Entwicklung eines Softwaresystems 2019

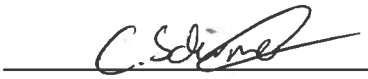
Landkartenerstellung

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

Inhaltsverzeichnis

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.

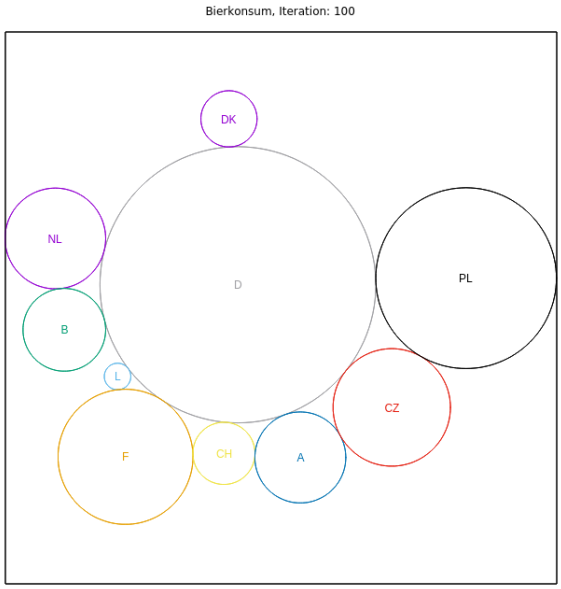


Christoph Schirmel, Stolberg den 17.05.2019

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:

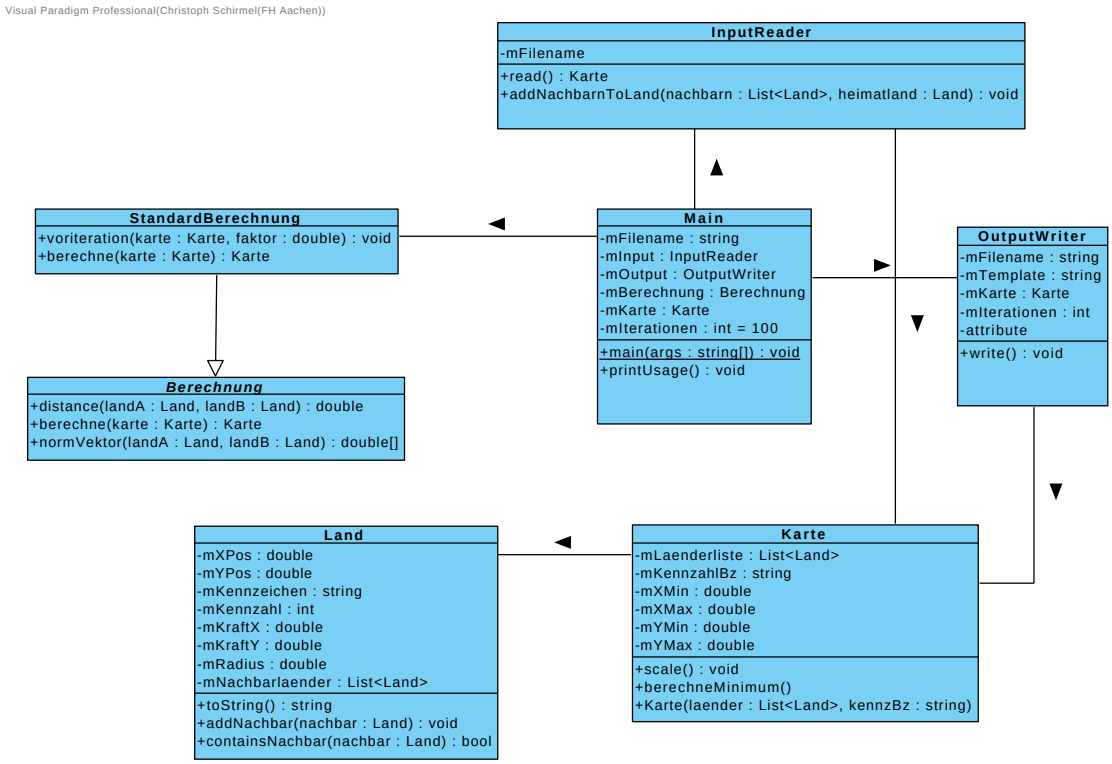
- Doppelt vorkommende Autokennzeichen sind nicht erlaubt, da diese eindeutig sein müssen, um einen Staat eindeutig identifizieren zu können.

