax_less_10, vMax_more_10, p_less_10,
p_more_10;

    private Button button_spawn, button_diagramme;
    private Button button_nasch, button_vdr;

    private Button start;

    public MyWorld()
    {
        super(1500, 900, 1);
        Greenfoot.setSpeed(100);
        //Standarteinstellung:
        anzahlZellen = 350;
        dichte = 20;
        anzahlAutos = (int) Math.round(anzahlZellen * 20 / 100);
        vMax = 5;
        p = 15;
        spawnRandom = false; //Bei Änderung dieser Einstellung (im
Code) muss auch der Button geändert werden!!
        diagramme = true;
        nasch = false;

        zellen_less = new Button(-1);
        zellen_more = new Button(1);
        autos_less = new Button(-1);
        autos_more = new Button(1);
        vMax_less = new Button(-1);
        vMax_more = new Button(1);
        p_less = new Button(-1);
        p_more = new Button(1);
```

```
        zellen_less_10 = new Button(-10);
        zellen_more_10 = new Button(10);
        autos_less_10 = new Button(-10);
        autos_more_10 = new Button(10);
        vMax_less_10 = new Button(-10);
        vMax_more_10 = new Button(10);
        p_less_10 = new Button(-10);
        p_more_10 = new Button(10);

        button_spawn = new Button(false);
        button_diagramme = new Button(true);

        button_nasch = new Button("nasch");
        button_vdr = new Button("vdr");

        start = new Button(0);

        addTexts();
        addButtons();
    }

    public void act(){
        checkButtons();
        refresh();
    }

    public void addTexts(){
        addObject(new Text("Nagel-Schreckenberg-/VDR-Modell", 40),
785, 65);

        addObject(new Text("Anzahl der Zellen (max. 200)", 25,
270), 747, 180);
        anzahlZellen_text = new Text("" + anzahlZellen, 25, 60);
        addObject(anzahlZellen_text, 758, 223);

        addObject(new Text("Verkehrsdichte", 25, 160),
getWidth()/2, 280);
        anzahlAutos_text = new Text("" + dichte + "% ", 25, 80);
        addObject(anzahlAutos_text, getWidth()/2+16, 323);

        addObject(new Text("Maximale Geschwindigkeit", 25, 260),
745, 380);
        vMax_text = new Text("" + vMax, 25, 60);
        addObject(vMax_text, 767, 423);

        addObject(new Text("Trödelwahrscheinlichkeit", 25, 250),
750, 480);
        p_text = new Text("" + p + "%  ", 25, 80);
        addObject(p_text, 765, 523);

        addObject(new Text("Autos zufällig spawnen?", 26, 240),
750, 600);
        addObject(new Text("Untere Diagramme erstellen?", 26,
290), 750, 700);
    }

    public void refresh(){
        removeObject(anzahlZellen_text);
        anzahlZellen_text = new Text("" + anzahlZellen, 25, 60);
```

```
    addObject(anzahlZellen_text, 758, 223);

    removeObject(anzahlAutos_text);
    anzahlAutos_text = new Text("" + dichte + "% ", 25, 80);
    if(dichte == 100){
        addObject(anzahlAutos_text, getWidth()/2+12, 323);
    } else if (dichte == 0){
        addObject(anzahlAutos_text, getWidth()/2+21, 323);
    }else {
        addObject(anzahlAutos_text, getWidth()/2+16, 323);
    }

    removeObject(vMax_text);
    vMax_text = new Text("" + vMax, 25, 60);
    addObject(vMax_text, 767, 423);

    removeObject(p_text);
    p_text = new Text("" + p + "% ", 25, 80);
    if(p == 100){
        addObject(p_text, getWidth()/2+11, 523);
    } else if (p == 0){
        addObject(p_text, getWidth()/2+20, 523);
    }else {
        addObject(p_text, getWidth()/2+15, 523);
    }
}

public void addButtons(){
    addObject(zellen_less, 680, 200);
    addObject(zellen_more, getWidth()-680, 200);
    addObject(autos_less, 680, 300);
    addObject(autos_more, getWidth()-680, 300);
    addObject(vMax_less, 680, 400);
    addObject(vMax_more, getWidth()-680, 400);
    addObject(p_less, 680, 500);
    addObject(p_more, getWidth()-680, 500);

    addObject(zellen_less_10, 630, 200);
    addObject(zellen_more_10, getWidth()-630, 200);
    addObject(autos_less_10, 630, 300);
    addObject(autos_more_10, getWidth()-630, 300);
    addObject(vMax_less_10, 630, 400);
    addObject(vMax_more_10, getWidth()-630, 400);
    addObject(p_less_10, 630, 500);
    addObject(p_more_10, getWidth()-630, 500);

    addObject(button_spawn, 753, 625);
    addObject(button_diagramme, 753, 725);

    addObject(button_vdr, getWidth()/2+3, 110);

    addObject(start, getWidth()/2+3, 830);
}

public void checkButtons(){
    if(zellen_more.isClicked() == true){
        anzahlZellen++;
    }
    if(zellen_less.isClicked() == true){
        anzahlZellen--;
```

```
        }
        if(autos_more.isClicked() == true){
            dichte++;
            anzahlAutos = (int) Math.round(anzahlZellen * dichte /
100);
        }
        if(autos_less.isClicked() == true){
            dichte--;
            anzahlAutos = (int) Math.round(anzahlZellen * dichte /
100);
        }
        if(vMax_more.isClicked() == true){
            vMax++;
        }
        if(vMax_less.isClicked() == true){
            vMax--;
        }
        if(p_more.isClicked() == true){
            p++;
        }
        if(p_less.isClicked() == true){
            p--;
        }

