# Dokumentation - Erweiterte Nagel-Schreckenberg Verkehrssimulation

## Projektübersicht

Das vorliegende Projekt implementiert eine erweiterte Version des klassischen Nagel-Schreckenberg Verkehrsmodells mit einer grafischen Benutzeroberfläche. Die Simulation modelliert eine zweispurige Autobahn mit verschiedenen Fahrzeugtypen und inkludiert eine interaktive Parametersteuerung sowie eine Echtzeit-Visualisierung.

## Technische Spezifikationen

- **Programmiersprache:** Python 3.x
- **GUI-Framework:** Tkinter
- **Bildverarbeitung:** PIL (Python Imaging Library)
- **Architektur:** Objektorientierte Programmierung
- **Dateienstruktur:**
  - `stausimulation.py` - Hauptprogramm
  - `img/car.png` - PKW-Bild (normale Geschwindigkeit)
  - `img/car_slow.png` - PKW-Bild (langsame Geschwindigkeit)
  - `img/lkw.png` - LKW-Bild (normale Geschwindigkeit)
  - `img/lkw_slow.png` - LKW-Bild (langsame Geschwindigkeit)

## Kernfunktionen

### Fahrzeugmodellierung

Die Simulation implementiert zwei Fahrzeugtypen mit unterschiedlichen Eigenschaften:

- **PKW:** Länge 1 Zelle, Geschwindigkeitsbereich 60-250 km/h (konfigurierbar)
- **LKW:** Länge 2 Zellen, Geschwindigkeitsbereich 60-120 km/h (konfigurierbar)

### Spurwechsel-Algorithmus

Das Spurwechselverhalten basiert auf folgenden Prinzipien:

1. **Rechtsfahrgebot:** Alle Fahrzeuge bevorzugen die untere Spur (rechte Fahrbahn)
2. **Intelligente Überholmanöver:** Wechsel auf die linke Spur nur bei Bedarf
3. **Aktiver Rückwechsel:** Automatische Rückkehr zur rechten Spur nach Überholvorgang
4. **Sicherheitsabstände:** Kollisionsvermeidung durch Mindestabstände

## Implementierung der Vehicle-Klasse

Die zentrale Fahrzeug-Klasse implementiert die Fahrzeuglogik:

```
class Vehicle:
    def __init__(self, position, vehicle_type="car", lane=0):
        self.position = position
        self.fractional_position = 0.0  # Für flüssige Bewegung
        self.speed = 0
        self.vehicle_type = vehicle_type  # "car" oder "truck"

        # Realistische Geschwindigkeitsverteilung
        if vehicle_type == "car":
            self.max_speed = random.randint(MIN_SPEED_CAR, MAX_SPEED_CAR) *
                KMH_TO_CELLS
        else:
            self.max_speed = random.randint(
                BASE_SPEED_TRUCK - TRUCK_SPEED_VARIATION,
                BASE_SPEED_TRUCK + TRUCK_SPEED_VARIATION
            ) * KMH_TO_CELLS

        self.length = 1 if vehicle_type == "car" else 2  # Zellen belegt
        self.lane = lane  # 0 = obere Spur, 1 = untere Spur
        self.lane_change_cooldown = 0  # Verhindert häufige Spurwechsel
        self.preferred_lane = 1  # Alle Fahrzeuge bevorzugen untere Spur
```

