Entwicklung eines Softwaresystems 2019

Landkartenerstellung

Vorgelegt von:
Christoph Schirmel
Prüfungsnummer: 101-20510
Ausbildungsbetrieb: CAE Elektronik GmbH

Inhaltsverzeichnis

1 Aufgabenbeschreibung **4**

1.1 Eingabedatei 4

1.2 Ausgabedatei 6

1.3 Spezial- und Sonderfälle bei der Eingabe 7

2 Objektorientierter Entwurf **10**

2.1 Klassendiagramm 10

2.2 Beschreibung 11

2.3 Ablauf Eingabe 11

2.4 Ablauf Ausgabe 13

2.5 Ablauf Hauptalgorithmus 13

2.5.1 TODO 13

2.6 Sequenzdiagramm 13

3 Änderungen zum Montagsteil **13**

3.1 Änderungen in der Klassenstruktur 13

3.2 Änderungen beim Algorithmus 13

4 Zusammenfassung und Ausblick **13**

5 Benutzeranleitung **13**

5.1 Ordnerstruktur 13

5.2 Voraussetzungen 14

5.3 Laufzeitumgebung 14

5.4 Installation 14

5.5 Ausführen des Programms 14

5.5.1 Direkt 14

5.5.2 Mit dem Skript 14

5.6 Restriktionen beim Dateinamen 15

5.7 Anmerkung 16

6 Tests **16**

6.1 IHK Tests 16

6.1.1 IHK Example 01 etc. 16

6.2 Funktionierende Tests 16

6.2.1 F01 bliblablub etc. 16

6.3 Syntax-Fehler Tests 16

6.3.1 S01 lirumlarum 16

6.4 Logik Fehler Tests 16

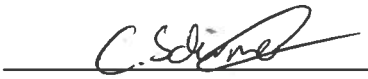
6.4.1 L01 loremipsum 16

6.5 Optional: Code Coverage Tests, Performance o.ae 16

7 Source-Code **16**

Eidesstattliche Erklärung

Ich erkläre verbindlich, dass das vorliegende Prüfprodukt von mir selbstständig erstellt wurde. Die als Arbeitshilfe genutzten Unterlagen sind in der Arbeit vollständig aufgeführt. Ich versichere, dass der vorgelegte Ausdruck mit dem Inhalt der von mir erstellten digitalen Version identisch ist. Weder ganz noch in Teilen wurde die Arbeit bereits als Prüfungsleistung vorgelegt. Mir ist bewusst, dass jedes Zuwiderhandeln als Täuschungsversuch zu gelten hat, der die Anerkennung des Prüfprodukts als Prüfungsleistung ausschließt.