        if(zellen_more_10.isClicked() == true){
            anzahlZellen = anzahlZellen + 10;
            anzahlAutos = (int) Math.round(anzahlZellen * dichte /
100);
        }
        if(zellen_less_10.isClicked() == true){
            anzahlZellen = anzahlZellen - 10;
            anzahlAutos = (int) Math.round(anzahlZellen * dichte /
100);
        }
        if(autos_more_10.isClicked() == true){
            dichte = dichte + 10;
            anzahlAutos = (int) Math.round(anzahlZellen * dichte /
100);
        }
        if(autos_less_10.isClicked() == true){
            dichte = dichte -10;
            anzahlAutos = (int) Math.round(anzahlZellen * dichte /
100);
        }
        if(vMax_more_10.isClicked() == true){
            vMax = vMax + 10;
        }
        if(vMax_less_10.isClicked() == true){
            vMax = vMax - 10;
        }
        if(p_more_10.isClicked() == true){
            p = p + 10;
        }
        if(p_less_10.isClicked() == true){
            p = p - 10;
        }

        if(anzahlZellen < 0){
            anzahlZellen = 0;
        }
```

```
        if(anzahlAutos < 0){
            anzahlAutos = 0;
        }
        if(anzahlAutos > anzahlZellen){
            anzahlAutos = anzahlZellen;
        }
        if(vMax < 0){
            vMax = 0;
        }
        if(p < 0){
            p = 0;
        }
        if(p > 100){
            p = 100;
        }
        if(dichte > 100){
            dichte = 100;
        }
        if(dichte < 0){
            dichte = 0;
        }
        if(button_spawn.isClicked() == true){
            removeObject(button_spawn);
            if(spawnRandom == true){
                button_spawn = new Button(false);
                addObject(button_spawn, 753, 625);
                spawnRandom = false;
            } else {
                button_spawn = new Button(true);
                addObject(button_spawn, 753, 625);
                spawnRandom = true;
            }
        }
        if(button_diagramme.isClicked() == true){
            removeObject(button_diagramme);
            if(diagramme == true){
                button_diagramme = new Button(false);
                addObject(button_diagramme, 753, 725);
                diagramme = false;
            } else {
                button_diagramme = new Button(true);
                addObject(button_diagramme, 753, 725);
                diagramme = true;
            }
        }

        if(start.isClicked() == true){
            Simulation w = new Simulation(anzahlZellen,
anzahlAutos, vMax, p, dichte, diagramme, spawnRandom, nasch);
            Greenfoot.setWorld(w);
        }
        if(button_nasch.isClicked() == true){
            removeObject(button_nasch);
            addObject(button_vdr, getWidth()/2, 110);
            nasch = false;
        }
        if(button_vdr.isClicked() == true){
            removeObject(button_vdr);
            addObject(button_nasch, getWidth()/2, 110);
            nasch = true;
```

```
        }
    }

}
```

Klasse Simulation

```
import greenfoot.*;
import java.util.List;

public class Simulation extends World
{
    Zelle_Line[] zelle_line;
    Zelle[] zelle;
    Auto[] auto;
    int height, width;
    int radius;
    Text avgSpeedAnzeige, text_dichte, text_p, text_vMax, text_p2;
    int anzahlZellen;
    int anzahlAutos;
    int vMax;
    int p;
    int p2;
    double dichte_global;

    Pfeil pfx1, pfy1,  pfx2, pfy2, pfx3, pfy3, pfx4, pfy4;

    Data data;

    Point[] point1; // wird in addPfeile() instanziiert
    Point[] point2; //   ...
    Point[] point3; //   ...
    Auto[][] point4; //   ...
    boolean diagramme; //bezieht sich auf die unteren vier
Diagramme
    boolean spawnRandom;
    boolean start;
    boolean nasch;
    Button button_start, button_pause;
    Button dichte_more, dichte_less, dichte_more_10,
dichte_less_10;
    Button p_more, p_less, p_more_10, p_less_10;
    Button vMax_more, vMax_less;
    Button p2_more, p2_less, p2_more_10, p2_less_10;

    Auto[][] table;       //bezieht sich auf oberes, großes
Diagramm
    int höheDiagramm;     // -------------------"-----------------
--
    int diagrammLine;     // -------------------"-----------------
--
    int pxl;              // -------------------"-----------------
--
```

```
    public Simulation(int anzZellen_n, int anzAutos_n, int vMax_n,
int p_n, double dichte_n, boolean diagramme_n, boolean
spawnRandom_n, boolean nasch_n)
    {
        super(1500, 900, 1);
        Greenfoot.setSpeed(100);
        height = getHeight();
        width = getWidth();
        diagrammLine = 0;
        button_start = new Button(0);
        button_pause = new Button(0);
        start = false;
        anzahlZellen = anzZellen_n;
        dichte_global = dichte_n;
        anzahlAutos = anzAutos_n;
        vMax = vMax_n;
        nasch = nasch_n;
        p = p_n; // p wird p*0.01 d.h wenn p = 50 dann ist die
trödelwahrscheinlichkeit 0.5
        if(nasch == true){
            p2 = p;
        } else {
            p2 = 2*p;
        }

        if(p2 > 100){
            p2 = 100;
        }

        diagramme = diagramme_n; //wenn true werden die unteren
diagramme auch angezeigt
        spawnRandom = spawnRandom_n;
        if(diagramme == true){
            data = new Data();
            addPfeile();
            höheDiagramm = 550;
        } else {
            höheDiagramm = height;
        }

        pxl = 2; //Bestimmt die Breite der Punkte des oberen
Diagramms in Pixeln (Textur muss seperat geändert werden!)
        table = new Auto[höheDiagramm/pxl][700/pxl];

        addObject(new Pfeil(), 390, 825); //Pfeil in der Mitte des
Kreises, der die Fahrtrichtung angibt
        createZellen(anzahlZellen);
        if(spawnRandom == true){
            createAutosRandom();
        } else {
            createAutos(anzahlAutos);
        }
        NaSchRegeln(); //direkt nach spawnen der Autos MUSS auf
Sichherheitsabstand geprüft werden, da dies in der Methode act()
erst nach dem move() passiert
        addTexts();
        addButtons();
        drawDiagramm();
    }
```