## Nagel-Schreckenberg Algorithmus

Der Simulationsschritt implementiert den klassischen Nagel-Schreckenberg Algorithmus mit Erweiterungen:

```
def update_simulation():
    """Ein Schritt des Nagel-Schreckenberg Modells mit Spurwechsel"""
    global step_count
    step_count += 1

    # Schritt 1: Spurwechsel-Prüfung
    for vehicle in vehicles:
        other_lane = 1 - vehicle.lane

        if (can_change_lane(vehicle, other_lane) and
            get_lane_change_benefit(vehicle)):
            vehicle.lane = other_lane
            vehicle.lane_change_cooldown = 10  # Cooldown setzen

    # Schritt 2: Beschleunigung
    for vehicle in vehicles:
        vehicle.speed = min(vehicle.speed + 1, vehicle.max_speed)

    # Schritt 3: Bremsung (Kollisionsvermeidung)
    for vehicle in vehicles:
        distance = get_distance_to_next_vehicle(vehicle)
```

```
        if distance <= vehicle.speed:
            vehicle.speed = max(0, distance - 1)

    # Schritt 4: Zufälliges Bremsen
    for vehicle in vehicles:
        if random.random() < BRAKE_PROB:
            vehicle.speed = max(0, vehicle.speed - 1)

    # Schritt 5: Bewegung
    for vehicle in vehicles:
        vehicle.position = (vehicle.position + vehicle.speed) % NUM_CELLS
```

## Spurwechsel-Logik

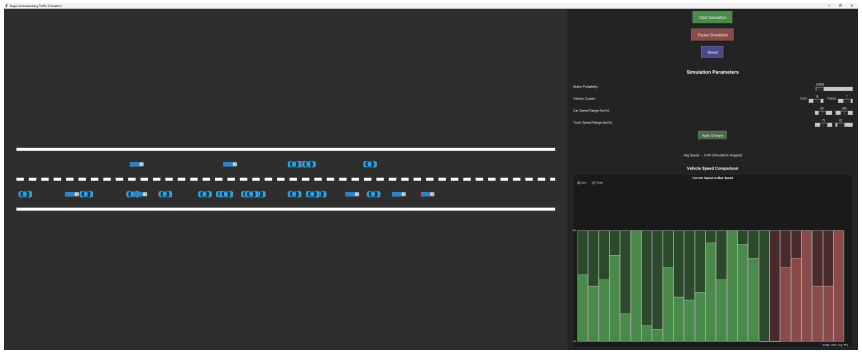
Die Spurwechsel-Entscheidung berücksichtigt mehrere Faktoren:

```
def get_lane_change_benefit(vehicle):
    """Bestimmt, ob ein Spurwechsel vorteilhaft wäre"""
    current_distance = get_distance_to_next_vehicle(vehicle)
    other_lane = 1 - vehicle.lane
    other_distance = get_distance_to_next_vehicle(vehicle, other_lane)

    # Prüfung ob Fahrzeug erheblich ausgebremst wird
    is_significantly_slowed = vehicle.speed < (vehicle.max_speed * 0.7)

    if vehicle.lane == 0 and other_lane == 1:  # Rückkehr zur bevorzugten Spur
        return other_distance > current_distance + 2
    elif vehicle.lane == 1 and other_lane == 0:  # Überholmanöver
        return (is_significantly_slowed and
                other_distance > current_distance + 8)

    return False
```

# Benutzeroberfläche

Die GUI ist in zwei Hauptbereiche unterteilt:

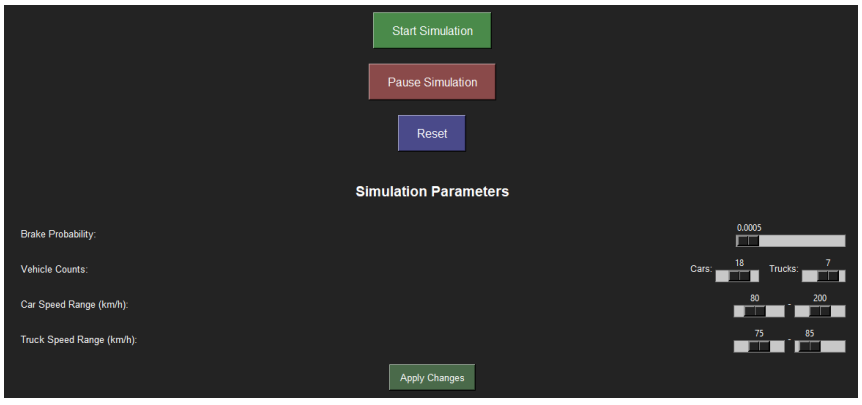
- **Simulationsbereich (66%):** Visualisierung der zweispurigen Autobahn
- **Kontrollbereich (34%):** Steuerungselemente und Statistiken