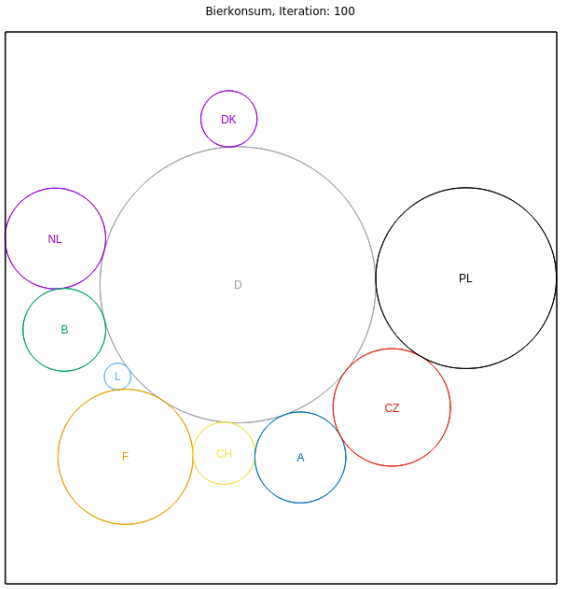


Christoph Schirmel, Stolberg den 17.05.2019

1 Aufgabenbeschreibung

Die MATSEgraphic AG stellt den Auftrag zur Erstellung von schemenhaften Karten auf Basis vorgegebener Kennwerte. Ausgangspunkt ist eine gegebene Menge an beliebigen Staaten mit ihrer realen Lage, angegeben in Läengen- und Breitengrad, ihren realen Nachbarschaftsbeziehungen und dem jeweiligen Kennwert. Die Staaten sollen auf der erstellten Karte als Kreis dargestellt werden. Die Gröesse des Kreises ist proportional zu der gegebenen Kenngröesse, also wird der Kreis für gröessere Kennwerte gröesser und für kleinere Kennwerte kleiner. In der erstellten Karte sollen die Lage- und Nachbarschaftsbeziehungen so gut wie möglich erhalten bleiben. Die Qualität der Karte bemisst sich insgesamt am Abstand der Länder zueinander, sowie den Lage- und Nachbarschaftsbeziehungen. Überschneidungen und zu grosse Abstände mindern die Qualität der Karte und sollen vermieden werden. Die numerische Grössenordnung der Karte kann beliebig skaliert werden und hat keinen unmittelbaren Einfluss auf die Qualität der Karte, welche lediglich durch die oben beschriebenen Kriterien bestimmt wird.

Beispiel:



Das obige Beispiel zeigt eine schematische Darstellung Mitteleuropas mit Deutschland und seinen Nachbarstaaten. Der Kennwert ist der Bierkonsum der einzelnen Staaten. Erkennbar ist, dass die Staaten nicht im Verhältnis ihrer Fläche sondern zum Kennwert dargestellt werden.

1.1 Eingabedatei

Zur Erstellung der Landkarte sollen über eine Eingabedatei Daten eingelesen werden. Die Datei hat folgendes Format: Die erste Zeile enthält den Titel der Karte, z.B. "Fläche der Staaten". Zeilen, die mit "#"beginnen, sind Kommentare und werden im weiteren Ablauf nicht betrachtet. Nach dem ersten Kommentar folgen alle Staaten zeilenweise mit jeweils einem Staat pro Zeile. Dabei enthält jede Zeile die folgenden Informationen:

- Autokennzeichen, z.B. "D"für Deutschland
- Kennzahl
- Längengrad
- Breitengrad

Die Informationen sind jeweils durch Whitespace, Tab oder Leerzeichen, getrennt. Auf die Liste der Staaten folgt ein Kommentar mit dem Hinweis, dass im Anschluss die Nachbarschaftsbeziehungen folgen. Jene folgen ebenfalls zeilenweise. Die Nachbarschaftsbeziehungen werden bidirektional angegeben, d.h. ist ein Staat A ein Nachbar von Staat B, so ist automatisch auch B Nachbar von A. Das Format der Nachbarschaftsbeziehungen sieht wie folgt aus: `<Kennz. Ausgangsland>: <Kennzeichen Nachbar 1> <...> <Kennzeichen Nachbar N>` Die einzelnen Nachbarstaaten sind ebenfalls durch Whitespace (Leerzeichen oder Tab) getrennt. Grundsätzlich wird angenommen, dass das Format der Eingabedatei syntaktisch korrekt ist. Die Syntax wird daher nicht mehr auf Korrektheit überprüft.

Beispiel:
Die Eingabedatei vom vorherigen Beispiel könnte wie folgt aussehen:

```
Bierkonsum
# Staat Fläche Längengrad Breitengrad
D 8692 10.0 51.3
NL 1156 5.3 52.2
B 781 4.8 50.7
L 80 6.1 49.8
F 2077 2.8 47.4
CH 440 8.2 46.9
A 945 14.2 47.6
CZ 1573 15.3 49.8
PL 3724 18.9 52.2
DK 360 9.6 56.0
# Nachbarschaften
D: NL B L F CH A CZ PL DK
NL: B
L: F
F: CH
CH: A
A: CZ
CZ: PL
```

1.2 Ausgabedatei

Nach dem Einlesen der oben beschriebenen Daten und der Berechnung der Landkarte soll eine Ausgabe mit den berechneten Werten erstellt werden. Diese soll dem folgenden Format genügen:

```
reset set xrange [<xmin>:<xmax>] set yrange [<ymin>:<ymax>]
set size ratio 1.0
set title "<Name des Kennwertes>, Iteration: <nr>"
unset xtics
unset ytics
$data << EOD
<Liste aus <xpos> <ypos> <radius> <autokennzeichen> <id> >
EOD
plot \
'$data' using 1:2:3:5 with circles lc var notitle, \
'$data' using 1:2:4:5 with labels font "arial,9" tc variable notitle
```

