Programmierpraktikum

Wintersemester 2018/2019

Prof. Dr. rer. nat. habil. Petra Hofstedt
Sven Löffler M. Sc.
Viktoria Köhler B. Sc, Sonja Breuß, Julius Schöning

Brandenburgische
Technische Universität
Cottbus - Senftenberg

Übungsblatt 8

Abgabedatum: 12.12.2018

Die Abgabe Ihrer Lösungen erfolgt vor Ablauf der Abgabefrist digital über die Moodle-Plattform an Ihren Tutor. Erstellen Sie dazu ein PDF-Dokument, das die Lösungen Ihrer schriftlichen Aufgaben enthält. Laden Sie dieses PDF-Dokument und die erzeugten Java-Klassen (.java-Datein) mit den in den Aufgaben vorgegebenen Namen, bei Moodle hoch.

Sie können maximal **(8 +1,5 Punkte)** mit diesem Übungsblatt erreichen.

Aufgabe 1 (Turtle-Grafik)

4 Punkte

Anmerkung: Für das richtige Bearbeiten der ersten 3 Aufgaben gibt es 2 Punkte und für Aufgabe 1.4 gibt es ebenfalls 2 Punkte.

Turtle-Grafik ist ein einfacher Ansatz zum Zeichnen linienbasierter Grafiken. Man stellt sich vor, dass sich eine Schildkröte (engl. turtle) über die Zeichenfläche bewegt und gegebenenfalls mit einem Stift den von ihr zurückgelegten Weg markiert. Die Schildkröte empfängt hierzu Befehle, die sie dann ausführt. Mögliche Befehle sind:

- „Setze den Stift auf die Zeichenfläche auf."
- „Nimm den Stift von der Zeichenfläche weg."
- „Bewege dich um eine bestimmte Strecke nach vorn."
- „Drehe dich um einen bestimmten Winkel gegen den Uhrzeigersinn."

Der Winkel kann auch negativ sein, dann dreht sich die Schildkröte im Uhrzeigersinn.

Offensichtlich bestimmt der aktuelle Zustand (engl. state) der Schildkröte - also ihre Position, ihre Ausrichtung und die Stellung des Stifts - den Effekt eines Befehls. Außerdem ändert ein Befehl im Allgemeinen den Zustand der Schildkröte.

1. Programmieren Sie eine Java-Klasse `TurtleState.java`, die den Zustand der Schildkröte speichert und Modifikationen des Zustands erlaubt. Führen Sie (`private`) Attribute `double x, double y` für die Position, `double angle` für die Ausrichtung und `boolean down` für die Stellung des Stifts ein. Fügen Sie für jedes Attribut Setter- und Getter- Methoden hinzu.
2. Programmieren Sie die Java-Klasse `Turtle.java`, welche die oben genannten Befehle empfängt und im Attribut `TurtleState state` entsprechende Zustandsänderungen durchführt. Gegebenenfalls zeichnet sie außerdem eine Linie in das grafische Ausgabefenster `LineFrame frame`. Verfollständigen Sie die Klasse des Weiteren um Sinnvolle JavaDoc-Kommentare.

3. Programmieren Sie ein Testprogramm für ihre Turtle-Klasse (`Wuerfel.java`), das einen Würfel mittels der Turtle-Grafik im Schrägbild zeichnet.

4. Informieren Sie sich z. B. durch den folgenden Wikipediabeitrag zum Thema Kochkurve `https://de.wikipedia.org/wiki/Koch-Kurve`.

   Schreiben Sie ein Programm `Kochkurve.java` das unter Verwendung der Turtle-Grafik eine kochsche Schneeflocke zeichnet. Dazu soll vom Nutzer zu Beginn eingegeben werden können, welche Iterationsstufe gezeichnet werden soll. Auf Stufe Null soll dabei lediglich ein gleichseitiges Dreieck mit Kantenlänge 0,7 Fenstergrößen gezeichnet werden. Für die erste Iteration wird dann jeweils jede Strecke in drei gleich große Teile geteilt und die mittlere Kante durch zwei ebenso große Strecken, die einen 60° Winkel einschließen, ersetzt. Auf jeder weiteren Iterationsstufe wird jede Strecke wieder gedrittelt und die mittlere Strecke ersetzt.

   Lassen Sie sich einen geeigneten rekursiven Ansatz zur Lösung einfallen. Beachten Sie beim Testen des Programmes, dass eine Iterationsstufe größer als zehn für den Computer sehr rechenaufwendig ist.

Aufgabe 2 (Nagel-Schreckenberg-Modell) 4 Punkte

Im Straßenverkehr kommt es gelegentlich zum mysteriösen „Stau aus dem Nichts“. Die Fahrzeuge kommen auf gerader Fahrbahn zum Stehen, obwohl es keine einleuchtende Ursache zu geben scheint: keine Hindernisse, keine Baustelle, keine Absperrung, kein Unfall, kein Unwetter – einfach nichts.