```
    public void act(){
        if(start == false){
            checkButtons();
        }
        if(start == true){
            if(diagramme == true){
                drawDiagramms();//AllE Berechnungen in dieser
Methode beziehen sich auf 100 zellen ab zelle[0] (berechnet und
zeichnet Werte der unteren Diagramme ein
            }
            move(); //bewegt die Autos um v(siehe Auto.cls) Zellen
weiter
            checkButtons();
            NaSchRegeln(); //berechnet v für die nächste Runde
            setColors();
            drawDiagramm(); //zeichnet eine Zeile des oberen
Diagramms
            avgSpeedRefresh();  // aktualisiert die
Durchschnittsgeschwindigkeitsanzeige
        }
    }

    public void createZellen(int anzahl){
        zelle_line = new Zelle_Line[anzahl];
        zelle = new Zelle[anzahl];
        for (int i = 0; i < anzahl; i++){
            zelle_line[i] = new Zelle_Line();
        }
        for (int i = 0; i < anzahl; i++){
            zelle[i] = new Zelle();
        }

        radius = height/100*42;
        double d = 2*Math.PI/anzahl;
        for (int i = 0; i < anzahl; i++){ //Berechnung der
Position der einzelnen Barrieren + Platzierung
            int x = (int) Math.round(radius * Math.cos(i*d));
            x = x + width/2-365;
            int y = (int) Math.round(radius * Math.sin(i*d));
            y = height/2 - y + 50;
            addObject(zelle_line[i], x, y);
            zelle_line[i].turn(90-i*360/anzahl);
        }
        for (int i = 0; i < anzahl; i++){ //Berechnung der
Position der einzelnen Zellen + Platzierung
            int x = (int) Math.round(radius *
Math.cos((i+(0.5))*d));
            x = x + width/2-365;
            int y = (int) Math.round(radius *
Math.sin((i+(0.5))*d));
            y = height/2 - y + 50;
            addObject(zelle[i], x, y);
            // zelle[i].turn(90-i*360/anzahl); //optional, da
Zellen ohnehin unsichtbar
        }

        //Erstellung zweier extra Linien zur Markierung von
zelle_linie[0]
        Zelle_Line z0 = new Zelle_Line();
        Zelle_Line z1 = new Zelle_Line();
```

```
        addObject(z0, zelle_line[0].getX()+10,
zelle_line[0].getY());
        addObject(z1, zelle_line[0].getX()-10,
zelle_line[0].getY());
        z0.setRotation(90);
        z1.setRotation(90);
    }

    public void createAutos(int anzahl){
        auto = new Auto[anzahlZellen];
        for(int i = 0; i < anzahlAutos; i++){ //Bestimmung der
Position jeden Autos ANHAND der Zellen, nicht des Winkels
            int pos = (int) Math.round(anzahlZellen * i
/anzahlAutos);
            zelle[pos].setGefüllt(true);
            int x = zelle[pos].getX();
            int y = zelle[pos].getY();
            auto[i] = new Auto(vMax, p);
            addObject(auto[i], x, y);
            auto[i].setColor();
        }
    }

    public void createAutosRandom(){
        auto = new Auto[anzahlZellen];
        for (int i = 0; i < anzahlAutos; i++){
            int k = Greenfoot.getRandomNumber(anzahlZellen);
            while(zelle[k].getGefüllt() == true){
                k = Greenfoot.getRandomNumber(anzahlZellen);
            }
            auto[i] = new Auto(vMax, p);
            addObject(auto[i], zelle[k].getX(), zelle[k].getY());
            zelle[k].setGefüllt(true);
            auto[i].setColor();
        }
    }

    public void setColors(){
        for(int i = 0; i < anzahlAutos; i++){
            if(auto[i] != null){
                auto[i].setColor();
            }
        }
    }

    public void NaSchRegeln(){
        for (int i = 0; i < anzahlAutos; i++){
            if(auto[i] != null){

                //Regel 0 (aus VDR-Modell)
                int p_n = p;
                if(nasch == false){
                    if(auto[i].getSpeed() == 0){ //p_n(eu)
entspricht HIER p2
                        p_n = p2;
                    }
                }

                //Regel 1
                if (auto[i].getSpeed() < vMax){
```

```
                    auto[i].setSpeed(auto[i].getSpeed()+1);
                }

                //Regel 2
                int z = -1;
                for (int k = 0; k < anzahlZellen; k++){
                    if((zelle[k].getX() == auto[i].getX()) &&
(zelle[k].getY() == auto[i].getY())){
                        z = k;
                        break;
                    }
                }
                if(z < 0){
                    System.out.println("Nicht allle Autos befinden
sich in einer Zelle!!!");
                }

                int frei = 0;
                for(int j = 0; j < auto[i].getSpeed(); j++){
                    if(auto[i].getSpeed() == 0){
                        break;
                    }
                    int sum = z + j + 1;
                    if(sum >= anzahlZellen){
                        sum = sum - anzahlZellen;
                    }
                    if(zelle[sum].getGefüllt() == false){
                        frei++;
                    } else if(zelle[sum].getGefüllt() == true){
                        break;
                    }
                }
                auto[i].setSpeed(frei);

                //Regel 3
                if (p != 0){
                    if(auto[i].getSpeed()>0){
                        if(Greenfoot.getRandomNumber(100) < p_n){
                            auto[i].setSpeed(auto[i].getSpeed()-
1);
                        }
                    }
                }
            }
        }
    }

    public void move(){ //Regel 4
        for(int i = 0; i < anzahlAutos; i++){
            if(auto[i] != null){
                int v = auto[i].getSpeed();
                int z = -1;
                //Zelle in der sich Auto nr. i befindet wird
ermittelt
                for (int k = 0; k < anzahlZellen; k++){
                    if((zelle[k].getX() == auto[i].getX()) &&
(zelle[k].getY() == auto[i].getY())){
                        z = k;
                    }
                }
```

```
                if(z < 0){  //check ob alle Autos in Zellen
                    System.out.println("Nicht alle Autos befinden
sich in einer Zelle!!!");
                }
                zelle[z].setGefüllt(false); //da Auto ja die Zelle
verlässt

                //check ob Auto nicht "über die Anzahl der Zellen
hinausfährt"
                int sum = z + v;
                if(sum >= anzahlZellen){
                    sum = sum - anzahlZellen;
                }