Die Intervalle `[<xmin>:<xmax>]` und `[<ymin>:<ymax>]` geben jeweils die kleinste und grösste x- bzw. y-Koordinate der Landkarte an. `<Name des Kennwertes>` gibt den Namen des Kennwertes an, der in der Eingabedatei in der 1. Zeile angegeben wird. Die Anzahl der Iterationen wird in `<nr>` angegeben. In der `<Liste aus <xpos> <ypos> <radius> <autokennzeichen> <id> >` werden alle Staaten nacheinander aufgelistet. Die Werte `<xpos> und <ypos>` sind die Koordinaten des jeweiligen Kreismittelpunktes, der `<radius>` ist der ermittelte Radius des Kreises. Das `<autokennzeichen>` gibt das Kennzeichen des Staats an und die `<id>` ist eine fortlaufende Nummerierung beginnend bei 0, die für die Farbgebung in Gnuplot erforderlich ist.

Beispiel:

Die Ausgabe für das Beispiel „Bierkonsum“ könnte wie folgt aussehen:

```
reset
set xrange[122.2902588107926:332.4746754411094]
set yrange[990.7016847962033:1200.88610142652]
set size ratio 1.0
set title "Bierkonsum, Iteration: 100"
unset xtics
unset ytics
$data ¡¡ EOD
211.053730834 1104.45871827 52.5999004819 D 0
141.472704651 1122.09603019 19.1824458406 NL 1
144.871946688 1087.31207915 15.7670549282 B 2
165.183777819 1069.53934032 5.04626504404 L 3
168.191312507 1038.92448868 25.7124412222 F 4
205.719560567 1040.2435555 11.8345405454 CH 5
234.860729713 1038.69277989 17.3436686558 A 6
269.709149473 1057.75886679 22.3763591982 CZ 7
298.045240126 1106.99471738 34.4294353156 PL 8
207.661916065 1167.67099407 10.7047446969 DK 9
EOD
plot \
'$data' using 1:2:3:5 with circles lc var notitle, \
'$data' using 1:2:4:5 with labels font “arial,9“ tc variable notitle
```

1.3 Spezial- und Sonderfälle bei der Eingabe

Syntaxfehler in der Eingabedatei sind laut Aufgabenstellung ausgeschlossen. Daher benutze ich als Format für meine Eingabedatei restriktiv das unter Punkt 1.1 angegebene Format, inklusive der Kommentarzeilen.
Den Inhalt der Kommentarzeilen ignoriere ich, allerdings benutze ich sie zur Orientierung beim Einlesen der Zeilen. Vor dem ersten Kommentarzeichen, das ein # ist, befindet sich der Titel, auf den ersten Kommentar folgen die Staaten und nach dem zweiten Kommentar kommen die Nachbarschaftsbeziehungen.

Vorschriften:

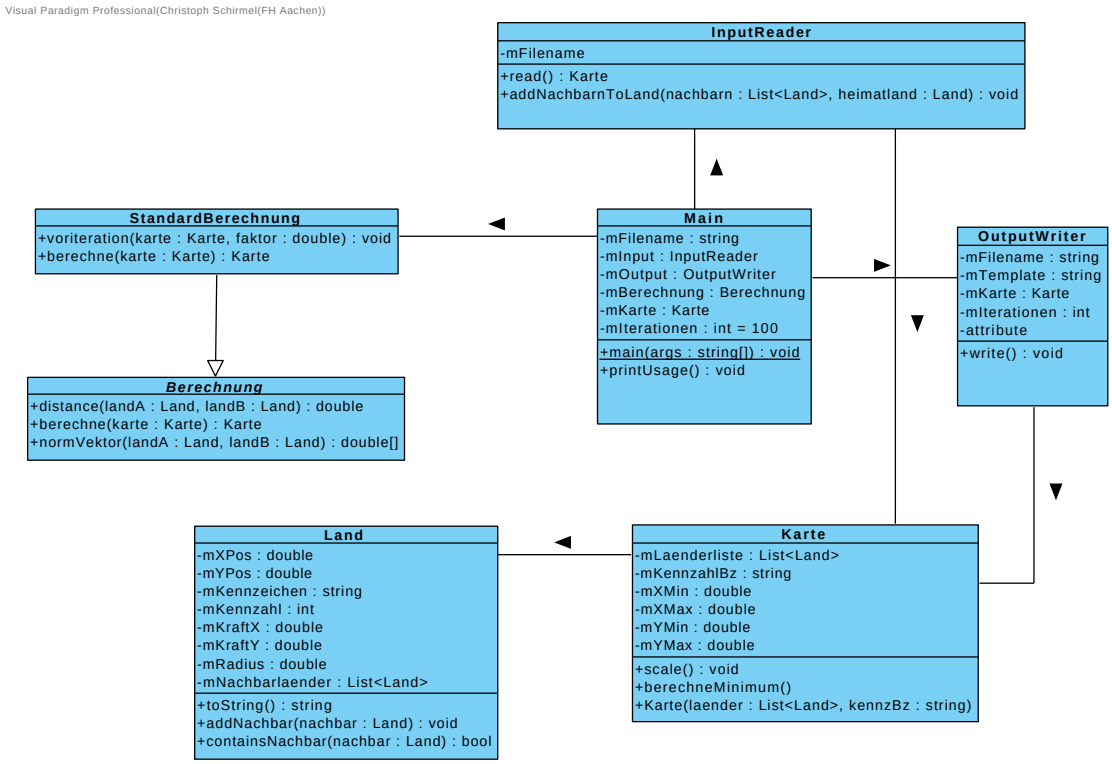
- Doppelt vorkommende Autokennzeichen sind nicht erlaubt, da diese eindeutig sein müssen, um einen Staat eindeutig identifizieren zu können.

- Zwei mal der selbe Mittelpunkt für zwei verschiedene Staaten ist nicht erlaubt.

- Der Kennwert muss mindestens 1 sein.
- Kennwerte sollten nicht zu gross gewählt werden, weil das Koordinatensystem durch den gewählten Datentyp double beschränkt ist und es ansonsten in Ausnahmefällen durch die berechneten Kräfte zu einer Überschreitung des erlaubten Wertebereichs kommen könnte. In diesem Fall wäre es erforderlich die Kennzahl anzupassen, bpsw. durch eine Änderung der Grössenordnung.
- Die Längen- und Breitengrade müssen im Wertebereich des *World Geodetic System 1984 (WGS84)* liegen. Das bedeutet, dass die Breitengrade im Intervall [-90;+90] liegen müssen und Längengrade im Intervall [-180;+180]. Andere Werte werden nicht akzeptiert.
- Alle Staaten müssen zusammenhängen. Ansonsten ist die Berechnung nicht durchfürbar.