Um eine Erklärung für dieses Phänomen zu finden, beschrieben die Physiker Kai Nagel und Michael Schreckenberg das nach ihnen benannte Nagel-Schreckenberg-Modell (kurz NaSch-Modell). Es simuliert auf sehr abstrakte Weise den Straßenverkehr auf einer Fahrbahn. Die folgenden Vereinfachungen werden angenommen:

Die Fahrbahn

- ist einspurig
- hat nur eine Richtung
- wird in gleich große Zellen unterteilt

Jedes Fahrzeug

- ist gleich groß
- passt in genau eine Zelle der Fahrbahn
- kann sich nie zwischen zwei Zellen der Fahrbahn aufhalten
- fährt immer in die gleiche Richtung
- hat eine diskrete Geschwindigkeit in Zellen pro Zeiteinheit
- hat die gleiche Maximalgeschwindigkeit

Weiterhin können sich nie zwei Fahrzeuge in einer Zelle aufhalten und kein Fahrzeug kann ein anderes überholen. Die zeitliche Entwicklung des Straßenverkehrs wird durch die folgenden Update-Regeln modelliert. Diese simulieren den Übergang von einer aktuellen Verkehrssituation zur nächsten in

diskreten Zeitschritten.

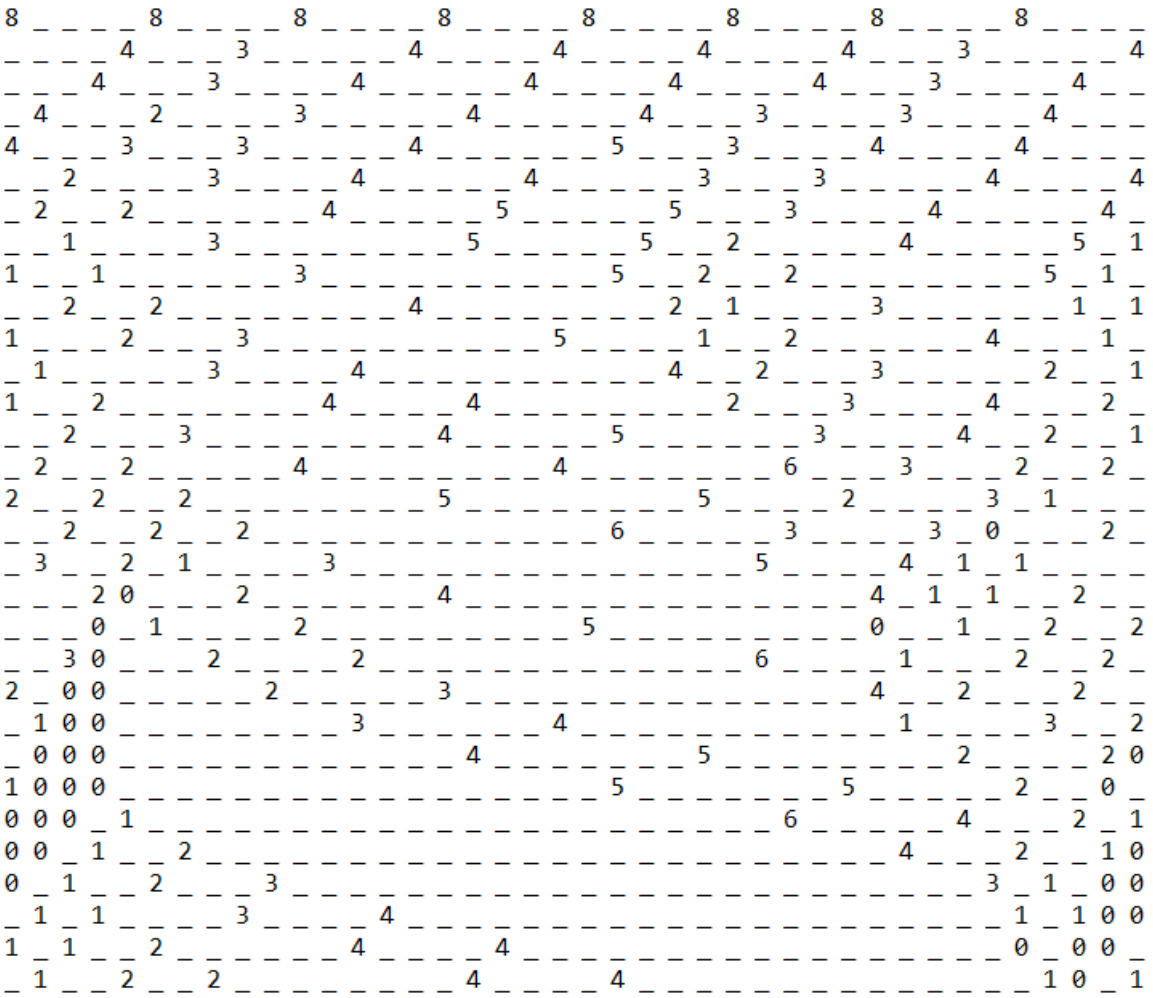
Dazu werden jeweils auf alle Fahrzeuge in der aktuellen Verkehrssituation nacheinander die folgenden Regeln angewendet:

