

# **29. Bundeswettbewerb Informatik**

## **1. Runde**

Vincent Schüßler  
Raphael Michel

November 2010

# Inhaltsverzeichnis

<b>1</b>	<b>Junioraufgabe: Horoskope</b>	<b>3</b>
1.1	Zusammenfassung der Aufgabenstellung . . . . .	3
1.2	Lösungsidee . . . . .	3
1.3	Programmdokumentation . . . . .	3
1.3.1	Eingabeformat . . . . .	4
1.3.2	Beispieldaten . . . . .	4
1.4	Programmablaufprotokoll . . . . .	7
1.5	Programmtext . . . . .	7
1.6	Zuverlässigkeitstext . . . . .	10
<b>2</b>	<b>Aufgabe 2: Robuttons</b>	<b>11</b>
2.1	Zusammenfassung der Aufgabenstellung . . . . .	11
2.2	Lösungsidee . . . . .	11
2.3	Programmdokumentation . . . . .	11
2.4	Programmablaufprotokoll . . . . .	12
2.5	Beobachtungen . . . . .	13
2.6	Programmtext . . . . .	14
2.6.1	robuttons.html . . . . .	14
2.6.2	robuttons.js . . . . .	16
<b>3</b>	<b>Aufgabe 3: Logistisch</b>	<b>29</b>
3.1	Zusammenfassung der Aufgabenstellung . . . . .	29
3.2	Lösungsidee . . . . .	29
3.3	Programmdokumentation . . . . .	29
3.4	Programmablaufprotokoll . . . . .	30
3.5	Programmtext . . . . .	31
<b>4</b>	<b>Aufgabe 4: Drehzahl / Kartenspiel</b>	<b>35</b>
4.1	Zusammenfassung der Aufgabenstellung . . . . .	35
4.2	Lösungsidee . . . . .	35
4.3	Programmdokumentation . . . . .	35
4.3.1	kartenspiel.py . . . . .	35
4.3.2	test.py . . . . .	36
4.4	Programmablaufprotokoll . . . . .	36
4.5	Durchschnittliche Punktzahl . . . . .	37
4.6	Programmtext . . . . .	37
4.6.1	kartenspiel.py . . . . .	37
4.6.2	test.py . . . . .	41

# 1 Junioraufgabe: Horoskope

## 1.1 Zusammenfassung der Aufgabenstellung

Die Redaktion der Schülerzeitung „Meuterei“ möchte ein Programm, das für jedes der 12 Tierkreiszeichen ein zufälliges Horoskop erstellt, das nach verschiedenen vorgefertigten Mustern und mit vorgefertigten Satzbausteinen zusammengestellt wird.

## 1.2 Lösungsidee

Das Programm bekommt als Eingabedaten eine Reihe von Satzmustern und einzelnen Bausteinen, die in diese Satzmuster eingesetzt werden. Die Auswahl der Satzmuster und der einzelnen Bausteine unter den passenden wird durch Zufall vorgenommen, wobei versucht wird, den Zufallsgenerator möglichst nach einer Vorauswahl nur zwischen Satzmustern wählen zu lassen, die noch nicht in Verwendung waren.

## 1.3 Programmdokumentation

Das Programm ist in Python implementiert. Es läuft sofort unter Python 2.6, 2.7 und 3.0. Für Python 2.5 wird das Modul `simplejson`<sup>1</sup> benötigt.

Es ruft zuerst die Funktion `init()` auf, die die zu verwendenden Daten lädt. Danach wird die Funktion `generiere()` aufgerufen. In `generiere()` wird in einer verschachtelten Schleife für jedes der zwölf Tierkreiszeichen in jeder der drei vorgegebenen Kategorien eine Schleife gestartet.

In dieser Schleife wird zuerst mit der Funktion `zufallssatz()` eine Satzstruktur ausgewählt. In der Funktion `zufallssatz()` selbst wird zuerst eine Liste mit allen Satzstrukturen berechnet, die zwar in der Liste aller verfügbaren Satzstrukturen enthalten ist, nicht aber in der Liste derer, die schon verwendet wurden. Wenn diese Differenz-Liste leer ist, wird eine zufällige Satzstruktur aus der Liste aller verfügbaren ausgewählt, wenn sie nicht leer ist, eine aus der Differenz-Liste, die danach der Liste der bereits verwendeten Satzstrukturen angefügt wird.

Anschließend (wieder in der Funktion `generiere()`) wird die ausgewählte Satzstruktur auf ihre Platzhalter (siehe 1.3.1 Eingabeformat) durchsucht. Sobald einer gefunden wird, wird dieser mithilfe der Funktion `zufallding()` durch ein zufälliges Objekt ersetzt. Doppelte Verwendungen werden nach dem gleichen Verfahren wie in `zufallssatz()` vermieden.

Wenn der so entstandene Satz im gesamten bisher generierten Horoskop noch nicht vorkommt, wird der Satz ausgegeben und die Schleife wird beendet. Kommt er allerdings im Horoskop bereits einmal vor, so läuft die Schleife weiter und ein neuer Satz wird generiert. Dieser Vorgang wird maximal 50 Mal wiederholt, danach wird davon ausgegangen, dass der Datenbestand zu klein ist und es wird ein zufälliger Satz gewählt, selbst wenn dieser damit doppelt im Horoskop vorkommt.

---

<sup>1</sup><http://pypi.python.org/pypi/simplejson/2.1.1>

### 1.3.1 Eingabeformat

Als Eingabeformat für die Daten wurde das verbreitete Datenformat **JSON** gewählt. Es werden vier Elemente auf erster Ebene erwartet, einmal ein Array mit den Namen der 12 Tierkreiszeichen und Objekte für jede der drei Kategorien (Schule, Gesundheit, Liebe). Diese Objekte enthalten mindestens ein Array mit verschiedenen Satzstrukturen sowie normalerweise weitere Satzbausteine, die auf Basis der Satzstrukturen zusammengesetzt werden.

Eine Satzstruktur verwendet Platzhalter. Der Platzhalter `%d1` beispielsweise würde für einen beliebigen Wert aus dem Array `d1` des gleichen Objekts stehen, der Platzhalter `%d2` für einen beliebigen Wert aus dem Array `d2`. Ich denke, das Prinzip dürfte hiernach soweit klar sein.

Mit dieser Technik lassen sich in jede der Satzstrukturen, zum Beispiel „Dank `%d1` wirst du bald `%d2` erreichen.“, jeweils inhaltlich und grammatisch passende Satzbausteine einsetzen, beispielsweise „deiner Kreativität“ für `%d1` und „den lang ersehnten Erfolg“ für `%d2`.

### 1.3.2 Beispieldaten

In der Datei `daten.json` (weiter unten abgedruckt) finden sich Beispieldatensätze mit Satzbausteinen zur Generierung von Horoskopen. Mit diesen Datensätzen – die von der Redaktion der Schülerzeitung „Meuterei“ natürlich noch erweitert werden würden – lassen sich bereits eine große Anzahl von Horoskopen generieren, innerhalb derer keine Sätze doppelt vorkommen.

```
{
  "sternzeichen":
    [
      "Widder",
      "Stier",
      "Zwillinge",
      "Krebs",
      "Löwe",
      "Jungfrau",
      "Waage",
      "Skorpion",
      "Schütze",
      "Steinbock",
      "Wassermann",
      "Fische"
    ],
  "schule":
    {
      "saetze": [
        "Dank %d1 bereitet dir %d3 bald keine Sorgen mehr.",
        "Dank %d1 wirst du bald %d2 erreichen.",
        "Wegen %d1 wirst du bald %d2 erreichen.",
        "Aufgrund %d1 bereitet dir %d3 bald keine Sorgen mehr.",
        "%d4 bringt dir bald %d2."
      ],
      "d1" : [
        "deiner kreativen Art",
        "deiner kreativen Ader",
        "deiner Kreativität",
```

```
        "deiner kreativen Vorgehensweise",
        "deiner starken Beharrlichkeit",
        "deines bezaubernden Lächelns",
        "deines gewinnenden Lächelns",
        "deiner überragenden Intelligenz"
    ],
    "d4" : [
        "Deine kreative Art",
        "Deine kreative Ader",
        "Deine Kreativität",
        "Deine kreative Vorgehensweise",
        "Deine starke Beharrlichkeit",
        "Dein bezauberndes Lächeln",
        "Dein gewinnendes Lächeln",
        "Deine überragende Intelligenz"
    ],
    "d2": [
        "den lang ersehnten Erfolg",
        "die gewünschte Wirkung",
        "den Erfolg",
        "den ersehnten Wohlstand",
        "die ersehnte Leistung",
        "den verdienten Erfolg"
    ],
    "d3": [
        "der lang ersehnte Erfolg",
        "die gewünschte Wirkung",
        "der Erfolg",
        "der ersehnte Wohlstand",
        "die ersehnte Leistung",
        "der verdiente Erfolg"
    ]
},
"gesundheit":
{
    "saetze": [
        "Langsam wirkt sich %d1 auf deine Gesundheit aus.",
        "Denke nicht zu viel an %d2, sondern genieße %d3.",
        "Es tut dir nicht gut, dass du %d4.",
        "Du solltest %d5!"
    ],
    "d1": [
        "der Stress auch negativ",
        "die hohe Belastung auch negativ",
        "dein übertriebener Ehrgeiz"
    ],
    "d2": [
        "die Zukunft",
        "die Vergangenheit",
```

```
        "das, was morgen kommen könnte",
        "das, was gestern war"
    ],
    "d3": [
        "die Gegenwart",
        "die schönen Momente",
        "das hier und jetzt"
    ],
    "d4": [
        "immer alles auf einmal erledigen willst",
        "dir so viel Stress machst",
        "dir selbst zu wenig Zeit lässt",
        "dich selbst zu kritisch siehst",
        "immer alles aufschiebst"
    ],
    "d5": [
        "dir mehr Zeit lassen",
        "dir höhere Ziele setzen",
        "dir lieber weniger Ziele setzen und dafür versuchen, diese zu erreichen",
        "nicht so anspruchsvoll sein",
        "nicht so viel von dir selbst erwarten",
        "weniger faul sein"
    ]
},
"liebe":
{
    "saetze": [
        "Dank %d1 ist es nicht mehr weit, bis %d2.",
        "Gib acht, nicht zu %d3, sondern %d4!",
        "Wenn du %d5, wirst du merken, dass %d6."
    ],
    "d1": [
        "deiner Zuverlässigkeit",
        "deiner Ehrlichkeit",
        "deiner Offenheit",
        "deiner Vertrauenswürdigkeit",
        "deiner vielfältigen Stärken",
        "deiner Vielfältigkeit"
    ],
    "d2": [
        "du endlich glücklich wirst",
        "deine Träume wahr werden",
        "passiert, was du dir wünschst",
        "geschieht, was du schon lange ersehnt"
    ],
    "d3": [
        "viel zu träumen",
        "sehr an morgen zu denken",
        "sehr die Realität auszublenden",
```

```
        "sehr die Realität zu verdrängen"
    ],
    "d4": [
        "genieße, was du hast",
        "gib dich mit dem zufrieden, was du hast",
        "bemühe dich aktiv, dass alles besser wird",
        "sondern geh auf Menschen zu"
    ],
    "d5": [
        "aufhörst zu träumen und zu handeln beginnst",
        "endlich aktiv wirst",
        "anderen auch Zeit zum Nachdenken gibst",
        "immer ehrlich zu den Menschen bist, die dir wichtig sind"
    ],
    "d6": [
        "auch du bald erreichst, wonach du dich sehnst",
        "du in naher Zukunft glücklich wirst",
        "auch du dein Glück finden wirst",
        "sich alle deine Probleme lösen werden"
    ]
}
}
```

