Simulation am Computer – Einspurige Verkehrssimulation

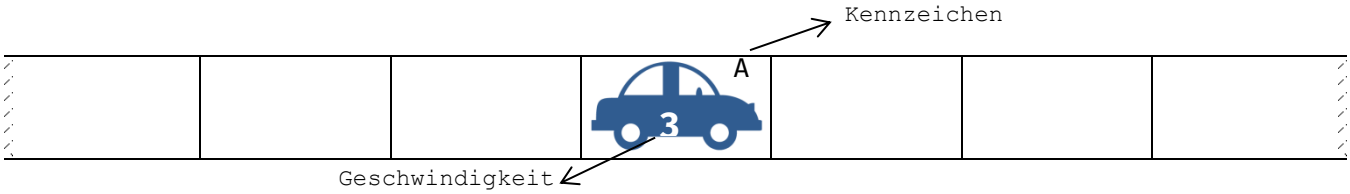
Um nicht immer viele Generationen für verschiedene Situationen per Hand auswerten zu müssen, könnt ihr die Simulation des Straßenverkehrs und den Stau aus dem Nichts am Computer realisieren.

Es stellt sich nun die Frage, was überhaupt für eine Verkehrssimulation benötigt wird. Autos natürlich! Dafür existiert bereits eine Klasse **Auto**, die eure Autos auf der Straße als Objekte definiert. Jedes Auto hat eine *Geschwindigkeit*, mit der es sich fortbewegt, und ein *Kennzeichen* (wir beschränken uns hier auf einen einzelnen Buchstaben), um es in der Simulation besser verfolgen zu können.

Damit hat die Klasse Auto zwei Attribute, auf die ihr mit folgenden Methoden zugreifen könnt:

- **public int** getGeschwindigkeit(): Liefert die aktuelle Geschwindigkeit eines Autos zurück.
- **public void** setGeschwindigkeit(**int** geschwindigkeit): Setzt die Geschwindigkeit eines Autos auf den übergebenen Wert geschwindigkeit.
- **public** String getBuchstabe(): Liefert den Buchstaben des Kennzeichens eines Autos zurück.

Im Applet findet ihr beide Attribute an folgenden Stellen wieder:



Für die eigentliche Simulation gibt es die Klasse **EinspurigeSimulation**, in der eurer Algorithmus realisiert wird. Sie enthält wie die Klasse zur Simulation der Zellvermehrung wieder zwei Attribute:

- **private** Auto[] aktuelleGeneration = **new** Auto[20]: Ein Array der Größe 20 vom Typ Auto. Es repräsentiert die Straße und speichert die Autos an den entsprechenden Positionen ab.
- **private int** troedelfaktor = 0: Speichert die Wahrscheinlichkeit von 0 bis 100, mit der Autos später trödeln. In der Erweiterung der Simulation könnt ihr diesen Trödelfaktor später einbauen.

Auch in dieser Klasse sind schon die get- und set-Methoden vorgegeben. Eure Aufgabe ist es, die Methode **public void** berechneNaechsteGeneration() zu realisieren. In dieser Methode verstecken sich wieder die Regeln, nach denen sich nun die Autos auf der Straße verhalten sollen.

Im **Package Explorer** unter *Verkehrssimulation – src – (default package)* findet ihr die Datei **EinspurigeSimulation.java**, die ihr mit einem Doppelklick öffnen könnt – und los geht's!.

Hinweis: Im Folgenden ist die Methode nach dem NaSch-Modell in die drei Schritte Beschleunigen, Bremsen und Bewegen unterteilt. Ihr könnt trotzdem alles nach einander in die Methode `berechneNaechsteGeneration()` schreiben.

Schritt 1 – Beschleunigen

Jedes Auto soll nach dem NaSch-Modell zunächst seine Geschwindigkeit um eins erhöhen, unabhängig davon, ob es später bremsen muss. Voraussetzung dafür ist natürlich, dass es nicht schon die Höchstgeschwindigkeit erreicht hat.

Mit einer geeigneten Schleife könnt ihr die Geschwindigkeit von jedem Auto im Array erhöhen. Tipps für den Umgang mit den Autos findet ihr auf folgendem Notizzettel. Schreibt euren Code in die Methode `berechneNaechsteGeneration()` unter den Kommentar `//Beschleunigen`.