- Erhöhe die Geschwindigkeit um eins, sofern nicht schon die Maximalgeschwindigkeit erreicht wurde.
- Falls die Geschwindigkeit größer ist als der Abstand (Anzahl freier Zellen) zum Vorderfahrzeug, verringere die Geschwindigkeit auf diesen Abstand.
- Verringere mit einer bestimmten Wahrscheinlichkeit („Trödelwahrscheinlichkeit“) die Geschwindigkeit um eins, falls das Fahrzeug nicht bereits steht.
- Setze das Fahrzeug um so viele Zellen weiter, wie seine Geschwindigkeit groß ist.

1. Implementieren Sie die drei Klassen `Auto.java`, `Fahrbahn.java` und `NaSchModel.java`. In der `main`-Methode der `NaSchModel.java` soll die `Fahrbahn` erstellt und mit Autos der Klasse `Auto.java` gefüllt werden. An den Stellen, an denen kein Auto ist, soll `null` stehen. Nehmen Sie dabei an, dass es sich bei der Fahrbahn um eine Ringstraße handelt, das heißt, die Fahrzeuge fahren im Kreis. Die Anfangssituation soll dabei so aussehen, dass alle Fahrzeuge mit möglichst gleichem Abstand hinteinander auf die Ringstraße mit Maximalgeschwindigkeit gesetzt werden. Mit dieser Ausgangssituation soll ein Stau gleich zu Beginn der Simulation vermieden werden.
2. Überschreiben Sie die `toString`-Methode der `Auto`-Klasse so, dass sie die aktuelle Geschwindigkeit des Autos als String zurückgibt. Geben Sie die Situation am Anfang und anschließend nach jedem Update aus. Die Ausgabe der Fahrbahn soll in Textform erscheinen, pro Situation (Zeiteinheit) eine Zeile. Für Fahrzeuge soll dabei die eben überschriebene `toString`-Methode genutzt werden. Die Zellen ohne Fahrzeuge (also die reine Fahrbahn) können Sie jeweils mit einem Unterstrich darstellen. Die Fahrtrichtung auf der Fahrbahn ist von links nach rechts.
3. Lassen Sie zu Beginn die folgenden Daten vom Benutzer eingeben,
   - aus wie vielen Zellen die Ringfahrbahn bestehen soll
   - wie viele Fahrzeuge auf ihr fahren sollen
   - wie groß die Maximalgeschwindigkeit der Fahrzeuge sein soll
   - wie groß die Trödelwahrscheinlichkeit (aus dem Intervall [0; 1]) sein soll
   - wie viele Updates vollzogen werden sollen.

   Simulieren sie das Nagel-Schreckenberg-Modell.

   Die Ausgabe einer Simulation mit 40 Fahrbahnzellen, 8 Autos mit Maximalgeschwindigkeit 8 und Trödelwahrscheinlichkeit 0,25 könnte bei 30 Updates zum Beispiel wie folgt aussehen.



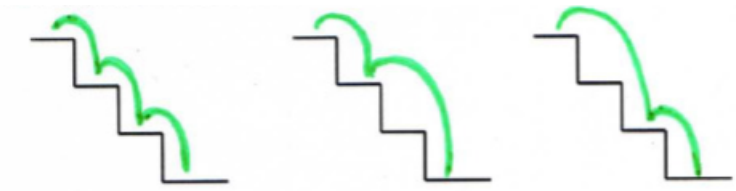
4. Halten Sie sich bei allem an die vereinbarten CodeConventions!

Aufgabe 3 (Zusatzaufgabe: Treppenproblem) (+1,5) Punkte

1. Eine Treppe kann man auf unterschiedliche Arten hinaufgehen. Gehen Sie davon aus, dass man immer die Möglichkeit hat eine Stufe oder aber zwei Stufen aufeinmal zu nehmen. Entwickeln Sie ein Programm `Treppen.java`, das als Eingabe die Höhe einer Treppe, also die Anzahl ihrer Stufen erhält und ausgibt, wie viele verschiedene Möglichkeiten es gibt die Treppe hinauf zu gehen.

   (0,5 Pkt.)

   Eine Treppe mit drei Stufen hätte z. B. die folgenden drei Möglichkeiten.



2. Schreiben Sie ein weiteres Programm `Treppen2.java` welches als Eingabe zusätzlich die ma-

ximale Schrittweite, also die Anzahl der Stufen die Maximal mit einmal überschritten werden können, einliest und die Anzahl der unterschiedlichen Möglichkeiten für das Besteigen der Treppe ausgibt.

(1 Pkt.)