Zweispurige Autobahn-Simulation mit Kontrollpanel

## Steuerungselemente

- **Simulationskontrolle:** Start/Pause/Reset-Buttons
- **Bremswahrscheinlichkeit:** Slider von 0,0001 bis 0,05
- **Fahrzeuganzahl:** PKW (0-30), LKW (0-10)
- **Geschwindigkeitsbereiche:** Separate Einstellung für PKW und LKW
- **Parameteranwendung:** "Änderungen anwenden"-Button




Parametersteuerung mit Slidern und Buttons

# Visualisierung und Statistiken

## Echtzeit-Geschwindigkeitsdiagramm

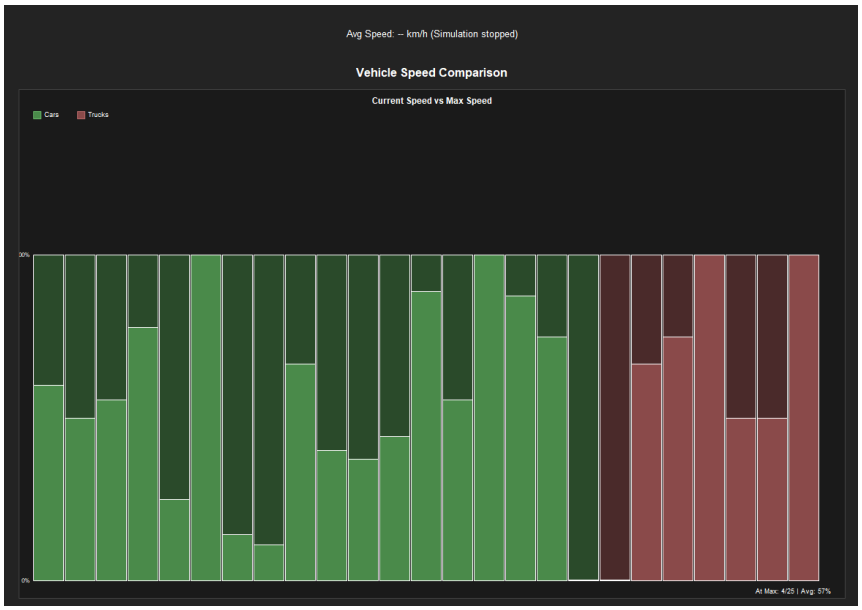
Das Programm erstellt ein Live-Diagramm aller Fahrzeuggeschwindigkeiten:

```
def draw_max_speed_diagram():
    """Zeichnet Diagramm der aktuellen vs. maximalen Geschwindigkeit"""
    diagram_canvas.delete("all")

    # Sortiere Fahrzeuge nach Typ
    cars = [v for v in vehicles if v.vehicle_type == "car"]
    trucks = [v for v in vehicles if v.vehicle_type == "truck"]

    # Zeichne Balkendiagramm
    for vehicle in cars + trucks:
        speed_percentage = (vehicle.speed / vehicle.max_speed) * 100
        bar_height = int((speed_percentage / 100) * max_bar_height)

        # Farbkodierung: Grün für PKW, Rot für LKW
        color = "#4a8a4a" if vehicle.vehicle_type == "car" else "#8a4a4a"

        # Zeichne Balken
        diagram_canvas.create_rectangle(
            x_pos, h - 25 - bar_height,
            x_pos + bar_width, h - 25,
            fill=color, outline="white"
        )
```

## Live-Statistiken

Die Anwendung berechnet und zeigt folgende Statistiken in Echtzeit:

- Durchschnittsgeschwindigkeit aller Fahrzeuge
- Separate Durchschnittsgeschwindigkeit für PKW und LKW
- Prozentuale Geschwindigkeitsverteilung im Diagramm
- Anzahl der Fahrzeuge bei Maximalgeschwindigkeit