2 Objektorientierter Entwurf

2.1 Klassendiagramm



2.2 Beschreibung

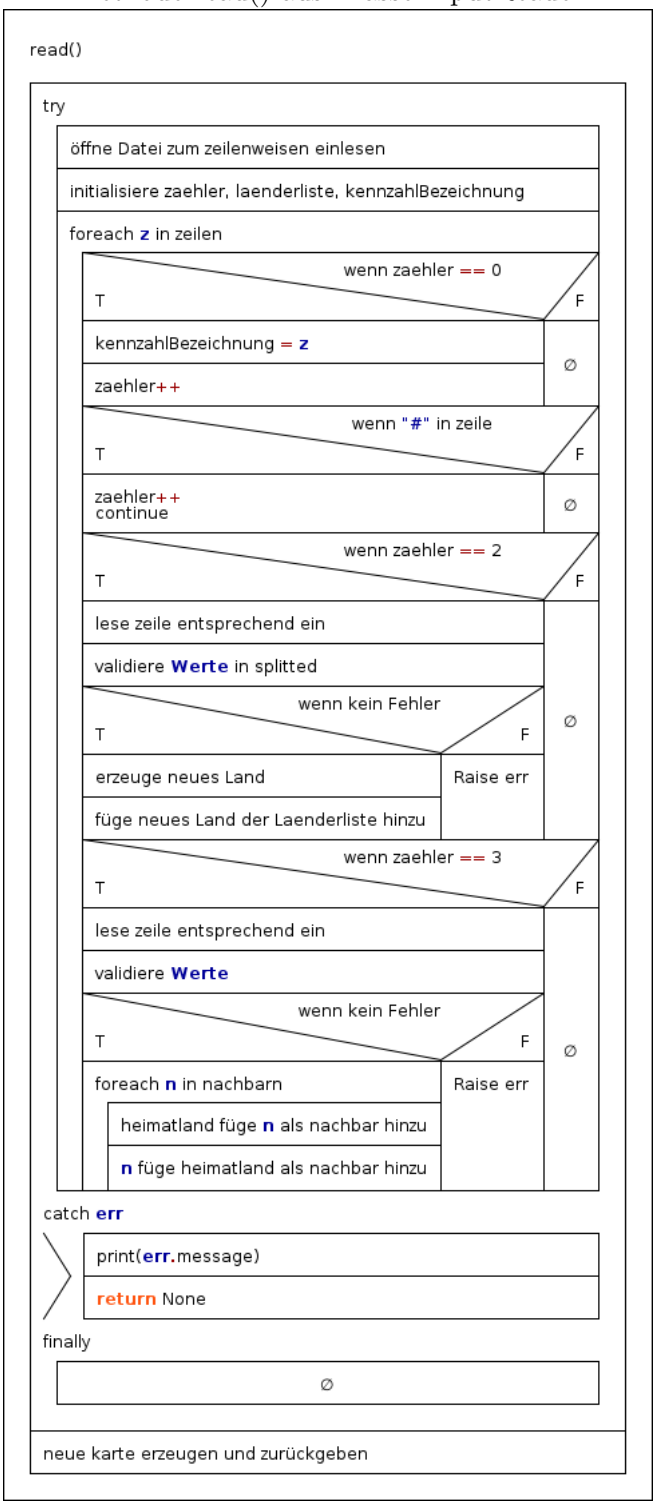
Die Hauptklasse des Programms ist die Klasse `Main`, die den Einstiegspunkt des Programms markiert und alle anderen Vorgänge steuert. Hierfür hat sie über ihre Attribute Zugriff auf die Funktionen für Ein- und Ausgabe, die Berechnung und die zurzeit eingelesenen Daten. Auch die Anzahl der Iterationen, die berechnet werden sollen, werden in der Klasse Main gespeichert. Das zeilenweise Einlesen der Eingabedatei ist in der Klasse `InputReader` in der Methode `read` implementiert, die ein Objekt der Container-Klasse `Karte` zurückgibt. In einem Objekt von Karte sind die Minima und Maxima des aktuellen Wertebereichs, sowie alle Informationen über alle eingelesenen Länder und die Bezeichnung der Kennzahl gespeichert. Sämtliche Informationen über ein `Land` werden in der gleichnamigen Klasse gespeichert. Diese sind jeweils das Autokennzeichen des Landes, die Nachbarlaender, welche in einer Liste abgelegt werden, der Radius des Kreises, der sich aus der ebenfalls gespeicherten Kennzahl ergibt und die Summe aller wirkenden Anziehungs- und Abstossungskräfte, die jeweils während einer Iteration gespeichert werden. Der Kern des Programms, befindet sich in der Klasse `Berechnung` bzw. `Standardberechnung`. Hier wird in der Methode `berechne`, die den Hauptalgorithmus implementiert die „eigentliche“ Berechnung der neuen Kreismittelpunkte durchgeführt. Der Hauptalgorithmus greift zusätzlich auf die Hilfsmethoden `voriteration`, `normVektor` und `distance` zu.