## 1.4 Programmablaufprotokoll

Das Horoskop-Programm wird von der Kommandozeile aus aufgerufen. Mit dem Befehl

```
$ python horoskop.py
```

wird ein vollständiges Horoskop unter Verwendung der Datei `daten.json` im gleichen Ordner generiert. Eine andere Daten-Datei kann mit dem Parameter `-data` benutzt werden:

```
$ python horoskop.py --data=dateiname.json
```

Die Ausgabe erfolgt nach folgendem Schema:

Widder:

Aufgrund deiner kreativen Vorgehensweise bereitet dir der Erfolg bald keine Sorgen mehr.

Langsam wirkt sich der Stress auch negativ auf deine Gesundheit aus.

Gib acht, nicht zu sehr die Realität auszublenden, sondern bemühe dich aktiv, dass alles besser wird!

## 1.5 Programmtext

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import json
import sys
import random
from optparse import OptionParser
```

```
kategorien = ['schule', 'gesundheit', 'liebe']
genutzte_saetze = {'schule': [], 'gesundheit': [], 'liebe': []}
genutzte_parameter = {'schule': {}, 'gesundheit': {}, 'liebe': {}}
genutzte_fertige = []
data = None

# Funktionen
def zufallssatz(kat):
    "wählt einen zufälligen Satz"
    if len(genutzte_saetze[kat]) < len(data[kat]['saetze']):
        # wenn es noch ungenutzte Sätze gibt
        choosen = random.choice(list(
            set(data[kat]['saetze']) -
            set(genutzte_saetze[kat])))
        # es wird eine Liste
        # mit der Differenz
        # zweier anderer
        # Listen erstellt, die
        # dann die ungenutzen
        # Sätze enthält. Hierraus
        # wird einer per Zufall
        # gewählt.
        genutzte_saetze[kat].append(choosen)
        return choosen
    else:
        # es gibt eh keine unbenutzen mehr => irgendeinen nehmen
        return random.choice(data[kat]['saetze'])

def zufallsding(kat, n):
    "wählt ein zufälliges Objekt aus"
    k = 'd'+str(n)
    if k not in genutzte_parameter[kat]:
        genutzte_parameter[kat][k] = []
    if len(genutzte_parameter[kat][k]) < len(data[kat][k]):
        return random.choice(list(
            set(data[kat][k]) -
            set(genutzte_parameter[kat][k])))
    else:
        return random.choice(data[kat][k])

def init(datafile):
    "Initialisiere Horoskopgenerator, Lade Daten"

    global data
    try:
        datafilepointer = open(datafile, "r")
    except:
        print("Fehler beim Öffnen der Daten-Datei")
        sys.exit(1)

    try:
        data = json.load(datafilepointer)
```

```
except:
    print("Die Datei %s enthält kein gültiges JSON." % datafile)
    sys.exit(1)

def generiere(benchmark = False):
    "generiere Horoskop"
    global genutzte_fertige, genutzte_parameter, genutzte_saetze, data

    # Variablen initialisieren
    ugly = False # bekommen wir eine elegante Lösung hin?

    # Programm
    for sternzeichen in range(0, 12):
        # Für jedes der 12 Tierkreiszeichen...

        if not benchmark:
            print("\n"+data['sternzeichen'][sternzeichen]+":")

        for kat in kategorien:
            # ...in jeder der Kategorien
            versuche = 0
            while True:
                # eine zufällige Satzstruktur, die dann geparkt wird
                satz = zufallssatz(kat)

                # Parser für die %dN Platzhalter:
                while "%" in satz:
                    letztes = ""
                    vorletztes = ""
                    pos = 0
                    for char in satz:
                        if vorletztes == "%" and letztes == "d":
                            # Objekt ("Ding") einsetzen
                            ding = zufallsding(kat, int(char))
                            links = satz[0:pos-2]
                            rechts = satz[pos+1:]
                            # Satz neu zusammenbauen:
                            satz = links + ding + rechts
                            break
                        pos += 1
                    vorletztes = letztes
                    letztes = char

                if satz not in genutzte_fertige:
                    # wenn es genau diesen Satz noch nie gab, nehmen wir ihn
                    genutzte_fertige.append(satz)
                    break
            elif versuche > 50:
                # wenn wir nach 50 Versuchen nichts finden, was es noch
```

```
        # nicht gab, geben wir uns halt mit etwas zufrieden, was
        # es schon gab.
        ugly = True
        break
    else:
        # nochmal probieren!
        versuche += 1

    if not benchmark:
        print(satz) # Satz ausgeben

    sternzeichen += 1

if benchmark:
    return ugly

if __name__ == '__main__':
    # Parsen der Kommandozeile
    parser = OptionParser()
    parser.add_option(
        "-d",
        "--data",
        dest="datafile",
        type="string",
        help="Daten-Datei",
        default="daten.json",
        metavar="FILE"
    )
    (options, args) = parser.parse_args()

    init(options.datafile)
    generiere()
```

## 1.6 Zuverlässigkeitstext

Das im Projektordner beiliegende Skript *benchmark.py* dient dazu, sehr viele verschiedene komplette Horoskope zu generieren und zu zählen, wie oft ein identischer Satz in ein und dem selbem Horoskop mehr als einmal auftaucht. In unseren Tests erhielten wir folgendes Ergebnis, dass uns die Zuverlässigkeit unseres Horoskopgenerators zeigte:

```
$ ./benchmark.py -n 10000
```

0/10000 Horoskope enthalten einen Satz doppelt.

## 2 Aufgabe 2: Robuttons

### 2.1 Zusammenfassung der Aufgabenstellung

Robuttons sind kleine runde Roboter, die knapp über der Oberfläche eines Tisches schweben. Auf dem Tisch liegen Münzen. Trifft ein Robutton auf eine Münze, nimmt er sie mithilfe eines Magneten auf und schwebt weiter. Trifft er eine Münze, obwohl er bereits eine mit sich trägt, legt er seine Münze vor der anderen ab, dreht um 180 Grad und schwebt weiter.

Treffen sich zwei Robuttons, drehen sie sich zuerst um 180 Grad, drehen sich dann aber beide unabhängig voneinander in einem zufälligen Winkel zwischen -90 und 90 Grad und schweben weiter. Trifft ein Robutton auf die Tischkante, so wird er „reflektiert“.

### 2.2 Lösungsidee

Das Programm zeichnet als Visualisierung eine Tischfläche mit zufällig angeordneten Robuttons und Münzen. Es bewegt pro Frame jeden Roboter einen Schritt weiter. Dabei prüft es jeweils auf Kollisionen mit anderen Robuttons oder Münzen. Wenn solche eintreten, werden entsprechende Aktionen berechnet und ausgeführt.

### 2.3 Programmdokumentation

Das Programm ist in JavaScript auf einer HTML-Seite implementiert. Die HTML-Datei `robuttons.html` muss in einem Webbrowser aufgerufen werden, um das Programm auszuführen. Getestet wurde in aktuellen Versionen von Firefox, Chrome/Chromium und Opera, wobei Firefox nicht die volle Geschwindigkeit erreicht. Es wird daher Opera oder Chrome/Chromium empfohlen.

Bei Klick auf den „Start“-Button wird die JavaScript-Funktion `start()` aufgerufen. Diese liest die Einstellungen aus und initialisiert das Tisch-Objekt, eine Instanz der Klasse `RectTable` oder `CircleTable`. Danach ruft sie die Funktion `makeObjects()` auf. Diese generiert zuerst alle Robuttons (`new Robutton()`) an zufälligen Positionen und danach alle Münzen (`new Coin()`). Wenn die Funktion `makeObjects()` es nach einer einstellbaren Zeit nicht geschafft hat, die Robuttons und Münzen so zu verteilen, dass sie sich nicht überschneiden, so wird die Funktion `failed()` aufgerufen, die eine Fehlermeldung ausgibt, und der Vorgang wird abgebrochen.

Nachdem die Objekte erstellt worden sind, startet `start()` ein Intervall, das in regelmäßigen Abständen (je nach eingestellten Frames pro Sekunde) die Funktion `tick()` aufruft. Diese „zeichnet“ dann die Simulation. Dazu zeichnet sie erst mit `drawBackground()` den Hintergrund neu, auf den dann mit `table.draw()` der Tisch gezeichnet wird. Anschließend wird für jeden Robutton die Funktion `moveRobutton()` aufgerufen. Diese berechnet die neue Position des Robuttons, wobei sie Kollisionen erkennt und die Reaktionen darauf ausführt. Danach werden mit den Funktionen `drawCoins()` und `drawRobuttons()` die Objekte auf die Tischfläche gezeichnet.

## 2.4 Programmablaufprotokoll

Im oberen Teil der HTML-Seite befinden sich zahlreiche Konfigurationsmöglichkeiten.

**Tisch:** Es kann ein kreisförmiger und ein rechteckigen Tisch ausgewählt werden. Bei beiden kann jeweils über den Radius bzw. über Höhe und Breite die Größe eingestellt werden.

**Objekte:** In diesem Abschnitt kann eingestellt werden, wie viele Robuttons und wie viele Münzen simuliert werden sollen.

**Anzeige:** Es ist einstellbar, wie viele Schritte bzw. Bilder in einer Sekunde ausgeführt werden sollen. Je nach obigen Einstellungen und je nach Browser sowie Computerleistung gibt es hier eine Grenze, bis zu der noch genug Bilder pro Sekunde berechnet und gezeichnet werden können. Ebenso kann der Radius der Robuttons und Münzen beliebig eingestellt werden.

**Sonstiges:** Es kann das Timeout konfiguriert werden. Dieses bestimmt, wie lange die Funktion *makeObjects()* sich Zeit lässt, bevor sie abbricht, wenn sie es nicht schafft, die Robuttons und Münzen auf dem Tisch zu verteilen.

Nach einem Klick auf den Button „Start“ wird die Simulation gestartet. Sie kann mit „Pause“ pausiert oder mit „Stopp“ wieder beendet werden. Ein Bildschirmfoto ist auf Seite 13 abgebildet.

**Tisch**

Form:

Rechteck der Breite  und der Höhe

Kreis mit dem Radius

**Objekte**

Robottuns:

Münzen:

**Anzeige**

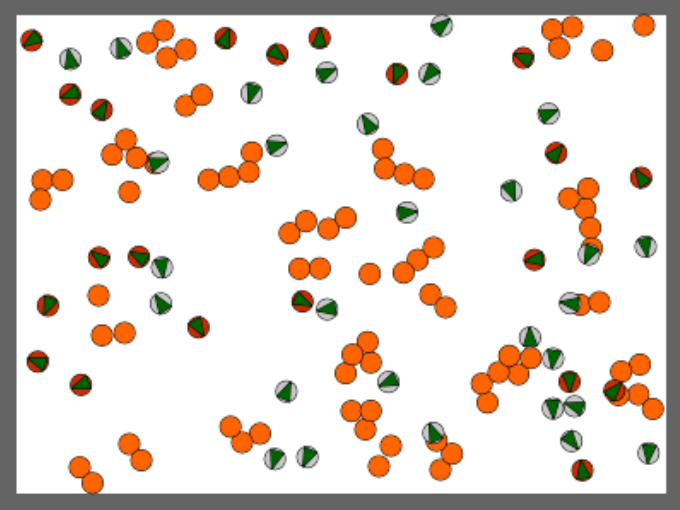
Bilder pro Sekunde:

Radius der Objekte:

**Sonstiges**

Timeout:

Wenn das Programm es nach dieser Zeitspanne (in Millisekunden) nicht geschafft hat, die Münzen und Robottuns auf dem Feld zu verteilen, wird abgebrochen.



Frame 496 FPS 72

Abbildung 2.1: Screenshot der Programmoberfläche

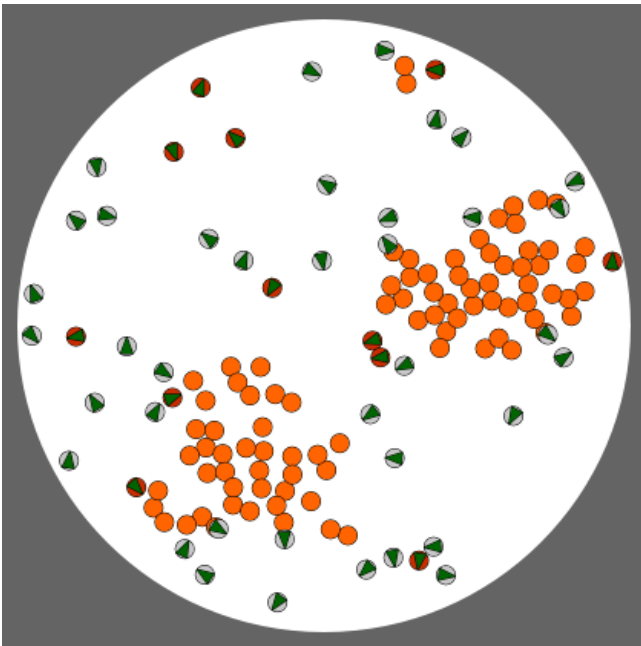
## 2.5 Beobachtungen

Wenn auf dem Feld doppelt so viele Robottuns wie Münzen vorhanden sind, hängen nach kurzer Zeit sämtliche Münzen an den Magneten von Robottuns und die Situation verändert sich dementsprechend nicht weiter. Je größer allerdings das Feld ist, desto wahrscheinlicher bilden sich ein, selten auch zwei, Haufen, die trotzdem weiter bestehen bleiben, weil jeder Robottun, der eine der Münzen aufnimmt, sofort an der dahinterliegenden wieder abprallt und der Haufen somit lange bestehen bleibt.

Platziert man gleich viele Münzen wie Robottuns auf dem Feld, so zeigt sich nach einiger Zeit eine Veränderung der Situation. Die Münzen, die vorher zufällig auf dem Feld verteilt waren, liegen jetzt in dichteren Gruppen nah beieinander. Die Anzahl der Gruppen variiert natürlich je nach Anzahl der Robottuns und Anzahl der Münzen pro Tischfläche. Wir beobachteten meist vier bis fünf größere Haufen sowie zwei bis drei kleinere, die sich nach längerer Zeit wahrscheinlich noch weiter aufgelöst und an die großen angelagert hätten.

Nimmt doppelt so viele Münzen wie Robottuns tritt der selbe Effekt ein, nur dass sich die

Situation deutlich langsamer verändert. Nach der gleichen Zeitspanne wie bei den Versuchen mit der gleichen Anzahl von Münzen und Robuttons (1500 Frames) konnten wir anstatt der wenigen großen und ganz wenigen kleinen Häufungen sieben oder mehr eher kleinere Häufungen von Münzen sehen. Nach sehr viel längerer Zeit verändert sich aber auch hier die Situation hin zu wenigen Haufen, die sich über eine sehr große Fläche erstrecken. In einem Test (kreisförmiger Tisch mit dem Radius 200 Pixel, 50 Robuttons, 100 Münzen) zeigte sich nach XXX Frames folgende typische Situation:



## 2.6 Programmtext

### 2.6.1 robuttons.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Robuttons</title>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <style type="text/css">
      p, fieldset, legend, input {
        font-family: sans-serif;
      }
      fieldset input {
        border: 1px solid #ccc;
        border-radius: 3px;
        padding: 1px 3px;
        width: 50px;
      }
      fieldset {
        border: 1px solid #ccc;
        width: 500px;
        padding: 0 10px 5px 10px;
      }
    </style>
  </head>
  <body>
    <div style="text-align: center; padding: 20px 0 0 0;">
      <h1>Robuttons</h1>
      <div style="display: flex; justify-content: space-around; align-items: center; margin-top: 10px;">
        <div style="text-align: center; width: 40%;">
          <input type="text" value="100" style="width: 80%;"/>
          <br/>
          <input type="text" value="50" style="width: 80%;"/>
        </div>
        <div style="text-align: center; width: 40%;">
          <input type="text" value="1000" style="width: 80%;"/>
          <br/>
          <input type="text" value="1000" style="width: 80%;"/>
        </div>
      </div>
      <div style="margin-top: 20px; text-align: center;">
        <input type="button" value="Start Simulation" style="width: 100px; height: 30px; border: none; background-color: #ccc; border-radius: 15px; cursor: pointer; opacity: 0.5;"/>
      </div>
    </div>
  </body>
</html>
```

```
        border-radius: 3px;
        margin-bottom: 5px;
    }
    input.radio {
        width: 20px;
    }
    fieldset legend {
        padding: 0 5px 0 5px;
        font-weight: bold;
        background: #fff;
    }
    #framenr {
        font-family: monospace;
        font-size: 12px;
    }
</style>
</head>
<body>
    <p>
        <fieldset>
            <legend>Tisch</legend>
            Form:<br />
            <input type="radio" class="radio" name="field"
                checked="checked" value="rect" id="rect" />
            <label for="rect">Rechteck</label> der Breite
            <input type="text" name="width" value="400" id="width" />
            und der Höhe
            <input type="text" name="height" value="300" id="height" />
            <br />
            <input type="radio" class="radio" name="field"
                value="circle" id="circle" />
            <label for="circle">Kreis</label> mit dem Radius
            <input type="text" name="cradius" value="200" id="cradius" />
        </fieldset>
        <fieldset>
            <legend>Objekte</legend>
            <label for="robbottuns">Robottuns:</label>
            <input type="text" name="robbottuns" value="30"
                id="robbottuns" /><br />
            <label for="coins">Münzen:</label>
            <input type="text" name="coins" value="50" id="coins" />
        </fieldset>
        <fieldset>
            <legend>Anzeige</legend>
            <label for="fps">Bilder pro Sekunde:</label>
            <input type="text" name="fps" value="30" id="fps" /><br />
            <label for="radius">Radius der Objekte:</label>
            <input type="text" name="radius" value="6" id="radius" />
        </fieldset>
    </p>
</body>
</html>
```

```
<fieldset>
  <legend>Sonstiges</legend>
  <label for="timeout">Timeout:</label>
  <input type="text" name="timeout"
        value="5000" id="timeout" /><br />
  <small>Wenn das Programm es nach dieser Zeitspanne
  (in Millisekunden) nicht geschafft hat, die Münzen und
  Robuttons auf dem Feld zu verteilen, wird
  abgebrochen.</small>
</fieldset>
<input type="button" id="s" value="Start" onclick="start()" />
<input type="button" id="p" value="Pause"
        onclick="pause()" disabled="disabled" />
</p>
<canvas id="c"></canvas>
<p id="rahmenr"></p>
<script type="text/javascript" src="robuttons.js"></script>
</body>
</html>
```