                //Neuplatzierung des Autos
                int x_new = zelle[sum].getX();
                int y_new = zelle[sum].getY();
                auto[i].setLocation(x_new, y_new);
                zelle[sum].setGefüllt(true);
            }
        }
    }

    public void addTexts(){
        if(nasch == true){
            addObject(new Text("Nagel-Schreckenberg-Modell", 35),
500, 65);
        }
        if(nasch == false){
            addObject(new Text("VDR-Modell", 35), 598, 70);
        }
        addObject(new Text("Durchschnittliche Geschwindigkeit:",
20), 540, 245);
        text_vMax = new Text("Maximale Geschwindigkeit:     " +
vMax, 20, 240);
        addObject(text_vMax, 380, 395);
        int dichte_n = (int) Math.round(dichte_global);
        text_dichte = new Text("Anzahl der Autos:   " +
anzahlAutos + "    (Dichte: " + dichte_n + "%)", 20, 280);
        addObject(text_dichte, 383, 495);
        text_p = new Text("Trödelwahrscheinlichkeit:   " + p + "%
", 20, 250);
        addObject(text_p, 380, 595);

        if(nasch == true){
            text_p2 = new Text("für stehende Autos: siehe oben",
20, 235);
        }
        if(nasch == false){
            text_p2 = new Text("für stehende Autos:   " + p2 + "%
", 20, 190);
        }
        addObject(text_p2, 374, 695);

        if(diagramme == true){ // erstellt Beschriftung der Achsen
            addObject(new Text("Zeit t", 14, 80), 1085, 745);
            addObject(new Text("Zeit t", 14, 80), 1085+350, 745);
            addObject(new Text("v", 14, 30), 1085+390, 1200);
            addObject(new Text("Anzahl", 14, 80), 1095+45, 780);
            addObject(new Text("d. Autos", 14, 80), 1085+45, 800);
```

```
            addObject(new Text(0), 786, 590);
            addObject(new Text(1), 786+350, 590);
            addObject(new Text(0), 786, 590+170);
            addObject(new Text(1), 1110, 890);
        }

        double totalSpeed = 0;
        for (int i = 0; i < anzahlAutos; i++){
            totalSpeed = totalSpeed + auto[i].getSpeed();
        }
        double avgSpeed = totalSpeed/anzahlAutos;
        avgSpeedAnzeige = new Text(""+ avgSpeed, 30);
        addObject(avgSpeedAnzeige, 540, 300);
        addObject(button_start, 390, 770);
    }

    public void addPfeile(){
        //Diagramm 1
        pfx1 = new Pfeil("x");
        addObject(pfx1, 948, 710);
        pfy1 = new Pfeil("y");
        addObject(pfy1, 800, 635);
        pfy1.turn(270);
        //Diagramm 2
        pfx2 = new Pfeil("x");
        addObject(pfx2, 948+350, 710);
        pfy2 = new Pfeil("y");
        addObject(pfy2, 800+350, 635);
        pfy2.turn(270);
        //Diagramm 3
        pfx3 = new Pfeil("x");
        addObject(pfx3, 948, 880);
        pfy3 = new Pfeil("y");
        addObject(pfy3, 800, 805);
        pfy3.turn(270);
        //Diagramm 4
        pfx4 = new Pfeil("x");
        addObject(pfx4, 948+350, 880);
        pfy4 = new Pfeil("y");
        addObject(pfy4, 800+350, 805);
        pfy4.turn(270);

        point1 = new Point[140];
        point2 = new Point[140];
        point3 = new Point[140];
        point4 = new Auto[140][70];

    }

    public void avgSpeedRefresh(){
        double totalSpeed = 0;
        for (int i = 0; i < anzahlAutos; i++){
            if(auto[i] != null){
                totalSpeed = totalSpeed + auto[i].getSpeed();
            }
        }
        removeObject(avgSpeedAnzeige);
        double avgSpeed = totalSpeed/anzahlAutos; // Neuberechnung
der Durchschnittsgeschwindigkeit
        avgSpeedAnzeige = new Text(" "+ avgSpeed, 30);
```

```
        addObject(avgSpeedAnzeige, 540, 300);
    }

    public void drawDiagramm(){ //oberes Diagramm
        int höhe = höheDiagramm;
        if((5 + pxl * diagrammLine) > höhe){
            for(int i = 0; i < 700/pxl; i++){ //entfernen der
obersten Zeile, falls unten kein Platz mehr
                if(table[0][i] != null){
                    removeObject(table[0][i]);
                }
            }

            for(int z = 1; z < (höhe/pxl)-1; z++){
                for(int s = 0; s < 700/pxl; s++){
                    if(table[z][s] != null){
                        int x = table[z][s].getX();
                        int y = table[z][s].getY()-pxl;
                        table[z][s].setLocation(x, y);
                    }
                }
            }

            Auto[][] table_n = new Auto[höhe/pxl][700/pxl];
            for(int z = 0; z <((höhe/pxl)-1); z++){
                for(int s = 0; s < 700/pxl; s++){
                    if(table[z+1][s] != null){
                        table_n[z][s] = table[z+1][s];
                    }
                }
            }
            table = table_n;
            diagrammLine--;
        }
        if((1 + pxl * diagrammLine) < height){
            int[] auto_row = new int[700/pxl];
            int anzAutosProZ = 700/pxl;  // anzahl der Autos pro
Zeile
            if(anzAutosProZ > anzahlZellen){
                anzAutosProZ = anzahlZellen;
            }
            for(int z = 0; z < anzAutosProZ; z++){
                if(zelle[z].getGefüllt() == true){
                    for(int i = 0; i < anzahlAutos; i++){
                        if(auto[i] != null){
                            if(zelle[z].getX() == auto[i].getX()
&& zelle[z].getY() == auto[i].getY()){
                                auto_row[z] = auto[i].getSpeed();
                                break;
                            }
                        }
                    }
                } else {
                    auto_row[z] = -1;
                }
            }

            for(int i = 0; i < anzAutosProZ; i++){
                if(auto_row[i] >= 0){
                    Auto a = new Auto(vMax, p, auto_row[i]);
```