2.3 Ablauf Eingabe

Das Einlesen der Eingabedatei läuft wie folgt ab:
Zuerst wir die Eingabedatei zum Lesen geöffnet. Zeilenweise wird nun der Inhalt der Datei eingelesen. Das Format der Datei muss dabei dem vorgegebenen Format entsprechen (vgl. Punkt 1.1 Eingabedatei). Über eine Zählvariable wird bestimmt, welche Informationen als nächstes eingelesen werden. Zu Beginn hat diese Zählvariable den Wert 0. Wenn die Variable 0 ist, muss die nächste Zeile die erste Zeile der Datei sein. Diese enthält die Bezeichnung der Kenngrösse und wird in eine temporäre Variable gespeichert und anschliessend erhöht sich der Zähler um 1. Bei jedem Kommentar wird der Zäehler ebenfalls erhöht. Da sich zwischen der ersten Zeile und dem Einlesen der Länder ein Kommentar befindet, hat der Zähler beim Einlesen der Länder den Wert 2. Wenn also der Zähler gleich 2 ist, werden die Länder nacheinander eingelesen. Die Zeile wird gesplittet, sodass in einer Liste alle Informationen (Kennzeichen, Kennzahl, Längen- und Breitengrad) einzeln vorliegen. Hier wird nun überprüft, ob die angegebenen Werte korrekt sind, also ob die Kennzahl grösser als 1 ist, Längen- und Breitengrade im entsprechenden Intervall liegen usw. Wenn nicht, wird ein Fehler ausgelöst, der das Programm mit einer Meldung an den Benutzer beendet. Sind alle Werte korrekt, wird ein neues Objekt der Klasse `Land` erzeugt, das dann in einer Länderliste zwischengespeichert wird. Dies läuft für alle Länder gleich ab. Nach den Ländern folgt ein Kommentar, der den Zähler auf 3 erhöht. Das bedeutet, dass als nächstes die Länderbeziehungen eingelesen werden. Hier wird für jede Zeile am Doppelpunkt aufgeteilt zwischen Ausgangsland und dessen Nachbarn. Im Anschluss werden dem Ausgangsland seine Nachbarn zugeordnet und gleichzeitig jedem Nachbarn das Ausgangsland als Nachbar zugeordnet.
Aus der Länderliste und der Bezeichnung der Kenngrösse wird ein Objekt der Klasse Karte

erzeugt. Dieses wird zurückgegeben.