- 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

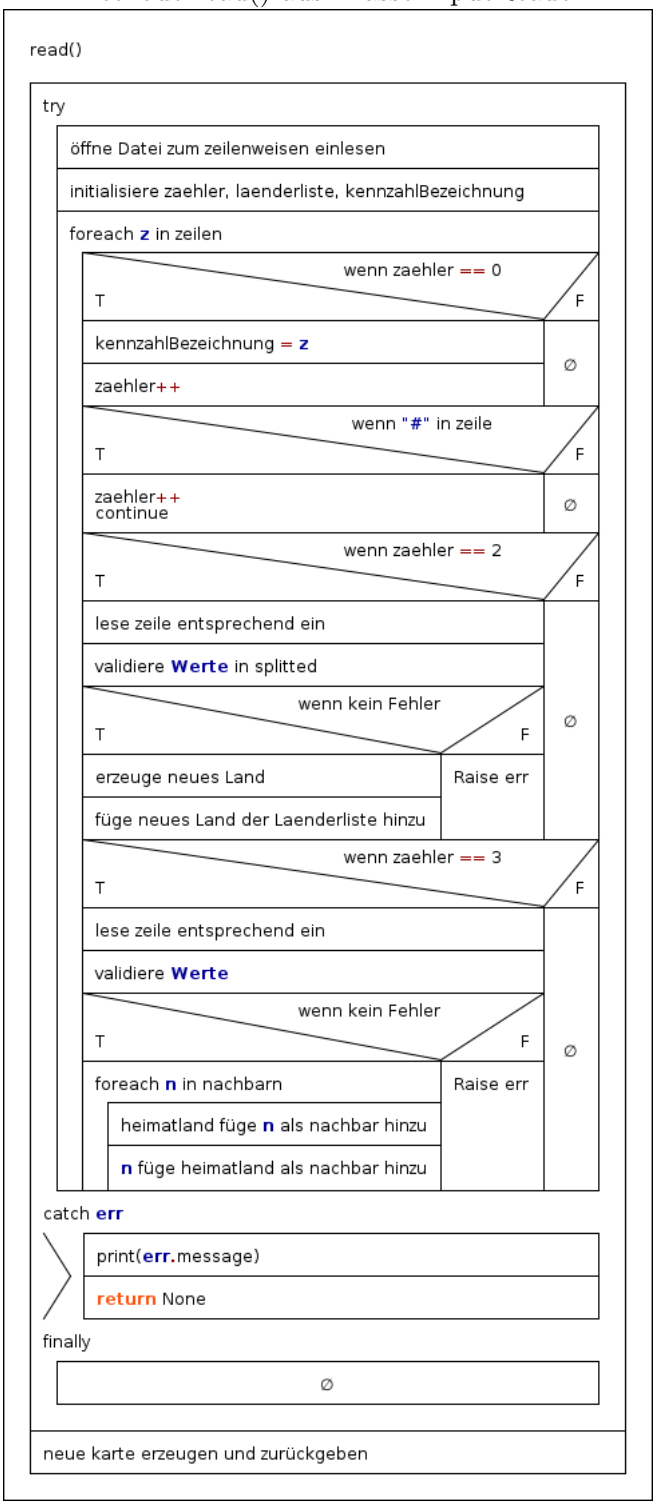
Die Hauptklasse des Programms ist die Klasse `Main`, die den Einstiegspunkt des Programms markiert und alle anderen Vorgänge steuert. Hierfür hat sie über ihre Attribute Zugriff auf die Funktionen für Ein- und Ausgabe, die Berechnung und die zurzeit eingelesenen Daten. Auch die Anzahl der Iterationen, 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