```
                    table[diagrammLine][i] = a;
                    addObject(table[diagrammLine][i], 800 + pxl *
i, 3 + pxl * diagrammLine);
                    table[diagrammLine][i].setColorPxl();
                }
            }
            diagrammLine++;
        }
    }

    public void addButtons(){
        dichte_more = new Button(1);
        addObject(dichte_more, 550, 470);
        dichte_less = new Button(-1);
        addObject(dichte_less, 210, 470);
        dichte_more_10 = new Button(10);
        addObject(dichte_more_10, 590, 470);
        dichte_less_10 = new Button(-10);
        addObject(dichte_less_10, 170, 470);

        p_more = new Button(1);
        addObject(p_more, 540, 570);
        p_less = new Button(-1);
        addObject(p_less, 220, 570);
        p_more_10 = new Button(10);
        addObject(p_more_10, 580, 570);
        p_less_10 = new Button(-10);
        addObject(p_less_10, 180, 570);

        vMax_more = new Button(1);
        addObject(vMax_more, 540, 370);
        vMax_less = new Button(-1);
        addObject(vMax_less, 220, 370);

        if(nasch == false){
            p2_more = new Button(1);
            addObject(p2_more, 510, 670);
            p2_less = new Button(-1);
            addObject(p2_less, 250, 670);
            p2_more_10 = new Button(10);
            addObject(p2_more_10, 550, 670);
            p2_less_10 = new Button(-10);
            addObject(p2_less_10, 210, 670);
        }
    }

    public void checkButtons(){
        if((button_start.isClicked() == true) && (start ==
false)){
            start = true;
            removeObject(button_start);
            addObject(button_pause, 390, 770);
        }
        if((button_pause.isClicked() == true) && (start == true)){
            start = false;
            removeObject(button_pause);
            addObject(button_start, 390, 770);
        }

        if(dichte_more.isClicked() == true){
```

```
            if(anzahlAutos < anzahlZellen){
                if(auto[anzahlAutos] == null && anzahlAutos <
anzahlZellen){
                    auto[anzahlAutos] = new Auto(vMax, p);

                    //zufälliger Zelle wird berechnet
                    int z =
Greenfoot.getRandomNumber(anzahlZellen);
                    while(zelle[z].getGefüllt() == true){
                        z =
Greenfoot.getRandomNumber(anzahlZellen);
                    }

                    if(zelle[z].getGefüllt() == false){
                        zelle[z].setGefüllt(true);
                        addObject(auto[anzahlAutos],
zelle[z].getX(), zelle[z].getY());
                        for(int k = z; k >= z-vMax; k--){
                            int m = k;
                            if(m < 0){ //verhindert, dass auto[-1]
gecheckt wird (vorallem wichtig für dichte_more_10)
                                m = m+anzahlZellen;
                            }
                            if(zelle[m].getGefüllt() == true){
                                for(int i = 0; i < anzahlAutos;
i++){ //verhindert direkten zusammenstoß nach spawn
                                    if((auto[i].getX() ==
zelle[m].getX()) && (auto[i].getY() == zelle[m].getY())){
                                        auto[i].setSpeed(0);
                                    }
                                }
                            }
                        }

                    }

                    anzahlAutos++;
                }
                //Textfeld aktualisieren
                removeObject(text_dichte);
                int dichte_n = (int)
Math.round(anzahlAutos*100/anzahlZellen);
                text_dichte = new Text("Anzahl der Autos:   " +
anzahlAutos + "    (Dichte: " + dichte_n + "%)", 20, 280);
                addObject(text_dichte, 383, 495);
            }
        }
        if(anzahlAutos > 1){
            if(dichte_less.isClicked() == true){
                for(int z = 0; z < anzahlZellen; z++){
                    if((auto[anzahlAutos-1].getX() ==
zelle[z].getX()) && (auto[anzahlAutos-1].getY() ==
zelle[z].getY())){
                        zelle[z].setGefüllt(false);
                    }
                }
                removeObject(auto[anzahlAutos-1]);
                auto[anzahlAutos-1] = null;
                anzahlAutos--;
                //Textfeld aktualisieren
```

```
                removeObject(text_dichte);
                int dichte_n = (int)
Math.round(anzahlAutos*100/anzahlZellen);
                text_dichte = new Text("Anzahl der Autos:   " +
anzahlAutos + "    (Dichte: " + dichte_n + "%)", 20, 280);
                addObject(text_dichte, 383, 495);
            }
        }
        if(dichte_more_10.isClicked() == true){
            int w = 10;
            if(anzahlAutos == anzahlZellen){
                w = 0;
            } else if(anzahlAutos+10 > anzahlZellen){
                w = anzahlZellen - anzahlAutos - 1;
                // w = 1;
            }
            for(int j = 0; j < w; j++){
                if(auto[anzahlAutos] == null && anzahlAutos <
anzahlZellen){
                    auto[anzahlAutos] = new Auto(vMax, p, 0);

                    //zufällige Zelle wird berechnet
                    int z =
Greenfoot.getRandomNumber(anzahlZellen);
                    while(zelle[z].getGefüllt() == true){
                        z =
Greenfoot.getRandomNumber(anzahlZellen);
                    }

                    if(zelle[z].getGefüllt() == false){
                        zelle[z].setGefüllt(true);
                        addObject(auto[anzahlAutos],
zelle[z].getX(), zelle[z].getY());
                        for(int k = z; k >= z-vMax; k--){ // autos
hinter dem neuen auto bekommen v = 0
                            int m = k;
                            if(m < 0){
                                m = m+anzahlZellen;
                            }
                            if(zelle[m].getGefüllt() == true){
                                for(int i = 0; i < anzahlAutos;
i++){
                                    if((auto[i].getX() ==
zelle[m].getX()) && (auto[i].getY() == zelle[m].getY())){
                                        auto[i].setSpeed(0);
                                    }
                                }
                            }
                        }

                    }
                    anzahlAutos++;
                }
            }
            //Textfeld aktualisieren
            removeObject(text_dichte);
            int dichte_n = (int)
Math.round(anzahlAutos*100/anzahlZellen);
            text_dichte = new Text("Anzahl der Autos:   " +
anzahlAutos + "    (Dichte: " + dichte_n + "%)", 20, 280);
```