## 2.6.2 robuttons.js

```
// Setzen/Deklarieren der nötigen Variablen
var canv = document.getElementById("c");
var context = canv.getContext("2d");
var interval;
var table;
var button = document.getElementById("s");
var pausebutton = document.getElementById("p");
var robuttons;
var coins;
var frame;
var lastFPSUpdate;
var fps_real;
var framesSinceUpdate;

// Initiale Einstellungen der Konfigurationsparameter
var FPS = 30;
var WIDTH = canv.width = 400;
var HEIGHT = canv.height = 300;
var RADIUS = 6;
var NUM_ROBUTTONS = 30;
var NUM_COINS = 50;
var TIMEOUT = 5000;

function start() {
  // Übernahme der Konfigurationsparameter aus dem Formular
  //
```

```
// Auf eine Prüfung, ob ein Nutzer etwas anderes als eine
// Zahl eingibt, wird verzichtet. Sollte er sich entscheiden,
// solches zu tun, wird er mit entsprechenden Ergebnissen
// leben müssen.
FPS = parseInt(document.getElementById('fps').value);
WIDTH = parseInt(document.getElementById('width').value);
HEIGHT = parseInt(document.getElementById('height').value);
RADIUS = parseInt(document.getElementById('radius').value);
CRADIUS = parseInt(document.getElementById('cradius').value);
NUM_ROBUTTONS = parseInt(document.getElementById('robuttons').value);
NUM_COINS = parseInt(document.getElementById('coins').value);
TIMEOUT = parseInt(document.getElementById('timeout').value);

// Übernahme der Werte in das Canvas
canv.width = WIDTH
canv.height = HEIGHT

// Auswahl der Tischform
if(document.getElementById('rect').checked)
    table = new RectTable(10,10,WIDTH-20, HEIGHT-20);
else if(document.getElementById('circle').checked){
    table = new CircleTable(10,10,CRADIUS);
    HEIGHT = canv.height = CRADIUS*2+20
    WIDTH = canv.width = CRADIUS*2+20
}else
    alert('Ungültige Tischform.');
```

```
// Aufruf der Funktionen, die die Simulation durchführen
coins = [];
robuttons = [];
if(makeObjects()) {
    frame = 0;
    fps_real = 0;
    framesSinceUpdate = 0;
    lastFPSUpdate = (new Date()).getTime();
    // Austauschen des Start-/Stop-Buttons
    // und Aktivierung des Pause-Buttons
    button.value = "Stop";
    button.onclick = stop;
    pausebutton.disabled = false;
    pausebutton.value = "Pause";
    pausebutton.onclick = pause;
    interval = setInterval(tick, 1000/FPS);
}
}
```

```
function randomPoint() {
    // gibt einen beliebigen Punkt auf dem Tisch zurück
    x = Math.round(Math.random() * (table.width-1));
```

```
    y = Math.round(Math.random() * (table.height-1));
    return [x,y];
}

function randomDegAngle() {
    // gibt einen beliebigen Winkel in Grad zurück
    return Math.round(Math.random() * 359);
}

function makeObjects() {
    var starttime = (new Date()).getTime();

    // Generieren der Robuttons an zufälligen Punkten
    for(var i=0; i<NUM_ROBUTTONS; i++) {
        point = []
        do { // Es werden so lange zufällige Punkte probiert,
            // bis einer gefunden wird, an dem ein Robutton
            // abgesetzt werden kann, ohne dass dadurch eine
            // Kollision verursacht wird.
            point = randomPoint();
            no_collission = (collisionWithRobutton(point, null) == null);

            if(((new Date()).getTime() - starttime > TIMEOUT)) {
                // Dauert dieser Prozess länger als X Sekunden,
                // also kann länger als fünf Sekunden keine mögliche
                // Position ermittelt werden, wird sicherheitshalber
                // mal abgebrochen.
                failed();
                return false;
            }
        } while(table.edgeCollission(point) || !no_collission);

        // an der als frei bekannten Stelle wird der neue Roboter erzeugt
        robuttons.push(new Robutton(point, randomDegAngle()));
    }

    // Generieren der Münzen an zufälligen Punkten
    // Vorgehen siehe oben
    for(var i=0; i<NUM_COINS; i++) {
        do {
            point = randomPoint();
            no_collission = (collisionWithRobutton(point, null) == null)
                && (collisionWithCoin(point, null) == null);

            if(((new Date()).getTime() - starttime > TIMEOUT)) {
                failed();
                return false;
            }
        } while(table.edgeCollission(point) || !no_collission);
    }
}
```

```
        coins.push(new Coin(point));
    }
    return true;
}

function failed() {
    // Notbremse
    alert("Fehler! Nach der bei 'Timeout' angegebenen Zeit konnte es nicht "
        +"geschafft werden, alle Robuttons und Münzen auf dem Feld zu "
        +"verteilen.\n Zur Behebung dieses Problems sollten weniger "
        +" Robuttons und Münzen oder ein höherer Timeout-Wert gewählt werden."
        +"\nAutomatischer Abbruch...");
    stop();
}

function stop() {
    // Beendet die Ausführung der Simulation
    clearInterval(interval);
    table.draw();
    frame = 0;
    drawFrameNumber();
    // Austauschen der Buttons
    button.value = "Start";
    button.onclick = start;
    pausebutton.disabled = true;
    pausebutton.value = "Pause";
    pausebutton.onclick = pause;
}

function pause() {
    // Pausiert die Simulation
    clearInterval(interval);
    pausebutton.value = "Fortfahren";
    pausebutton.onclick = cont;
}

function cont() {
    // Setzt die Simulation nach einer Pause fort
    interval = setInterval(tick, 1000/FPS);
    pausebutton.value = "Pause";
    pausebutton.onclick = pause;
}

function tick() {
    // Führt einen Schritt in der Simulation aus und aktualisiert die
    // Darstellung

    // Darstellung zurücksetzen
    drawBackground();
}
```

```
table.draw();

updateFrameNumber();

// Aktualisiere Position etc. der Robuttons und Münzen
for(var i=0;i<robuttons.length;i++) {
    moveRobutton(robuttons[i]);
}

// Zeichne Münzen
drawCoins();

// Zeichne Robuttons
drawRobuttons();
}

function moveRobutton(robutton) {
    // Berechne neue Position des Robutton
    new_pos = robutton.getPosAfterMove();

    // Schwebt der Robutton nicht mehr über der zuletzt abgelegten Münze?
    if(robutton.lastCoin != null
        && !objectsCollide(robutton.pos, robutton.lastCoin.pos)
    ) {
        robutton.lastCoin = null;
    }

    collisionWithCoin(new_pos);
    coin_coll = collisionWithCoin(new_pos, robutton.lastCoin);
    if(coin_coll != null) {
        // Kollision mit Münze
        if(robutton.loaded) {
            // ist der Robutton nicht mehr über der zuletzt abgelegt Münze?
            if(robutton.lastCoin == null) {
                // Münze ablegen
                robutton.lastCoin = robutton.coin;
                robutton.loaded = false;
                robutton.coin.pickedUp = false;
                robutton.coin.pos = new_pos;
                robutton.coin = null;
            }
            // umdrehen
            robutton.setRadAngle(
                robutton.angle + Math.PI
            );
        }
        else {
            // Münze aufnehmen
            robutton.loaded = true;
        }
    }
}
```

```
        coin_coll.pickedUp = true;
        robutton.coin = coin_coll;
    }
}

robutton_coll = collisionWithRobutton(new_pos, robutton);
if(robutton_coll != null) {
    // Kollision mit Robutton
    // Neue Winkel für beide setzen
    robutton.setRadAngle(
        robutton.angle + (Math.PI/2)
        + (Math.random() * Math.PI)
    );
    robutton_coll.setRadAngle(
        robutton_coll.angle + (Math.PI/2)
        + (Math.random() * Math.PI)
    );
    new_pos = robutton.pos;
}

table_coll = table.edgeCollision(new_pos);
if(table_coll != null) {
    // Kollision mit Tisch
    robutton.setRadAngle(
        table.reflectionAngle(table_coll, robutton.angle)
    );
    new_pos = robutton.pos;
}

robutton.pos = new_pos;
}

function collisionWithCoin(pos, lastCoin) {
    // prüft auf Kollisionen eines Objekts an Position pos mit Münzen
    // lastCoin ist die Münze, die der Robutton (wenn das Objekt einer
    // ist) zuletzt abgelegt hat (damit er, wenn er von ihr herunterfährt,
    // sie nicht gleich wieder aufnimmt).
    coll = null;
    for(var i=0;i<coins.length;i++) {
        if(coins[i].pickedUp) continue;
        if(coins[i] == lastCoin) continue;
        if(objectsCollide(pos, coins[i].pos)) {
            coll = coins[i];
            break;
        }
    }
}
return coll;
}
```

```
function collisionWithRobutton(pos, robutton) {
    // prüft auf Kollisionen mit anderen Robotern, wenn man robutton an
    // Stelle pos setzt.
    coll = null;
    for(var i=0;i<robuttons.length;i++) {
        if(robuttons[i] == robutton) continue;
        if(objectsCollide(pos, robuttons[i].pos)) {
            coll = robuttons[i];
            break;
        }
    }
    return coll;
}

function drawCoins() {
    // Zeichnet alle Münzen auf den Tisch
    for(var i=0;i<coins.length;i++) {
        if(!coins[i].pickedUp) {