Methode read() aus Klasse InputReader

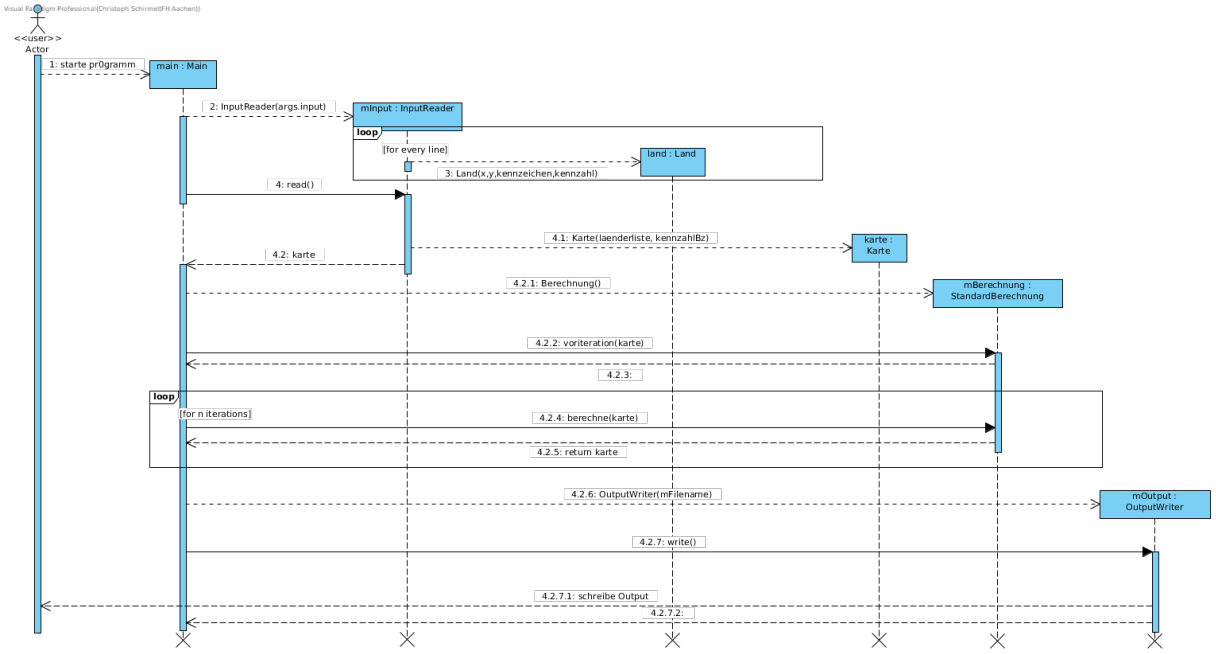


2.4 Ablauf Ausgabe

2.5 Ablauf Hauptalgorithmus

2.5.1 TODO

2.6 Sequenzdiagramm



3 Änderungen zum Montagsteil

3.1 Änderungen in der Klassenstruktur

3.2 Änderungen beim Algorithmus

4 Zusammenfassung und Ausblick

Zusammenfassung: Läuft.
Ausblick-Ideen: GUI, Parallelisierung, weitere Features

5 Benutzeranleitung

5.1 Ordnerstruktur

Die .zip Datei beinhaltet die Sourcen mit der Main.py als Hauptklasse zum Ausführen, ein Testskript und diese Dokumentation. Außerdem sind die Ordner „docu“ und „tests“ enthalten. Im „docu“-Ordner befindet sich ein PDF-Dokument mit der API, die mit epydoc generiert wurde. In dem tests Ordner befinden sich die Testfälle.
Ordnerstruktur:

- LandkartenDoku.pdf (dieses Dokument)
- readme.md (Eine Benutzeranleitung im Markdown Format)
- run_all_tests.py (Das Skript zum Ausführen der Tests)
- Main.py (Die Hauptklasse)
- Land.py
- Karte.py
- Berechnung.py
- InputReader.py
- OutputWrite.py
- \doc
- \tests

5.2 Voraussetzungen

Zum Ausführen wird Python benötigt. Das Programm wurde sowohl mit Version 2.7 als auch 3.7.3 getestet. Python ist frei benutzbar und läuft unter der *Python-Software-Foundation-Lizenz*, die kompatibel mit der GNU General Public License ist.
Des weiteren muss zum Ausführen die unter der BSD-Lizenz veröffentlichte Bibliothek NumPy installiert sein. Falls diese nicht installiert ist, kann sie über die Kommandozeile einfach installiert werden:
`pip install numpy`

Zum Erstellen der Grafiken kann optional das Programm Gnuplot installiert werden.

5.3 Laufzeitumgebung

Die Software wurde auf Windows 10 Education und Ubuntu Bionic Beaver 18.04 LTS auf 64-bit entwickelt. Als Entwicklungsumgebung wurde *Visual Studio Code Version 1.33* mit der mitgelieferten Python Extension benutzt.

5.4 Installation

Eine Installation ist nicht notwendig, weil die benutzte Sprache Python eine Interpretersprache ist. Die Dateien werden zur Laufzeit vom Interpreter in ausführbaren Code übersetzt.

5.5 Ausführen des Programms

5.5.1 Direkt

Für den direkten Aufruf kann folgender Befehl verwendet werden:
`python Main.py -i INPUT [-n ITERATIONEN | -s Skalierungsfaktor | -o OUTPUT]`

5.5.2 Mit dem Skript

Mit dem Skript `run_all_tests.py` können alle in einem Ordner liegenden Tests ausgeführt werden. Als erstes Argument wird der Pfad zur Datei `Main.py` erwartet. Das zweite Argument ist

optional der Pfad zu einem Ordner mit Inputdateien. Falls der angegebene Pfad nicht erreicht werden kann, wird stattdessen der lokale Ordner `tests` verwendet. Das Skript führt die Berechnung mit den Standardparametern aus, die i.d.R. ein passables Testergebnis liefern.
Windows: `python .\ run_all_tests.py .\Main.py`
Linux: `python ./run_all_tests.py ./Main.py`