```
            addObject(text_dichte, 383, 495);
        }
        if(dichte_less_10.isClicked() == true){
            int w = 10;
            if(anzahlAutos ==1){
                w = 0;
            } else if(anzahlAutos <= 10){
                w = anzahlAutos - 1;
            }
            for(int j = 0; j < w; j++){
                for(int z = 0; z < anzahlZellen; z++){
                    if((auto[anzahlAutos-1].getX() ==
zelle[z].getX()) && (auto[anzahlAutos-1].getY() ==
zelle[z].getY())){
                        zelle[z].setGefüllt(false);
                    }
                }
                removeObject(auto[anzahlAutos-1]);
                auto[anzahlAutos-1] = null;
                anzahlAutos--;
            }
            //Textfeld aktualisieren
            removeObject(text_dichte);
            int dichte_n = (int)
Math.round(anzahlAutos*100/anzahlZellen);
            text_dichte = new Text("Anzahl der Autos:   " +
anzahlAutos + "    (Dichte: " + dichte_n + "%)", 20, 280);
            addObject(text_dichte, 383, 495);
        }

        if(p_more.isClicked() == true){
            removeObject(text_p);
            p++;
            if(p > 100){
                p = 100;
            }
            text_p = new Text("Trödelwahrscheinlichkeit:   " + p +
"% ", 20, 250);
            addObject(text_p, 380, 595);
        }
        if(p_less.isClicked() == true){
            removeObject(text_p);
            p--;
            if(p < 0){
                p = 0;
            }
            text_p = new Text("Trödelwahrscheinlichkeit:   " + p +
"% ", 20, 250);
            addObject(text_p, 380, 595);
        }
        if(p_more_10.isClicked() == true){
            removeObject(text_p);
            p = p + 10;
            if(p > 100){
                p = 100;
            }
            text_p = new Text("Trödelwahrscheinlichkeit:   " + p +
"% ", 20, 250);
            addObject(text_p, 380, 595);
        }
```

```
        if(p_less_10.isClicked() == true){
            removeObject(text_p);
            p = p - 10;
            if(p < 0){
                p = 0;
            }
            text_p = new Text("Trödelwahrscheinlichkeit:   " + p +
"% ", 20, 250);
            addObject(text_p, 380, 595);
        }
        if(vMax_more.isClicked()){
            removePoints();
            removeObject(text_vMax);
            vMax++;
            for(int i = 0; i < anzahlAutos; i++){
                auto[i].setVMax(vMax);
            }
            text_vMax = new Text("Maximale Geschwindigkeit:     "
+ vMax, 20, 240);
            addObject(text_vMax, 380, 395);
        }
        if(vMax_less.isClicked()){
            removePoints();
            removeObject(text_vMax);
            vMax--;
            if(vMax < 0){
                vMax = 0;
            }
            for(int i = 0; i < anzahlAutos; i++){
                auto[i].setVMax(vMax);
                if(auto[i].getSpeed() > vMax){
                    auto[i].setSpeed(vMax);
                }
            }
            text_vMax = new Text("Maximale Geschwindigkeit:     "
+ vMax, 20, 240);
            addObject(text_vMax, 380, 395);
        }

        if(nasch == false){
            if(p2_more.isClicked() == true){
                removeObject(text_p2);
                p2++;
                if(p2 > 100){
                    p2 = 100;
                }
                text_p2 = new Text("für stehende Autos:   " + p2 +
"% ", 20, 190);
                addObject(text_p2, 374, 695);
            }
            if(p2_less.isClicked() == true){
                removeObject(text_p2);
                p2--;
                if(p2 < 0){
                    p2 = 0;
                }
                text_p2 = new Text("für stehende Autos:   " + p2 +
"% ", 20, 190);
                addObject(text_p2, 374, 695);
            }
```

```
            if(p2_more_10.isClicked() == true){
                removeObject(text_p2);
                p2 = p2 + 10;
                if(p2 > 100){
                    p2 = 100;
                }
                text_p2 = new Text("für stehende Autos:   " + p2 +
"% ", 20, 190);
                addObject(text_p2, 374, 695);
            }
            if(p2_less_10.isClicked() == true){
                removeObject(text_p2);
                p2 = p2 - 10;
                if(p2 < 0){
                    p2 = 0;
                }
                text_p2 = new Text("für stehende Autos:   " + p2 +
"% ", 20, 190);
                addObject(text_p2, 374, 695);
            }
        }

    }

    public void drawDiagramms(){
        if(data.getXWert() >=140){
            data.setXWert(data.getXWert()-1);
            moveDiagramm();
        }

        if((data.getRunde()%1) == 0){
            double dichte = calculateDichte();
            double vAvg = calculateVAvg();
            double flow = calculateVerkehrsfluss(dichte, vAvg);
            // System.out.println("Flow "+flow+"    Dichte
"+dichte+"    VAVG "+vAvg);
            drawDiagramm1(flow);
            drawDiagramm2(dichte);
            if((data.getXWert()) > 50){ //Verhindert, dass in den
ersten Runden "verfälschte" Werte eingezeichnet werden
                drawDiagramm3(flow, dichte);
            }
            drawDiagramm4();

            data.setXWert(data.getXWert()+1);
        }

        data.setRunde(data.getRunde()+1);
    }

    public double calculateDichte(){
        double dichte = 0;
        int m = 100; // zellen, die gemessen werden
        if(anzahlZellen < 100){
            m = anzahlZellen;
        }
        for(int z = 0; z < m; z++){
            if(zelle[z].getGefüllt() == true){
                dichte++;
            }
```