Was sind *Objekte & Attribute*?

Objekte in Java repräsentieren in der Regel auch reale Objekte wie z.B. ein Auto. Dabei hat solch ein Objekt immer auch Eigenschaften. So kann ein Auto z.B. eine bestimmtes Kennzeichen haben. Diese Eigenschaften sind in der Programmierung die sogenannten *Attribute*.

Möchtet ihr überprüfen, ob sich an Position i im Array aktuelleGeneration ein Auto befindet, könnt ihr dies mit folgender if-Abfrage tun:

```
if(aktuelleGeneration[i] != null) {...}
```

Möchtet ihr auf dieses Auto an der Stelle i zugreifen, könnt ihr es zunächst unter einem beliebigen Namen abspeichern (z.B. meinAuto):

```
Auto meinAuto = aktuelleGeneration[i];
```

Über die soeben erzeugte Variable mit dem Namen meinAuto könnt ihr nun auch auf die Attribute des Autos zugreifen. Die Geschwindigkeit des Autos könnt ihr in einer Integer-Variable speichern (z.B geschwindigkeit):

```
int geschwindigkeit = meinAuto.getGeschwindigkeit();
```

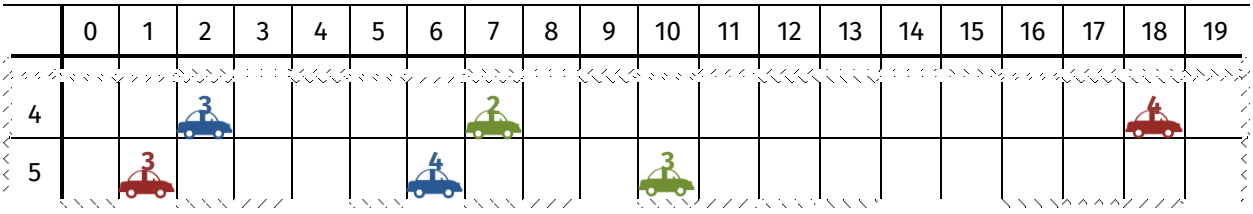
Möchtet ihr die Geschwindigkeit des Autos verändern (z.B. auf 4), geschieht dies über den Befehl:

```
meinAuto.setGeschwindigkeit(4);
```

Schritt 2 – Bremsen

Jedes Fahrzeug auf der Straße prüft nach dem Beschleunigen, ob es mit der gerade berechneten Geschwindigkeit auf ein anderes Fahrzeug auffahren oder dieses sogar überholen würde. Ist das der Fall, so reduziert es seine Geschwindigkeit sofort so weit, dass eine Kollision vermieden wird.

Doch aufgepasst! Die Straße soll einen Ring bilden. Autos, die also am „Ende“ der Straße angekommen sind, d.h. über das letzte Feld des Arrays fahren, müssen beachten, das ihnen am „Anfang“ niemand im Weg steht. Das rote Auto muss also bremsen und steht in der nächsten Generation im Array auf Position 1, da das blaue Auto an Position 2 den Weg versperrt.



Um die Position über den Rand hinaus automatisch berechnen zu lassen, gibt es die Division mit Rest – die *Modulo*-Rechnung.

Wie funktioniert die *Modulo*-Rechnung?

Die *Division mit Rest* berechnet für zwei ganze Zahlen ein ganzzahliges Ergebnis und den minimalen Rest, d.h. die Division mit Rest liefert für folgende Aufgabe das Ergebnis

23 : 5 = 4 Rest 3.

In *Java* repräsentiert das Prozentzeichen (%) die Division mit Rest und liefert als Ergebnis nur den Rest zurück, d.h. der Code

```
23 % 5
```

liefert als Ergebnis 3.

Wie würde die *konkrete* Modulo-Rechnung für das rote Auto aus der obigen Situation lauten?

(18 + _____) % _____ = 1

Welche *allgemeine* Modulo-Rechnung eignet sich also für Berechnung der neuen Position eines jeden Autos auf der Straße? Ergänzt folgende Formel (in Worten):

(_____ + _____) % _____

Ihr könnt wieder mit einer geeigneten Schleife das Array durchlaufen. Benutzt eure Formel, um die neue Position für jedes Auto in der Methode zu berechnen. Schreibt euren Code unter den Kommentar //Bremsen.