Live-Balkendiagramm mit Geschwindigkeitsverteilung

## Simulationsparameter

- **Straßenlänge:** 150 Zellen
- **Zellengröße:** 10 Pixel
- **Aktualisierungsrate:** 50ms (20 FPS)
- **Geschwindigkeitskonvertierung:** 0,05 Zellen/Schritt pro km/h
- **Bewegungsbruchteil:** 0,15 für flüssige Animation
- **Standard-Bremswahrscheinlichkeit:** 0,0005

## Entwicklungshistorie

Das Projekt durchlief mehrere Entwicklungsstufen:

1. **Version 1.0:** Grundfunktionalität mit einspuriger Simulation
2. **Version 2.0:** Erweiterung auf zwei Spuren mit Spurwechsel
3. **Version 3.0:** Realistische Fahrzeugdynamik und Geschwindigkeitsverteilung
4. **Version 4.0:** Implementierung des Rechtsfahrgebots
5. **Version 5.0:** Interaktive Parametersteuerung
6. **Version 6.0:** Erweiterte Fahrzeugkontrolle
7. **Version 7.0:** Visualisierung und Statistiken
8. **Version 8.0:** Benutzerfreundlichkeitsverbesserungen
9. **Version 9.0:** Finale Optimierungen

# Anwendung und Bedienung

## Systemanforderungen

- Python 3.6 oder höher
- PIL/Pillow Bibliothek
- Tkinter (normalerweise in Python enthalten)
- Windows-System (optimiert für Vollbild-Darstellung)

## Bedienungsanleitung

1. Programm starten: `python stausimulation.py`
2. "Simulation starten" klicken für erste Ausführung
3. "Pausieren" für Parameteränderungen
4. Parameter über Slider anpassen
5. "Änderungen anwenden" klicken
6. "Fortsetzen" für Weiterführung der Simulation
7. "Reset" für kompletten Neustart

# Empfohlene Einstellungen

- **Bremswahrscheinlichkeit:** 0,001-0,005 für realistische Verkehrsmuster
- **PKW-Anzahl:** 15-25 für moderate Verkehrsdichte
- **LKW-Anzahl:** 3-8 für realistische LKW-Anteile
- **Geschwindigkeitsbereiche:** Standard-Werte für Autobahnverkehr

# Wissenschaftliche Erkenntnisse

Die Simulation demonstriert verschiedene Verkehrsphänomene:

- **Staubildung:** Entstehung von Verkehrsstaus durch Dichtefluktuationen
- **Spurwechselverhalten:** Einfluss des Rechtsfahrgebots auf Verkehrsfluss
- **Geschwindigkeitsverteilung:** Realistische Geschwindigkeitsunterschiede zwischen Fahrzeugtypen
- **Kapazitätseffekte:** Auswirkungen der Fahrzeugdichte auf Durchschnittsgeschwindigkeit

# Erweiterungsmöglichkeiten

Zukünftige Entwicklungen könnten folgende Aspekte umfassen:

- Dreispurige Autobahn-Simulation
- Verkehrsschilder und Geschwindigkeitsbegrenzungen
- Unterschiedliche Fahrertypen (aggressiv, defensiv)
- Wetter- und Sichtbedingungen
- Unfallsimulation und Staubildung
- Datenexport für statistische Analysen

## Fazit

Die erweiterte Nagel-Schreckenberg Verkehrssimulation stellt eine umfassende Implementierung des klassischen Modells dar. Durch die Kombination von realistischen Fahrzeugtypen, intelligentem Spurwechselverhalten und einer benutzerfreundlichen GUI eignet sich das Programm sowohl für Bildungszwecke als auch für die Untersuchung von Verkehrsmustern. Die modulare Architektur ermöglicht einfache Erweiterungen für spezifische Anwendungsfälle.

## GitHub Repository

Das Projekt ist auf GitHub verfügbar:
https://github.com/darkprince2103/AngModSys

## Quellen

- https://de.wikipedia.org/wiki/Nagel-Schreckenberg-Modell
- https://docs.python.org/3/library/tkinter.html
- Zur Implementierung wurde GitHub Copilot (Agent Mode mit Claude Sonnet 4.0 Preview) verwendet.