Für die Ausgabe wird ein Template eingelesen, das der allgemeinen Form der Aufgabenstellung (Siehe Punkt 1.2 Ausgabedatei) entspricht eingelesen. Alle Tags werden durch die entsprechenden Werte ersetzt, wobei für den Tag <Liste aus ...> ein String erstellt wird, der alle Länder

enthält. Zuletzt wird die Datei unter dem neuen Namen abgespeichert. Dieser entspricht dem Eingabenamen und der Endung ".gplöder einem per Parameter frei gewählten Namen.

```
write()
    öffne die Datei mTemplate als String
    ersetze die Tags <xmin>, <xmax>, <ymin>, <ymax>, <Name des Kennwertes>, <nr> durch die tatsächlichen Werte
    s = ""
    for i ← 0 to len(karte.laenderliste)
        land = karte.laenderliste[i]
        füge <xpos> <ypos> <radius> <kennzeichen>, i und zeilenumbruch an s an
    ersetze <Liste aus ...> durch s
    schreibe in die datei mit neuem Namen mFilename
```

2.5 Ablauf Hauptalgorithmus

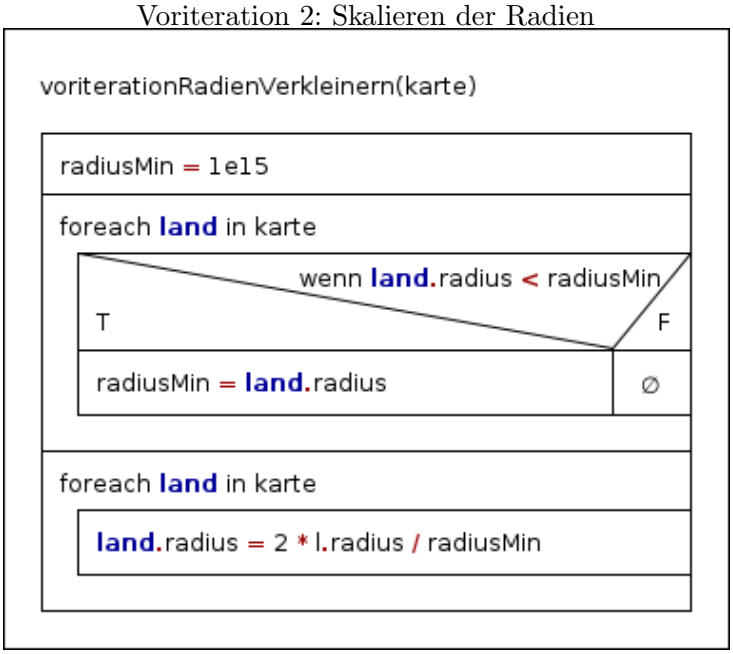
2.5.1 Voriteration

Vor der ersten Iteration wird eine Voriteration durchgeführt, um das Endergebnis zu verbessern. Bei der Voriteration soll der Abstand zwischen den Ländern vergrössert werden, ohne das Verhältnis der Länder zu beeinflussen. Hierfür habe ich mir zwei Ansätze überlegt:
Der erste Ansatz vergrössert die Abstände, indem alle Mittelpunkte um einen Faktor x skaliert werden. Der Faktor ist grundsätzlich erstmal eine positive reelle Zahl grösser bzw. gleich 1 (falls keine Voriteration gewollt ist). Die hierdurch erzielten Ergebnisse waren insbesondere beim dritten Beispiel der IHK besser, führten allerdings je nach gewähltem Faktor auch zu zu grossen Zahlen und dadurch zu einem unbrauchbaren Ergebnis. Daher habe ich mir einen anderen Ansatz überlegt, der auch besser funktioniert, nämlich die Verkleinerung der Radien. Hierfür habe ich alle Radien zuerst mit einem Faktor x>1 (default = 2) multipliziert und anschliessend durch den kleinsten Radius geteilt.