Ergänzt zudem eine Ausgabe für die Simulation, wenn ein Auto bremsen muss, um beim Testen eventuelle Fehler besser zu erkennen. Dazu steht euch die Methode

SimulationBoard.*write*(String text);

zur Verfügung. Eine mögliche Ausgabe kann wie folgt aussehen:

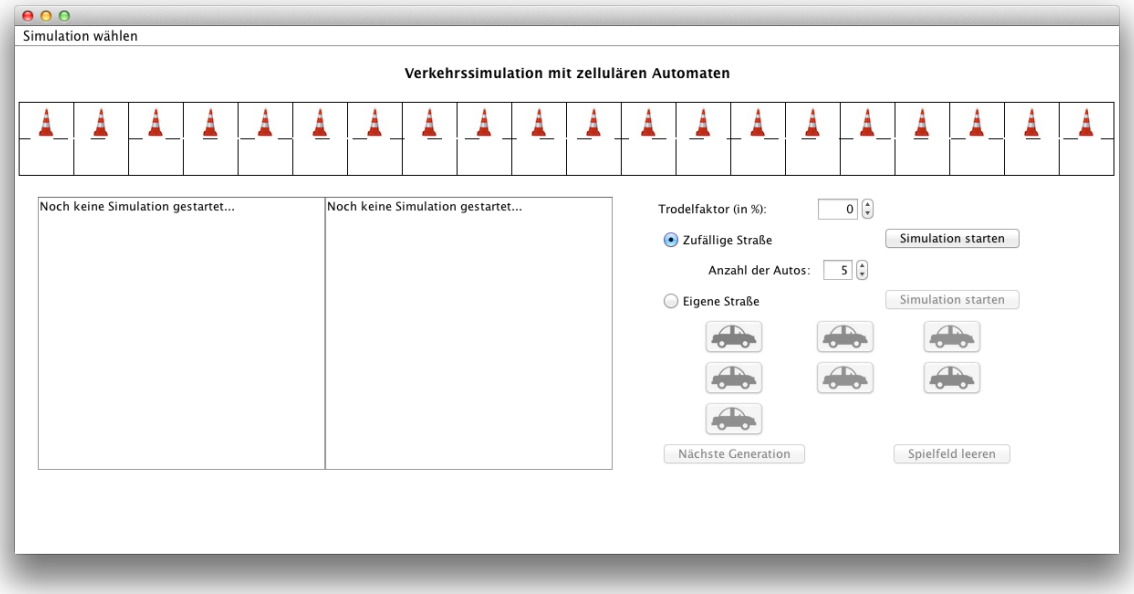
```
SimulationBoard.write("Auto" + meinAuto.getBuchstabe() +" muss bremsen!");
```

Schritt 3 – Bewegen

Nachdem nun die neue Geschwindigkeit für jedes Auto berechnet wurde, muss alle Autos nur noch um die entsprechenden Felder vorwärts bewegt werden. Aber aufgepasst! Auch hier besteht die Gefahr wie zuvor bei der Zellvermehrung, dass ihr eure Daten überschreibt, die ihr jedoch für die übrigen Autos noch benötigt. Überlegt euch wieder eine geeignete Lösung für das Problem. Schreibt euren Code unter den Kommentar `//Bewegen`.

Die Simulation testen

Startet nun die Simulation. Es öffnet sich wieder ein neues Fenster. Klick oben links in der Leiste auf *Simulation wählen* und dann *Einspurige Straße*. Nun könnt ihr eure Simulation testen.



Aufgabe 1:

Testet die Situation aus Aufgabe 2 vom vorigen Blatt und überprüft eure gelegten Ergebnisse. Stimmen eure Beobachtungen und Vermutungen?

_____

_____

Aufgabe 2:

Was passiert, wenn jedes Auto mindestens 5 Felder vor sich frei hat? Denkt euch eine geeignete Situation aus und testet sie in eurer Simulation. Testet auch Situationen mit höheren Verkehrsdichten. Ab wie vielen Autos entsteht ein Stau und wie wandert er über die Straße?

_____

_____

Aufgabe 3:

Wie realistisch ist eure Simulation? Würden sich Autofahrer im alltäglichen Straßenverkehr auch so verhalten wie eure Autos? Was würden menschliche Autofahrer eventuell anders machen?

_____

_____