            coins[i].draw();
        }
    }
}

function drawRobuttons() {
    // Zeichnet alle Robuttons auf den Tisch
    for(var i=0;i<robuttons.length;i++) {
        robuttons[i].draw();
    }
}

function drawBackground() {
    context.fillStyle = "rgb(100, 100, 100)";
    context.fillRect(0, 0, WIDTH, HEIGHT);
}

function updateFrameNumber() {
    frame++;
    framesSinceUpdate++;
    drawFrameNumber();
    time = (new Date()).getTime();
    if((time - lastFPSUpdate) > 1000) {
        lastFPSUpdate = time;
        fps_real = framesSinceUpdate;
        framesSinceUpdate = 0;
    }
}

function drawFrameNumber() {
    document.getElementById('framenr').innerHTML =
```

```
        'Frame '+frame+" FPS " + fps_real;
    }

function objectsCollide(point1, point2) {
    // Prüft, ob zwei Objekte, die sich an den Punkten
    // point1 und point2 befinden, miteinander kollidieren
    // (unter Beachtung ihres Radius).
    space = distance(point1,point2);
    if( space <= RADIUS * 2 ) {
        return true;
    }
    else {
        return false;
    }
}

function distance(point1, point2) {
    // Abstand zwischen 2 Punkten
    dist_x = Math.abs(point2[0] - point1[0]);
    dist_y = Math.abs(point2[1] - point1[1]);
    dist = Math.sqrt(dist_x*dist_x + dist_y * dist_y);
    return dist;
}

function slope(left, right) {
    // Berechnet die Steigung zwischen dem Punkt left und dem Punkt right
    return (right[1]-left[1])/(right[0]-left[0]);
}

function rotate(ref_point, point, phi) {
    // Bewegt den Punkt point auf einem Kreis um ref_point durch point um den
    // Winkel phi im Uhrzeigersinn und gibt diesen neuen Punkt zurück
    x = point[0] - ref_point[0];
    y = point[1] - ref_point[1];
    new_x = x * Math.cos(phi) + y * Math.sin(phi) + ref_point[0];
    new_y = - x * Math.sin(phi) + y * Math.cos(phi) + ref_point[1];
    return [new_x,new_y];
}

function degToRad(degree) {
    // Konvertierung von Grad zu Radiant
    return degree/180*Math.PI;
}

function radToDeg(radian) {
    // Konvertierung von Radiant zu Grad
    return radian*180/Math.PI;
}
```

```
function Robutton(point, deg_angle) {
    this.pos = point;
    this.setDegAngle(deg_angle);
    this.loaded = false;
    this.coin = null;
    this.ignored = [];
    this.lastCoin = null;
}

Robutton.prototype.draw = function () {
    if(this.loaded) {
        context.fillStyle = "rgb(200,45,0)";
    }
    else {
        context.fillStyle = "rgb(200,200,200)";
    }
    // Zeichne den Kreis
    context.beginPath();
    context.arc(this.pos[0], this.pos[1], RADIUS, 0, 2*Math.PI, true);
    context.stroke();
    context.fill();
    context.closePath();

    // Berechne die Koordinaten des Dreiecks
    var phi = degToRad(40);
    var tri_a = [];
    var tri_b = [];
    var tri_c = [];
    tri_a[0] = this.pos[0];
    tri_a[1] = this.pos[1] + RADIUS;
    tri_b[0] = this.pos[0] + (Math.sin(phi) * RADIUS);
    tri_b[1] = this.pos[1] - (Math.cos(phi) * RADIUS);
    tri_c[0] = this.pos[0] - (Math.sin(phi) * RADIUS);
    tri_c[1] = tri_b[1];

    // Drehe das Dreieck in die Richtung des Robuttons
    tri_a = rotate(this.pos, tri_a, this.angle);
    tri_b = rotate(this.pos, tri_b, this.angle);
    tri_c = rotate(this.pos, tri_c, this.angle);

    // Zeichne das Dreieck
    context.fillStyle = "rgb(0,100,0)";
    context.beginPath();
    context.moveTo(tri_a[0], tri_a[1]);
    context.lineTo(tri_b[0], tri_b[1]);
    context.lineTo(tri_c[0], tri_c[1]);
    context.lineTo(tri_a[0], tri_a[1]);
    context.stroke();
    context.fill();
}
```

```
    context.closePath();
}

Robutton.prototype.getPosAfterMove = function() {
    // Berechnet die neue Position des Robuttons nach einem Schritt in die
    // aktuelle Richtung
    x = Math.sin(this.angle) + this.pos[0];
    y = Math.cos(this.angle) + this.pos[1];
    return [x,y];
}

Robutton.prototype.setDegAngle = function(deg) {
    // Wandelt den Winkel vor dem setzen in das Bogenmaß um
    this.angle = degToRad((deg%360));
}

Robutton.prototype.getDegAngle = function() {
    // Gibt den Winkel im Gradmaß zurück
    return radToDeg(this.angle);
}

Robutton.prototype.setRadAngle = function(rad) {
    this.angle = rad%(Math.PI*2);
}

function Coin(point) {
    this.pos = point;
    this.pickedUp = false;
}

Coin.prototype.draw = function() {
    // aufgenommene Münzen sind unsichtbar
    if(this.pickedUp) return;
    // ansonsten Zeichne Münze
    context.fillStyle = "rgb(255,100,0)";
    context.beginPath();
    context.arc(this.pos[0], this.pos[1], RADIUS, 0, 2*Math.PI, true);
    context.stroke();
    context.fill();
    context.closePath();
}

// "Mutterklasse" für Tisch-Typen
function Table(x, y, width, height) {
    this.width = width;
    this.height = height;
    this.x = x;
    this.y = y;
}
```

```
Table.prototype.edgeCollission = function (point) {}
Table.prototype.draw = function () {}
Table.prototype.reflectionAngle = function (coll_type, angle) {}

RectTransform.prototype = new Table;
RectTransform.prototype.constructor = RectTransform;

// rechteckiger Tisch
function RectTransform(x, y, width, height) {
    Table.call(this, x, y, width, height);
}

RectTransform.prototype.edgeCollission = function (point) {
    // Kollidiert das Objekt am Punkt point mit den Kanten des Tisches?
    if(
        ((point[0]-RADIUS) < this.x)
        || ((point[0]+RADIUS) >= (this.width + this.x))
    ) {
        return 2;
    }
    else if(
        ((point[1]-RADIUS) < this.y)
        || ((point[1]+RADIUS) >= (this.height + this.y))
    ) {
        return 1;
    }
    else {
        return null;
    }
}

RectTransform.prototype.draw = function () {
    context.fillStyle = "rgb(255, 255, 255)";
    context.fillRect(this.x, this.y, this.width, this.height);
}

RectTransform.prototype.reflectionAngle = function (coll_type, angle) {
    // Berechnet den Ausfallswinkel nach einer Kollission mit dem Tisch

    if(coll_type == 1) {
        return Math.PI - angle;
    }
    else if(coll_type == 2) {
        return (Math.PI*2) - angle;
    }
}

CircleTable.prototype = new Table;
```

```
CircleTable.prototype.constructor = CircleTable;

// runder Tisch
function CircleTable(x, y, radius) {
    Table.call(this, x, y, radius*2, radius*2);
    this.radius = radius;
    this.x_middle = this.x+this.radius;
    this.y_middle = this.y+this.radius;
}

CircleTable.prototype.edgeCollission = function (point) {
    // Befindet sich das Objekt am Punkt point noch auf dem Tisch?

    // Das heißt: Das Objekt darf nicht weiter vom Mittelpunkt entfernt sein
    // als der Radius - der Radius des Objekts
    distFromMiddle = distance(point, [this.x_middle, this.y_middle]);
    if( distFromMiddle + RADIUS >= this.radius ) {
        return point;
    }
    else {
        return null;
    }
}

CircleTable.prototype.draw = function () {
    // Zeichne den Tisch
    context.fillStyle = "rgb(255, 255, 255)";
    context.beginPath();
    context.arc(this.x_middle, this.y_middle, this.radius, 0, 2*Math.PI, true);
    context.fill();
    context.closePath();
}

CircleTable.prototype.reflectionAngle = function (point, angle) {
    // liefert den Ausfallswinkel nach einer Kollission mit dem Tisch
    pointOnCircle = [];
    // Berechne den Kollissionspunkt auf dem Einheitskreis

    if(point[0] == this.x_middle) {
        // Wenn der Punkt in x-Richtung mit dem Mittelpunkt übereinstimmt, ist
        // die Lage auf dem Einheitskreis bekannt
        pointOnCircle[0] = 0;
        if(point[1] > 0) {
            pointOnCircle[1] = 1;
        }
        else {
            pointOnCircle[1] = -1;
        }
    }
}
```

```
else {
    // Steigung der Geraden durch die Mittelpunkte der Kreise
    slopeToMiddlePoint = slope([this.x_middle, this.y_middle], point);
    // Winkel dieser Steigung
    arcToMiddlePoint = Math.atan(slopeToMiddlePoint) + Math.PI;
    // aus dem Winkel ergeben sich die Koordinaten auf dem Einheitskreis
    pointOnCircle[0] = Math.cos(arcToMiddlePoint);
    pointOnCircle[1] = Math.sin(arcToMiddlePoint);
    if(point[0] > this.x_middle) {
        pointOnCircle[0] *= -1;
    }
    else {
        pointOnCircle[1] *= -1;
    }
}
tanSlope = tanSlopeInPoint(pointOnCircle);
// berechne die Steigung der Normalen
normSlope = - 1/tanSlope;
// der Winkel wird an der Normalen gespiegelt
angle += 2*Math.PI + 2*Math.atan(normSlope) - 2*angle;

return angle;
}

function tanSlopeInPoint(point) {
    // Steigung einer Kreistangenten im Punkt point (auf dem Einheitskreis)
    quot1 = Math.sqrt(1 - point[0]) / Math.sqrt(1 + point[0]) / 2;
    quot2 = Math.sqrt(1 + point[0]) / Math.sqrt(1 - point[0]) / 2
    if(point[1] >= 0) {
        return quot1 - quot2;
    }
    else {
        return quot2 - quot1;
    }
}
```