Voriteration 1: Skalieren der Kreismittelpunkte

```
voriterationHochskalieren(karte, faktor)
    foreach land in karte
        land.xPosition = land.xPosition * faktor
        land.yPosition = land.yPosition * faktor
```

Voriteration 2: Skalieren der Radien



2.5.2 Hauptalgorithmus

Der Hauptalgorithmus zur Bestimmung der Abstossungs- und Anziehungskräfte ist wie in der Aufgabenstellung angegeben ein Iterationsverfahren. Es bekommt die aktuelle geographische Lage als Karte übergeben. Dabei handelt es sich entweder um die Ausgangslage nach der Voriteration oder aber um das Ergebnis der letzten Iteration. Der Hauptalgorithmus wird so oft ausgeführt wie Iterationen N (default=100) angegeben werden. Der Hauptalgorithmus berechnet für jeden Staat dessen Anziehungs- und Abstossungskräfte zu allen Nachbarstaaten bzw. zu Staaten, die von ihm geschnitten werden. Dabei gilt für die Kraft F von einem Staat A zu einem Staat B folgende Formel:

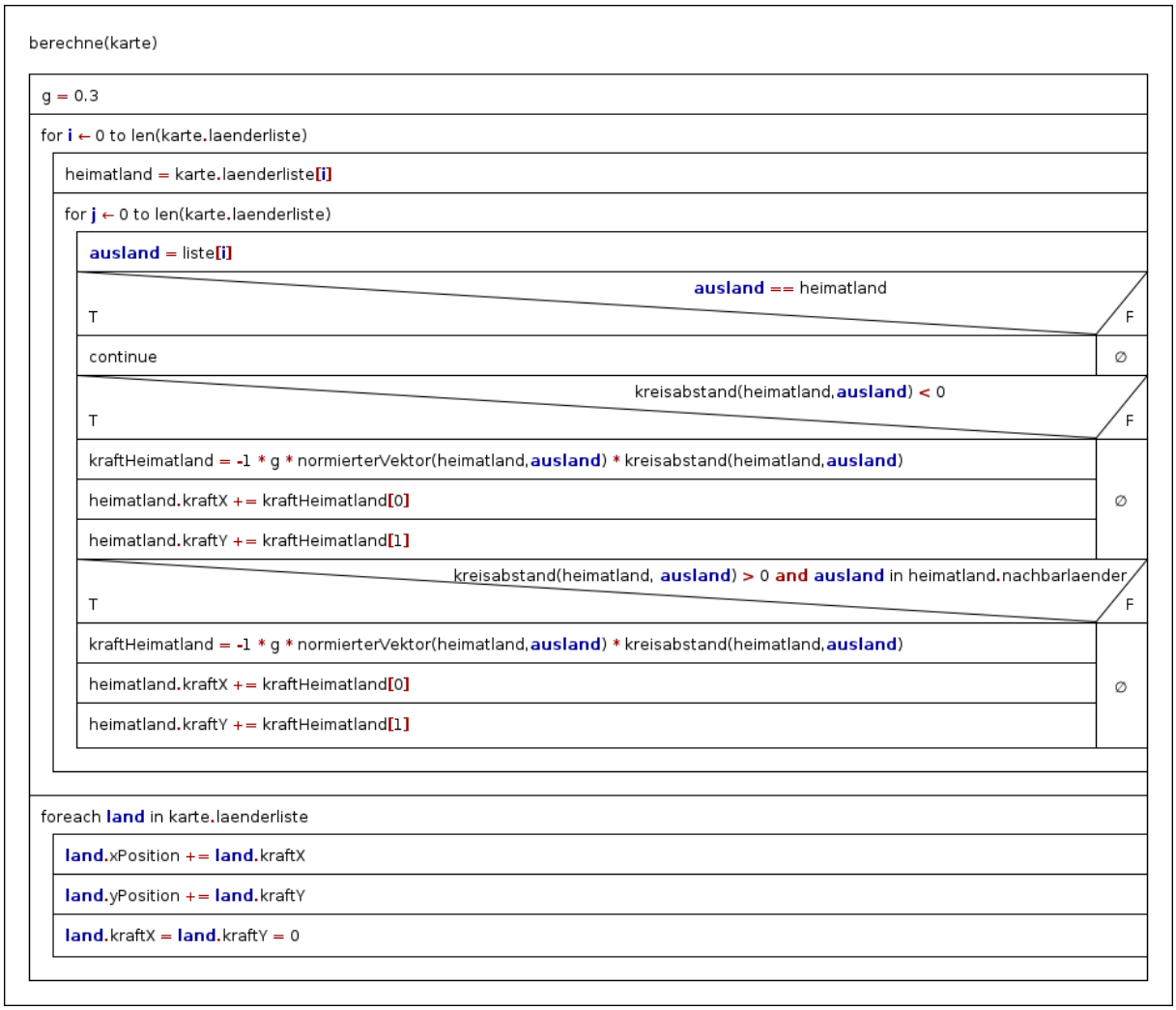
$$F = \frac{|r_A + r_B + |\vec{d_{BA}}||}{g * |\vec{d_{AB}}|} * \vec{d_{BA}}$$