5.6 Restriktionen beim Dateinamen

Die Datei muss dem benutzten Betriebssystem entsprechend einen gültigen Dateinamen besitzen. Sie kann sowohl über den relativen Pfad ausgehend vom Ordner des Aufrufs als auch über den absoluten Pfad angegeben werden. Die Pfade müssen auch dem Format des benutzten Betriebssystem entsprechen. Beispiele für Dateipfade: Absoluter Pfad:

`Windows: python „C:\Users\User\Documents\gropro\Main.py“`
`Linux: python /home/user/documents/gropro/Main.py`
Relativer Pfad:
`Windows: python .\Main.py`
`Linux: python ./Main.py`

5.7 Anmerkung

6 Tests

6.1 IHK Tests

6.1.1 IHK Example 01 etc.

6.2 Funktionierende Tests

6.2.1 F01 bliblablub etc.

6.3 Syntax-Fehler Tests

6.3.1 S01 lirumlarum

6.4 Logik Fehler Tests

6.4.1 L01 loremipsum

6.5 Optional: Code Coverage Tests, Performance o.ae

7 Source-Code

Land.py

```python
# -*- coding: utf-8 -*-
import math
class Land:
    xPos = 0.0
    yPos = 0.0
    kennzeichen = ""
    kennzahl = 0
    kraftX = 0.0
    kraftY = 0.0
    radius = 0.0
    nachbarlaender = []

    def __init__(self, x, y, kennZe, kennZa):
        self.xPos = x
        self.yPos = y
        self.kennzeichen = kennZe
        self.kennzahl = kennZa
        self.radius = math.sqrt(float(self.kennzahl) / (math.pi))

    def __str__(self):
        return str(self.xPos) + " " + str(self.yPos) + " " + str(self.radius) + " " + self.kennzeichen

    def __repr__(self):
        return str(self.xPos) + " " + str(self.yPos) + " " + str(self.radius) + " " + self.kennzeichen

    def addNachbar(self, nachbar):
        self.nachbarlaender.append(nachbar)

    def containsNachbar(self, nachbar):
        return nachbar in self.nachbarlaender
```

OutputWrite.py

```python
import re
from Karte import Karte

class OutputWriter:
    mFilename = ""
    mTemplate = ""
    mKarte = Karte([], "")
    mIterationen = 0

    def __init__(self, iFilename, iTemplate, iKarte, iIterationen):
        self.mFilename = iFilename
        self.mTemplate = iTemplate
        self.mKarte = iKarte
        self.mIterationen = iIterationen

    def write(self):
        # Read contents from file as a single string
        file_handle = open(self.mTemplate, 'r')
        file_string = file_handle.read()
        file_handle.close()
        # Use RE package to allow for replacement (also allowing for (multiline) REGEX)
        file_string = (re.sub("<xmin>", str(self.mKarte.xmin) , file_string))
        file_string = (re.sub("<xmax>", str(self.mKarte.xmax) , file_string))
        file_string = (re.sub("<ymin>", str(self.mKarte.ymin) , file_string))
        file_string = (re.sub("<ymax>", str(self.mKarte.ymax) , file_string))
        file_string = (re.sub("<Name des Kennwertes>", self.mKarte.kennzahlBz[:-1], file_string))
        file_string = (re.sub("<nr>", str(self.mIterationen) , file_string))
        s = ""
        i = 0
        for land in self.mKarte.laenderliste:
            s = s + str(land) + " " + str(i) + "\n"
            i = i + 1
        s = s[:-1]
        file_string = (re.sub("<Liste aus <xpos> <ypos> <radius> <autokennzeichen> <id> >", s , file_string))
        # Write contents to file.
        # Using mode 'w' truncates the file.
        file_handle = open(self.mFilename + ".gpl", 'w')
        file_handle.write(file_string)
        file_handle.close()
```

Karte.py

```python
class Karte:
    laenderliste = []
    kennzahlBz = ""
    xmin = 0.0
    xmax = 0.0
    ymin = 0.0
    ymax = 0.0

    def __init__(self, l, kennzBz):
        self.laenderliste = []
        for land in l:
            self.laenderliste.append(land)
        self.kennzahlBz = kennzBz
```