```
        }
        dichte = dichte/m;
        return dichte;
    }

    public double calculateVAvg(){ //berechnet
Durchschnittsgeschwindigkeit in den ersten 100 Zellen
        double anz = 0; // Anzahl der Autos in diesen 100 zellen
        double vGes = 0; // alle Geschwindigkeiten aller Autos in
diesen 100 Zellen zusammen addiert
        int m = 100; // zellen, die gemessen werden
        if(anzahlZellen < 100){
            m = anzahlZellen;
        }
        for(int z = 0; z < m; z++){
            for(int i = 0; i < anzahlAutos; i++){
                if(auto[i] != null){
                    if((zelle[z].getX() == auto[i].getX()) &&
(zelle[z].getY() == auto[i].getY())){
                        anz++;
                        vGes = vGes + auto[i].getSpeed();
                    }
                }
            }
        }
        double vAvg = vGes/anz;
        return vAvg;
    }

    public double calculateVerkehrsfluss(double d, double v){
        double flow = v * d;
        return flow;
    }

    public void drawDiagramm1(double flow){
        int i = data.getXWert();
        int y = (int) Math.round(flow*100);
        point1[i] = new Point();
        addObject(point1[i], 802 + (2*data.getXWert()), 710 -
(y));
    }

    public void drawDiagramm2(double d){
        int i = data.getXWert();
        int y = (int) Math.round(d*100);
        point2[i] = new Point();
        addObject(point2[i], 1152 + (2*data.getXWert()), 710 -
(y));
    }

    public void drawDiagramm3(double f, double d){
        int i = data.getXWert();
        int x = (int) Math.round(d*100);
        int y = (int) Math.round(f*100);
        // System.out.println(y);
        point3[i] = new Point();
        addObject(point3[i], 802 + (2*x), 880 - (1*y));
        if(point3[i].checkDoubles()){ //löscht den Punkt wieder,
wenn bereits ein anderer an gleicher Stelle vorhanden ist
            removeObject(point3[i]);
```

```
        }
        // System.out.println(point3[i].checkDoubles());
    }

        public void drawDiagramm4(){
        if(anzahlAutos > 0){
            int balkenbreite = (int) Math.round(140/(vMax+1));
            if(vMax < 12){
                balkenbreite = 10;
            }
            removePoints();
            for(int v = 0; v <= vMax; v++){ //geht alle
Geschwindigkeiten durch und erstellt so die Balken
                int anzahl = 0;
                for(int i = 0; i < anzahlAutos; i++){ //sucht alle
Autos mit besagter Geschwindigkeit
                    if(auto[i].getSpeed() == v){
                        anzahl++;
                    }
                }
                int höhe = (int)
Math.round(70*anzahl/anzahlAutos);

                for(int x = v*balkenbreite; x <
(v+1)*balkenbreite; x++){
                    for(int y = 0; y < höhe; y++){
                        point4[x][y] = new Auto(vMax, p, v);
                        System.out.println("" + x + "|" + y);
                        point4[x][y].setColorPxl();
                        addObject(point4[x][y], 1152+2*x, 879-
2*y);
                    }

                }
            }
        }
    }

    public void moveDiagramm(){ //bewegt Punkte der Diagramme 1
und 2
        //Diagramm1
        Point[] p1 = new Point[140];
        removeObject(point1[0]);
        for (int s = 0; s < 139; s++){
            p1[s] = point1[s+1];
            point1[s+1].setLocation(point1[s+1].getX()-2,
point1[s+1].getY());
        }
        point1 = p1;

        //Diagramm2
        Point[] p2 = new Point[140];
        removeObject(point2[0]);
        for (int s = 0; s < 139; s++){
            p2[s] = point2[s+1];
            point2[s+1].setLocation(point2[s+1].getX()-2,
point2[s+1].getY());
        }
        point2 = p2;
    }
```

```
    public void removePoints(){ //löscht die Punkt des 4.
Diagramms
        for(int x = 0; x < 140; x++){
            for(int y = 0; y < 70; y++){
                if(point4[x][y] != null){
                    removeObject(point4[x][y]);
                }
            }
        }
    }
}
```

Klasse Auto

```
import greenfoot.*;

public class Auto extends Actor
{
    private int v;
    private int vMax;
    private int p; // ---- >(1/p) >> wahrscheinlichkeit mit der
ein auto um 1 abbremst

    public Auto(int vM, int pNew){
        vMax = vM;
        p = pNew;
        v = vMax;
    }

    public Auto(){
        vMax = 4;
        p = 5;
        v = 1;
    }

    public Auto(int i){
        setImage("diagramm_point.png");
    }

    public Auto(int vM, int pNew, int v_n){
        vMax = vM;
        p = pNew;
        v = v_n;
        setImage("zelle_empty.png");
    }

    public void setColor(){
        int pic_nr;
        if(vMax == 0){
            pic_nr = 0;
        } else {
            pic_nr = (int) Math.round((v*10/vMax));
```

```
        }
        // int pic_nr = Greenfoot.getRandomNumber(vMax+1);
        switch(pic_nr){
            case 0:
            setImage("v0.png");
            break;
            case 1:
            setImage("v1.png");
            break;
            case 2:
            setImage("v2.png");
            break;
            case 3:
            setImage("v3.png");
            break;
            case 4:
            setImage("v4.png");
            break;
            case 5:
            setImage("v5.png");
            break;
            case 6:
            setImage("v6.png");
            break;
            case 7:
            setImage("v7.png");
            break;
            case 8:
            setImage("v8.png");
            break;
            case 9:
            setImage("v9.png");
            break;
            default:
            setImage("v9.png");
        }
    }

    public void setColorPxl(){ //bestimmt die Farben der Pixel
fürs Diagramm
        int pic_nr;
        if(vMax == 0){
            pic_nr = 0;
        } else {
            pic_nr = (int) Math.round((v*10/vMax));
        }
        // int pic_nr = Greenfoot.getRandomNumber(vMax+1);
        switch(pic_nr){
            case 0:
            setImage("pxl0.png");
            break;
            case 1:
            setImage("pxl1.png");
            break;
            case 2:
            setImage("pxl2.png");
            break;
            case 3:
            setImage("pxl3.png");
            break;
```

```
            case 4:
            setImage("pxl4.png");
            break;
            case 5:
            setImage("pxl5.png");
            break;
            case 6:
            setImage("pxl6.png");
            break;
            case 7:
            setImage("pxl7.png");
            break;
            case 8:
            setImage("pxl8.png");
            break;
            case 9:
            setImage("pxl9.png");
            break;
            default:
            setImage("pxl9.png");
        }
    }

    public int getSpeed(){
        return v;
    }

    public void setSpeed(int j){
        v = j;
    }

    void test(){
        System.out.println(200 * 90 / 90);
    }

    public void setVMax(int n){
        vMax = n;
    }

    public boolean checkDoubles(){
        if(isTouching(Auto.class) == true){
            return true;
        } else {
            return false;
        }
    }
}
```

Klasse Zelle_Linie