## 3 Aufgabe 3: Logistisch

### 3.1 Zusammenfassung der Aufgabenstellung

Ein Logistikunternehmen möchte Container zwischen drei Standorten mit Fahrzeugen transportieren, die jeweils eine Strecke pro Tag fahren können. Diese werden nach Bedarf für eine Woche im Vorraus gemietet.

- a) Ein Programm, welches errechnet, wie viele Fahrzeuge an welchen Standorten für eine Woche angemietet werden müssen, wenn keine Leerfahrten gemacht werden.
- b) Ein Programm, welches einen Tourplan errechnet, mit dem – bei erlaubten Leerfahrten – die Gesamtanzahl der Fahrzeuge verringert werden kann.

Leider konnten wir aus zeitlichen Gründen nur den ersten Teil der Aufgabe bearbeiten, wir möchten aber trotzdem auch diese Aufgabe mit einsenden. Im Folgenden wird daher nur auf die Teilaufgabe 1. bzw. a) eingegangen.

### 3.2 Lösungsidee

Zuerst wird die Datei mit den Angaben über die zu fahrenden Strecken geladen. Danach werden die Werte nicht nach Strecken, sondern nach ankommenden und abfahrenden Fahrzeugen an den jeweiligen Orten geordnet.

Jetzt wird für jeden Tag jeder Ort durchgegangen. Die ankommenden Fahrzeuge werden zur Zahl der Fahrzeuge an diesem Ort addiert und die abfahrenden subtrahiert. Fällt nun der Wert der Fahrzeuge an diesem Ort unter 0, so müssen am Anfang der Woche mehr Fahrzeuge gemietet werden. Daher muss der Betrag dieses negativen Wertes zur Zahl der zu mietenden Fahrzeuge am jeweiligen Ort hinzugezählt werden.

Sind alle Tage durchgearbeitet liegt die Anzahl an Fahrzeugen vor, die insgesamt gemietet werden müssen.

### 3.3 Programmdokumentation

Das Programm wurde in Python für Python ab 2.6 entwickelt. Python 3 wird nicht unterstützt, da es nicht abwärtskompatibel ist. Als erstes lädt das Programm die als Argument übergebene Datei, in der sich der Fahrplan befindet. Mit *map()* wird für jede Zeile die Funktion *parseLine()* aufgerufen, die die Zeile in eine Liste aus Integern umwandelt.

Danach ermittelt *departuresOnDay()* beziehungsweise *arrivalsNextDay()* durch simple Umrechnung aus dieser Liste die Anzahl der abfahrenden beziehungsweise ankommenden Wagen. Da die Transporte immer einen Tag unterwegs sind, sind die Ankünfte auch um einen Tag verschoben.

An dieser Stelle erfolgt der Sprung ins eigentliche Programm. Hier springt das Programm entweder zur Aufgabe a oder b beziehungsweise in einen Testmodus für diese. Der Testmodus funktioniert im Prinzip wie der normale, er erzeugt nur eine ausführlichere Ausgabe.

Die Funktion für den Modus *a*, *mode\_a()*, lässt von *neededVehiclesA()* ermitteln, wie viele Fahrzeuge gebraucht werden und gibt dies aus. Diese fasst die Ergebnisse, die sie aus *neededVehiclesDetailed* erhält, zusammen.

Diese ermittelt schließlich wie oben beschrieben, an welchem Punkt mehr Fahrzeuge für die ganze Woche gemietet werden müssen.

### 3.4 Programmablaufprotokoll

Das Programm kann mit verschiedenen Argumenten aufgerufen werden. Eine Auflistung derer erhält man mit der Option *-h*.

```
$ ./logistik.py -h
Usage: logistik.py [options] SOURCEFILE
```

Options:

```
-h, --help          show this help message and exit
-m MODE, --mode=MODE Modus: 'a', 'b', 'testa' oder 'testb' (Standard ist 'a')
```

Verpflichtend ist die Angabe einer Datei, die das Auftragsbuch enthält. Diese muss im gleichen Format wie die Beispiele aus dem Material vorliegen.

Mit dem Modus kann die zu bearbeitende Teilaufgabe angegeben werden. *a* steht für 1, *b* für 2. Für beide Aufgaben gibt es einen Testmodus, der durch ein vorangestelltes *test* aufgerufen wird und eine ausführlichere Ausgabe zeigt.

Die Ausgabe im normalen Modus für die Aufgabe *a* enthält drei Zahlen. Diese stellen die zu mietenden Fahrzeuge am Standort A, B und C in dieser Reihenfolge dar.

Ein Beispielaufruf des Programms:

```
$ ./logistik.py logistisch1.txt
3 17 25
```

Und hier im Testmodus:

```
$ ./logistik.py -m testa logistisch1.txt
Start mit 3 17 25
```

Tag 1:

```
A: 3 ab und 0 an -> 0
B: 7 ab und 0 an -> 10
C: 11 ab und 0 an -> 14
```

Tag 2:

```
A: 5 ab und 8 an -> 3
B: 9 ab und 7 an -> 8
C: 7 ab und 6 an -> 13
```

Tag 3:

```
A: 7 ab und 10 an -> 6
B: 11 ab und 3 an -> 0
C: 15 ab und 8 an -> 6
```

Tag 4:

A: 3 ab und 12 an -> 15

B: 7 ab und 11 an -> 4

C: 11 ab und 10 an -> 5

Tag 5:

A: 3 ab und 8 an -> 20

B: 7 ab und 7 an -> 4

C: 11 ab und 6 an -> 0

Tag 6:

A: 7 ab und 8 an -> 21

B: 11 ab und 7 an -> 0

C: 3 ab und 6 an -> 3

## 3.5 Programmtext

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

import sys
from optparse import OptionParser

def main():
    # Behandlung der Argumente
    usage = "usage: %prog [options] SOURCEFILE"
    parser = OptionParser(usage=usage)
    parser.add_option(
        "-m",
        "--mode",
        dest="mode",
        type="choice",
        default='a',
        help="Modus: 'a', 'b', 'testa' oder 'testb' (Standard ist 'a')",
        choices=["a", "b", "testa", "testb"]
    )
    (options, args) = parser.parse_args()

    if len(args) < 1:
        # es muss eine Quelldatei angegeben werden
        parser.print_usage()
        return 1

    try:
        infile = False
        infile = open(args[0])
        # Wandle jede Zeile in eine Liste von Integern um
        data = map(parseLine, infile)
```