Bei g handelt es sich um eine Konstante, die die Kräfte anteilig auf die beiden Staaten verteilt, sodass nicht beide die volle Kraft aufaddiert bekommen und nicht einer allein die ganze Kraft. In der Aufgabenstellung wurde vorgeschlagen die Kraft hälftig zu verteilen. Ich habe im Praxiseinsatz festgestellt, dass meine Implementierung besser funktioniert, wenn g<0.5 ist und habe einen Standardwert von 0.3 festgelegt. Die berechneten Kräfte werden auf den Kraftvektor des jeweiligen Landes addiert, weil für die Gesamtkraft, die auf ein Land A wirkt, gilt:

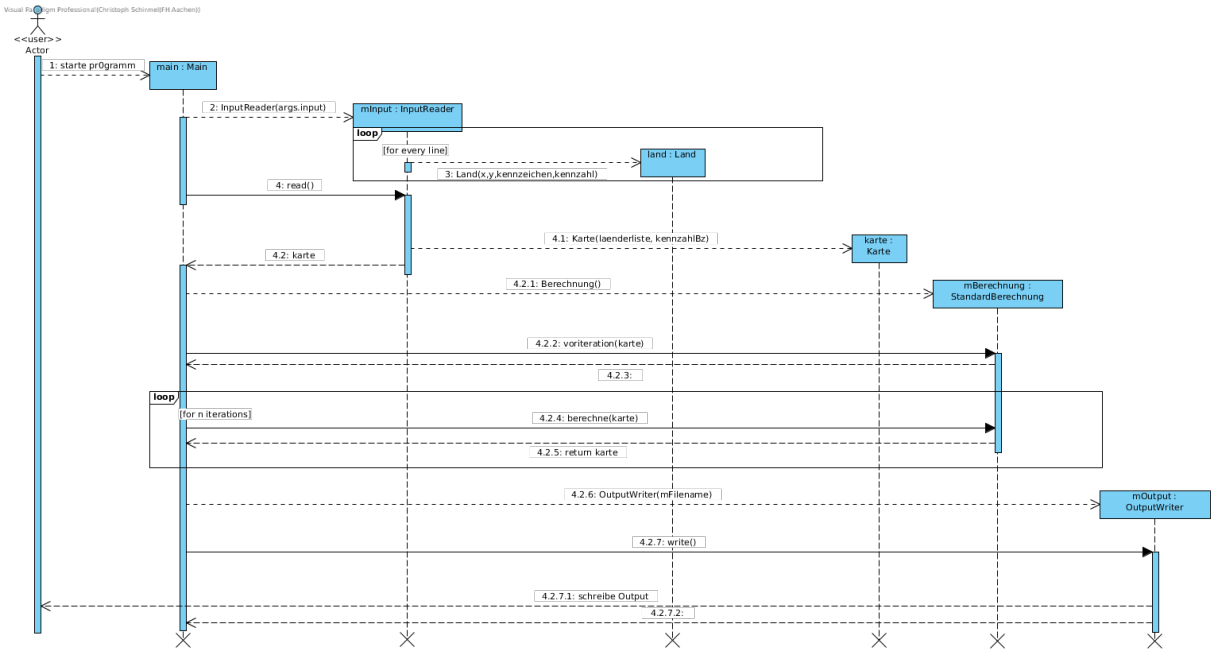
$$F_A = \Sigma_{i=1}^{n} F_i$$

wobei F_i alle auf das Land A wirkende Kräfte sind.

Nach dem Berechnen werden alle Kräfte auf die Mittelpunkte der Kreise angewandt und so die „neuen“ Mittelpunkte berechnet. Der gesamte Ablauf wird im folgenden in einem Nassi-Shneiderman-Diagramm dargestellt:



2.6 Sequenzdiagramm



3 Änderungen zum Montagsteil

3.1 Änderungen in der Klassenstruktur

Die Klassenstruktur wurde auf die Programmiersprache angepasst, ist aber grundsätzlich erhalten geblieben. Aus dem Interface Berechnung habe ich eine abstrakte Klasse gemacht, um sinnvolle Methoden abstrahieren zu können. Da ich in Python gearbeitet habe, waren Getter und Setter nicht mehr erforderlich, da alle Variablen, die ausserhalb von Funktionen definiert sind global sind. Im Klassendiagramm wurden in der Klasse `Land` die beiden Membervariablen `radius` und `nachbarlaender` eingefügt. Ausserdem habe ich die folgenden Methoden ergänzt:
Klasse InputReader:

- `zusammenhaengend(laenderliste)` - Wrapper um Rekusionsaufruf