```
import greenfoot.*;

public class Zelle_Line extends Actor
```

```
{

}
```

Klasse Zelle

```
import greenfoot.*;

public class Zelle extends Actor
{
    boolean gefüllt;

    public boolean getGefüllt(){
        return gefüllt;
    }

    public void setGefüllt(boolean gef){
        gefüllt = gef;
    }
}
```

Klasse Button

```
import greenfoot.*;

public class Button extends Actor
{
    public Button(){

    }

    public Button(boolean b){
        if(b == true){
            setImage("button_yes.png");
        }
        if(b == false){
             setImage("button_no.png");
        }
    }

    public Button(String s){
        if(s == "nasch"){
            setImage("button_nasch.png");
        }
        if(s == "vdr"){
            setImage("button_vdr.png");
```

```
        }
    }

    public Button(int i){
        if(i == 1){
            setImage("button_add.png");
        }
        if(i == -1){
            setImage("button_sub.png");
        }
        if(i == 10){
            setImage("button_add_10.png");
        }
        if(i == -10){
            setImage("button_sub_10.png");
        }
        if(i == 0){
            setImage("button_start.png");
        }
    }

    public void act()
    {

    }

    public boolean isClicked(){
        if(Greenfoot.mouseClicked(this)){
            return true;
        } else {
            return false;
        }
    }

    public void remove(){


    }
}
```

Klasse Pfeil

```
import greenfoot.*;

public class Pfeil extends Actor
{
    public Pfeil(){

    }

    public Pfeil(String achse){
        if(achse == "x"){
            setImage("pfeil_x_achse.png");
        }
```

```
        if(achse == "y"){
            setImage("pfeil_y_achse.png");
        }
    }
}
```

Klasse Text

```
import greenfoot.*;

public class Text extends Actor
{
    private String text;

    public Text(int i)
    {
        if(i == 0){
            setImage("verkehrsfluss_symbol.PNG");
        }
        if(i == 1){
            setImage("dichte_symbol.PNG");
        }
    }

    public Text(String t, int size){
        text = t;
        Color bgColor = new Color(0, 0, 0, 0); // transparent
background
        Color fontColor = Color.BLACK;
        int fontsize = size;

        GreenfootImage txtImg = new GreenfootImage(text, fontsize,
fontColor, bgColor);
        GreenfootImage img = new GreenfootImage (600, 80);

        img.setColor(bgColor);
        img.fill();
        img.drawImage(txtImg, 10, 5);
        setImage(img);

        // https://www.greenfoot.org/topics/4227
    }

    public Text(String t, int size, int breite){
        text = t;
        Color bgColor = new Color(0, 0, 0, 0); // transparent
background
        Color fontColor = Color.BLACK;
        int fontsize = size;

        GreenfootImage txtImg = new GreenfootImage(text, fontsize,
fontColor, bgColor);
        GreenfootImage img = new GreenfootImage (breite, 80);

        img.setColor(bgColor);
        img.fill();
```

```
        img.drawImage(txtImg, 10, 5);
        setImage(img);

        // https://www.greenfoot.org/topics/4227
    }

    public Text(String t){
        text = t;
        Color bgColor = new Color(0, 0, 0, 0); // transparent
background
        Color fontColor = Color.BLACK;
        int fontsize = 30;

        GreenfootImage txtImg = new GreenfootImage(text, fontsize,
fontColor, bgColor);
        GreenfootImage img = new GreenfootImage (50, 50);

        img.setColor(bgColor);
        img.fill();
        img.drawImage(txtImg, 10, 5);
        setImage(img);

        // https://www.greenfoot.org/topics/4227
    }

    public void setText(String t){
        text = t;
    }
}
```

Klasse Point

```
import greenfoot.*;

public class Point extends Actor
{
    public Point(){

    }
    public boolean checkDoubles(){
        if(isTouching(Point.class) == true){
            return true;
        } else {
            return false;
        }
    }
}
```

Klasse Data

```
public class Data
{
    private int runde;
    private int autosAnz;
    private int auto_nr; //nummer des letzten Autos, welches
gemessen wurde
    private int xWert; //gibt an an welchem x-Wert der nächste
Punkt gesetzt wird (für diagramm1)
    private int xWert2; //gibt an an welchem x-Wert der nächste
Punkt gesetzt wird (für diagramm 2)

    private int[] flow_value;
    private int counter;

    public Data()
    {
        runde = 0;
        autosAnz = 0;
        auto_nr = -1;
        xWert = 0;
        xWert2 = 0;

        flow_value = new int[1000];
        flow_value[1000-1] = -1;
        counter = 0;
    }

    public void setXWert2(int x){
        xWert2 = x;
    }

    public int getXWert2(){
        return xWert2;
    }

    public void setXWert(int x){
        xWert = x;
    }

    public int getXWert(){
        return xWert;
    }

    public void setRunde(int r){
        runde = r;
    }

    public int getRunde(){
        return runde;
    }

    public void setAutosAnz(int i){
        autosAnz = i;
    }

    public int getAutosAnz(){
        return autosAnz;
    }
```

```
    public void setAutoNr(int i){
        if(i == -1){
        }
        if(i == auto_nr){
        }
        if(i != auto_nr){
            autosAnz++;
        }
        auto_nr = i;
    }

    public int getAutoNr(){
        return auto_nr;
    }

    public void addFlowValue(int f){
        flow_value[counter] = f;

        counter++;
        if(counter >=1000){
            counter = 0;
        }


        int anz;
        if(flow_value[1000-1] == -1){
            anz = counter;
        } else {
            anz = 1000;
        }

        int avgFlow = 0;
        for (int i = 0; i < anz; i++){
            avgFlow = avgFlow + flow_value[i];
        }

        avgFlow = avgFlow/anz+1;
        System.out.println("" + avgFlow);
    }


}
```

Erklärung der eigenständigen Anfertigung

Ich erkläre hiermit, dass ich die Seminararbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benützt habe.

…....., den ….....

Ort Datum

….....