```
# eliminiere leere Zeilen
data = filter(lambda x: x is not None, data)
except IOError as (errno, strerror):
    # Datei existiert nicht, keine Rechte o.ä. ...
    print "Von der angegebenen Datei konnte nicht gelesen werden"
    print "Fehler " + str(errno) + ": " + strerror
    return 1
finally:
    if infile:
        infile.close()

if(len(data) != 6):
    print "Ungültige Quelldatei!"
    return 1

departures = [departuresOnDay(x) for x in data]
# Ankünfte sind um einen Tag verschoben
arrivals = [[0,0,0]] + [arrivalsNextDay(x) for x in data[:-1]]

# starte in den spezifizierten Modus
functions = {
    'a': mode_a,
    'b': mode_b,
    "testa": mode_testa,
    "testb": mode_testb
}
return functions[options.mode](departures, arrivals)

def parseLine(line):
    """Wandelt eine Zeile der Quelldatei in eine Liste von Zahlen um. Für
    Kommentarzeilen (die mit einem '#' beginnen, wird 'None' zurückgegeben"""
    if line[0] == '#':
        # Zeile ist ein Kommentar
        return None
    numbers = []
    buf = ""
    for char in line:
        if (char == '\n') or (char == ' '):
            # Zahl ist komplett:
            # wandle buffer in Integer um und hänge es an die Liste
            if buf.isdigit():
                number = int(buf)
                buf = ""
                numbers.append(number)
            else:
                # Ansonsten Buffer erweitern
                if char.isdigit():
                    buf += char
    return numbers
```



```
        " an -> " + str(locations[loc]))
    print
    return 0

def mode_testb(departures, arrivals):
    raise(NotImplementedError)

def neededVehiclesA(departures, arrivals):
    """gibt die Anzahl der benötigten Fahrzeuge an jedem Ort als Liste zurück,
    wenn keine Leerfahrten gemacht werden"""
    vehicles = neededVehiclesDetailed(departures, arrivals)
    vehicles = [sum(x) for x in vehicles]
    return vehicles

def neededVehiclesDetailed(departures, arrivals):
    """Gibt die Anzahl der zusätzlich benötigten Fahrzeuge pro Tag und Ort
    zurück"""
    locations = [0, 0, 0]
    vehicles = [[], [], []]
    for day in range(6):
        for loc in range(3):
            # Differenz zwischen ankommenden und abfahrenden Fahrzeugen
            diff = arrivals[day][loc] - departures[day][loc]

            if (locations[loc] + diff) < 0:
                # Wenn die Fahrzeuge am Ort nicht ausreichen, wird die Zahl der
                # fehlenden Fahrzeuge auf die Anzahl der insgesamt am Ort
                # benötigten addiert
                vehicles[loc].append(-1 * (locations[loc] + diff))
                locations[loc] = 0
            else:
                vehicles[loc].append(0)
                locations[loc] += diff
    return vehicles

if __name__ == '__main__':
    main()
```

## 4 Aufgabe 4: Drehzahl / Kartenspiel

### 4.1 Zusammenfassung der Aufgabenstellung

Gegeben ist ein Kartenspiel, das aus Karten mit den Nummern eins bis neun besteht. Mit zwei normalen (sechseitigen) Würfeln wird eine Zahl gewürfelt. Daraufhin müssen eine oder zwei Karten umgedreht werden, deren Werte summiert gleich der Summe der Augenzahlen auf den Würfeln sind. Sobald einmal ein Zug nicht mehr möglich ist (da die gewürfelte Zahl nicht aus den noch offen liegenden Karten zusammengesetzt werden kann), endet das Spiel. Die erreichte Punktzahl errechnet sich aus der Summe der bereits umgedrehten Karten. Ziel ist es, eine möglichst hohe Punktzahl zu erreichen.

Es sollte ein Programm geschrieben werden, dem ein Spieler mitteilen kann, was er gewürfelt hat, und das daraufhin sagt, welche Karten der Spieler umdrehen sollte, um die höchstmögliche Punktzahl zu erreichen.

### 4.2 Lösungsidee

Das Programm berechnet nach jedem eingegebenen Würfelwurf die beste Kombination von Karten, die umgedreht werden soll. Da die Punkte, die durch den jeweiligen Wurf erzielt wurden, nicht beeinflussbar sind, kann es nur zu einer Steigerung der Punktzahl kommen wenn die Auswirkung auf zukünftige Würfe berücksichtigt wird.

Dazu werden zunächst alle Kombinationen, die überhaupt umgedreht werden können, ermittelt. Dies geschieht auf Basis der erzielten Punktzahl der Würfel und der bereits umgedrehten Karten.

Danach werden für jede mögliche Kombination die Kosten berechnet, das heißt ein Wert, der angibt wie viele Punkte wahrscheinlich nicht erzielt werden können wenn die entsprechende Kombination gewählt wird. Der Benutzer bekommt dann die Kombination mit den niedrigsten Kosten vorgeschlagen.

Um zu ermitteln wie viel Punkte ein Spieler bei optimalen Entscheidungen erreichen kann, werden sehr viele Spiele simuliert und ein Durchschnittswert aus den erreichten Punktzahlen gebildet.

### 4.3 Programmdokumentation

Das Programm ist in Python implementiert und läuft ab Python 2.6. Für Python ab 3.0 müssen Anpassungen von Modulnamen etc. vorgenommen werden, auf die hier jedoch verzichtet wurde. Es besteht aus zwei Teilen, *kartenspiel.py* und *test.py*.

#### 4.3.1 kartenspiel.py

*kartenspiel.py* startet in einen interaktiven Modus, bei dem der Spieler seine Würfelergebnisse eingeben kann und darauf Vorschläge erhält.

Als erstes wird von *main()* die Funktion *spiel()* in einem Modus gestartet, in dem sie sowohl Eingaben der Würfelzahlen entgegen nimmt, als auch die Tipps ausgibt. Sie nimmt in einer Schleife so lange Würfelergebnisse vom Benutzer an, bis keine Karten mehr umgedreht werden können.

Nach jeder Eingabe wird die Funktion *karten\_auswaehlen()* aufgerufen, die die Entscheidung über die Karten trifft, die umgedreht werden sollen. Dafür ermittelt sie erst alle Kombinationen mit *moeglichkeiten()* und lässt von *karten\_kosten()* die Kosten der einzelnen Karten neu berechnen. Dann verknüpft sie beide Ergebnisse, indem die Kosten für die Kombinationen ausgerechnet werden. Im letzten Schritt wird die beste Kombination ausgewählt und zurückgegeben.

Falls keine Kombination gefunden wurde ist das Spiel jetzt vorbei und die erreichten Punkte werden ausgegeben. Andernfalls kann der Benutzer einen neuen Würfelwurf eingeben.

### 4.3.2 test.py

Im Gegensatz dazu bietet *test.py* die Möglichkeit, die Funktion von *kartenspiel.py* zu testen, indem Spiele simuliert werden können. Das Programm bindet dabei *kartenspiel.py* als Modul ein und ruft dessen Funktion *spiel()* mit dem durch den Benutzer spezifizierten Modus mehrmals auf. Die Punktzahlen die diese Funktion liefert, werden addiert, durch die Anzahl der Spiele geteilt und schließlich dieser Durchschnitt dem Nutzer ausgegeben.

Dabei gibt es die Möglichkeit, sowohl das Auswahlkriterium für die Kombination als auch die Anzahl der simulierten Spiele anzugeben. Für das Auswahlkriterium gibt es drei verschiedene Möglichkeiten:

**‘gut’**: Es wird die beste Kombination gewählt (wie im interaktiven Modus)

**‘zufall’**: Das Programm trifft gar keine Entscheidung und simuliert damit ein Spiel ohne Beratung

**‘schlecht’**: Es werden stets die schlechtesten Möglichkeiten gewählt

Damit kann getestet werden, wie gut die Tipps des Programms sind und wie viel Punkte damit erreicht werden können.

## 4.4 Programmablaufprotokoll

Bei Ausführen des Skriptes *kartenspiel.py* kann der Benutzer eingeben, was er gewürfelt hat, und der Computer gibt daraufhin eine Anweisung, welche Karte(n) nun umgedreht werden sollen. Ebenfalls ausgegeben wird ein Statusbericht, welche Karten bereits umgedreht sind und welche noch offen liegen sollten. Beispielablauf:

```
$ ./kartenspiel.py
Würfelergebnis: 3
drehe [3] um
umgedrehte Karten: [3]
offene Karten: [1, 2, 4, 5, 6, 7, 8, 9]
```

```
Würfelergebnis: 3
drehe [1, 2] um
umgedrehte Karten: [3, 1, 2]
offene Karten: [4, 5, 6, 7, 8, 9]
```

Würfelergebnis: 3  
Konnte 3 nicht auf die Karten aufteilen  
Erzielte Punktzahl: 6

Das Skript *test.py* bietet die Möglichkeit, das Programm zu testen. Ohne Parameter spielt es fünfhundert Mal mit zufälligen Würfelergebnissen und gibt die erreichte Durchschnittspunktzahl aus. Diese Anzahl kann mit dem Kommandozeilenparameter `-n` modifiziert werden. Über `-m` kann auch der Modus gewechselt werden (genauere Informationen erhält man mit `-h`). Hiermit kann man, um Vergleichswerte zu erhalten, viele Spiele mit besonders schlechter oder auch mit zufälliger Kartenauswahl durchführen. Beispielausgabe mit der Einstellung, dass das Programm die bestmögliche Wahl trifft:

```
$ ./test.py
500 Spiele simuliert
Durchschnittlich erreichte Punktzahl: 33.388
```

Ausgabe der Hilfe (`-h`)

```
$ ./test.py -h
Usage: test.py [options]
```

Options:

```
-h, --help          show this help message and exit
-n ANZAHL, --anzahl=ANZAHL
                    simuliere ANZAHL Spiele
-m MODE, --mode=MODE wie die Karten ausgewaehlt werden sollen ('gut',
                    'schlecht' oder 'zufall')
```

## 4.5 Durchschnittliche Punktzahl

Mit dem Programm *test.py* konnten wir eine durchschnittlich durch die Tipps des Programms erreichbare Punktzahl von etwa **33** ermitteln. Eine zufällige Auswahl ergab durchschnittlich ungefähr **27** Punkte, eine möglichst schlechte Auswahl circa **23**. Daraus ergibt sich, dass mit dem Programm eine durchschnittliche Steigerung der Punktzahl um gut 20% gegenüber einer rein zufälligen Auswahl erfolgen kann.

## 4.6 Programmtext

### 4.6.1 kartenspiel.py

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
```

```
import itertools
import random
```

```
"""Definitionen:
```

Augenzahl: Das Ergebnis eines Würfelwurfs

Kosten: Die Kosten einer Karte oder einer Kartenkombination sind ein Wert, der die Nachteile für die folgenden Züge angibt, wenn diese Karte oder

-kombination gewählt wird. Sie werden aufgrund von Wahrscheinlichkeiten ermittelt.

```
"""
```

```
def moeglichkeiten(zahl, k):
```

```
    """
```

```
    Generiert alle möglichen Kombinationen von Karten aus k, deren Summe genau zahl ergibt
```

```
    """
```

```
    karten = k[:] # kopiere Liste statt der Referenz
```

```
    moeglichkeiten = []
```

```
    # die Karten können nicht größer sein als die Augenzahl
```

```
    moegl_karten = filter(lambda x: x <= zahl, karten)
```

```
    for karte1 in moegl_karten:
```

```
        # beide Karten zusammen ergeben die zahl
```

```
        karte2 = (zahl-karte1)
```

```
        if karte1 is karte2:
```

```
            # Karten gleichen Wertes existieren nur ein Mal  
            continue
```

```
        if karte2 is 0:
```

```
            # es ist nur eine Karte notwendig
```

```
            moeglichkeiten.append([karte1])
```

```
            karten.remove(karte1)
```

```
        elif karte2 in karten:
```

```
            moeglichkeiten.append([karte1, karte2])
```

```
            karten.remove(karte1)
```

```
            karten.remove(karte2)
```

```
    return moeglichkeiten
```

```
def karten_kosten(haeufigkeit_augenzahl, karten):
```

```
    """
```

```
    Berechnet die Kosten der einzelnen Karten basierend auf den noch nicht umgedrehten
```

```
    """
```

```
    karten_kosten = {}
```

```
    for augenzahl, haeufigkeit in haeufigkeit_augenzahl.iteritems():
```

```
        # ermittle alle Kartenkombinationen für die Augenzahl
```

```
        moegl = moeglichkeiten(augenzahl, karten)
```

```
        # generiere aus allen darian verwendeten Karten eine Liste
```

```
        # Wichtig: Karten können mehrfach vorkommen!
```

```
        verwendete_karten = itertools.chain(*moegl)
```

```
        # addiere auf die Kosten jeder Karte die Häufigkeit der Augenzahl
```

```
        for karte in verwendete_karten:
```

```
            if karte in karten_kosten.keys():
```

```
                karten_kosten[karte] += haeufigkeit
```

```
            else:
```

```
                karten_kosten[karte] = haeufigkeit
```

```
    return karten_kosten
```

```
def karten_auswaehlen(haeufigkeit_augen Zahl, karten, wuerfelsumme,
    modus="gut"):
    """
    Wählt die wahrscheinlich beste Kartenkombination aus und gibt diese zurück
    """
    # berechne die Kosten jeder Karte neu mit den noch vorhandenen Karten
    kosten_tabelle = karten_kosten(haeufigkeit_augen Zahl, karten)
    # generiere alle möglichen Kartenkombinationen
    moegl = moeglichkeiten(wuerfelsumme, karten)
    kosten_der_moeglichkeiten = {}
    if moegl == []:
        # mit den vorhandenen Karten kann die Augenzahl nicht abgebildet werden
        return []

    for moeglichkeit in moegl:
        # die Kosten einer Möglichkeit sind die Summe der Kosten der Karten aus
        # denen die Möglichkeit besteht
        kosten = sum(map(lambda x: kosten_tabelle[x], moeglichkeit))
        kosten_der_moeglichkeiten[kosten] = moeglichkeit
    if modus == "zufall":
        #wähle irgendeine Möglichkeit aus
        geringste_kosten = random.choice(list(kosten_der_moeglichkeiten))
    elif modus == "schlecht":
        #wähle die Möglichkeit mit den höchsten Kosten aus
        geringste_kosten = sorted(kosten_der_moeglichkeiten)[-1]
    else:
        # finde die Möglichkeit mit den geringsten Kosten und gib diese zurück
        geringste_kosten = sorted(kosten_der_moeglichkeiten)[0]
    return kosten_der_moeglichkeiten[geringste_kosten]

def spiel(wuerfle, ausgeben, modus="gut"):
    """
    Führt ein Spiel durch und gibt die Punktzahl zurück
    wuerfle(): Funktion, die eine maximale Augenzahl eines Würfels bekommt und
        die gewürfelte Augenzahl zurückgibt
    """
    karten = range(1,10) # die Spielkarten, die am Anfang vorliegen
    MAX_WUERFEL = 6
    MAX_AUGENZAHL = MAX_WUERFEL * 2
    MIN_AUGENZAHL = 1 * 2
    umgedrehte_karten = [] # Karten die bereits aus dem Spiel genommen wurden
    haeufigkeit_augen Zahl = {}

    # ermittle die Häufigkeiten für alle möglichen Augenzahlen
    for wuerfel_1 in range(1, MAX_WUERFEL+1):
        for wuerfel_2 in range(1, MAX_WUERFEL+1):
            augenzahl = wuerfel_1 + wuerfel_2
```

```
        if augenzahl not in haeufigkeit_augenzahl.keys():
            haeufigkeit_augenzahl[augenzahl] = 1
        else:
            haeufigkeit_augenzahl[augenzahl] += 1

# führe für jeden Schleifendurchlauf einen Spielzug durch
punkte = 0
wuerfelwerte = []
while karten != []:
    korrekte_eingabe = False
    while korrekte_eingabe == False:
        try:
            augenzahl = wuerfle(MAX_WUERFEL)
            if augenzahl >= MIN_AUGENZAHL and augenzahl <= MAX_AUGENZAHL:
                korrekte_eingabe = True
                continue
        except ValueError:
            pass
        print("Die Eingabe muss eine ganze Zahl sein und zwischen "
              + str(MIN_AUGENZAHL) + " und " + str(MAX_AUGENZAHL)
              + " liegen! Neuer Versuch:")

    wuerfelwerte.append(augenzahl)
    ausgewaehlt = karten_auswaehlen(haeufigkeit_augenzahl, karten,
                                     augenzahl, modus)
    if ausgewaehlt == []:
        # es können keine Karten ausgewählt werden
        # folglich ist das Spiel zu Ende
        if ausgeben:
            print "Konnte", augenzahl, "nicht auf die Karten aufteilen"
        break
    for karte in ausgewaehlt:
        # drehe Karte um
        karten.remove(karte)
        umgedrehte_karten.append(karte)

    if ausgeben:
        print "drehe", ausgewaehlt, "um"
        print "umgedrehte Karten:", umgedrehte_karten
        print "offene Karten:", karten
        print
    punkte = sum(umgedrehte_karten)
    return punkte

def wuerfel_eingabe(max_wuerfel):
    return int(raw_input("Würfelergebnis: "))

def wuerfel_zufaellig(max_wuerfel):
    return random.randint(1, max_wuerfel) + random.randint(1, max_wuerfel)
```

```
def main():
    punkte = spiel(wuerfel_eingabe, ausgeben=True)
    print "Erzielte Punktzahl:", punkte
    return 0

if __name__ == '__main__':
    main()
```

## 4.6.2 test.py

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

from kartenspiel import *
from optparse import OptionParser

def main():
    # Parsen der Kommandozeilenoptionen
    parser = OptionParser()
    parser.add_option(
        "-n",
        "--anzahl",
        dest="anzahl",
        type="int",
        help="simuliere ANZAHL Spiele",
        default=500,
        metavar="ANZAHL"
    )
    parser.add_option(
        "-m",
        "--mode",
        dest="mode",
        type="choice",
        help="""wie die Karten ausgewaehlt werden sollen ('gut',
        'schlecht' oder 'zufall')""",
        default="gut",
        choices=["gut", "schlecht", "zufall"]
    )
    (options, args) = parser.parse_args()

    punkte = 0
    for i in range(options.anzahl):
        punkte += spiel(wuerfel_zufaellig, False, options.mode)
    avg = punkte / float(options.anzahl)
    print options.anzahl, "Spiele simuliert"
    print "Durchschnittlich erreichte Punktzahl:", avg
```

```
    return 0
```

```
if __name__ == '__main__':  
    main()
```