- `zusammenhaengendRek(startland, besucht)` - rekursive Methode, die ermittelt, ob alle Länder zusammenhängen
- `addNachbarnToLand` - fügt einem Land einen Nachbar hinzu.

Klasse Karte:

- `scale()` - Erfüllt die Bedingung ymax-ymin = xmax-xmin
- `berechneMinimum()` - Berechnet die Minima xmin, ymin und Maxima xmax, ymax

Klasse Land:

- `__str__()` - Stringdarstellung eines Landes
- `__repr__()` - Stringdarstellung eines Landes (für debugging)
- `addNachbar(nachbar)` - Fügt einen Nachbar hinzu
- `containsNachbar(nachbar)` - Gibt true zurück wenn nachbar Nachbar ist

Klasse *Berechnung:*

- `distance(landA, landB)` - Berechnet den Kreisabstand zwischen zwei Ländern
- `normVektor(landA, landB)` - Gibt den normierten Verbindungsvektor AB zurück

Klasse Standardberechnung:

- `mittelwertRadien(karte)` - Gibt den Mittelwert der Radien zurück
- `voriteration` - führt eine Voriteration durch

3.2 Änderungen beim Algorithmus

An meinem Hauptalgorithmus habe ich im Vergleich zu Montag so verändert, dass die Überprüfung weggefallen ist, ob eine Kraft bereits hälftig auf ein Land angewandt wurde oder nicht. Dies habe ich dadurch ersetzt, dass nun über alle Länder iteriert wird und von vornherein nur die anteilige Kraft angewandt wird. Die beiden möglichen Voriterationen wurden nach Montag ebenfalls eingefügt. Die Kraft wird anteilig mit einem Faktor von 0.3 auf beide Mittelpunkte angewandt. Dies verbesserte das Ergebnis im Vergleich zu 0.5.

4 Zusammenfassung und Ausblick

Das Programm funktioniert wie in der Aufgabenstellung vorgegeben: Es liest die syntaktisch korrekte Datei ein, prüft auf semantische Fehler wie z.B. ein doppeltes Land, berechnet die geforderte Anzahl Iterationen und schreibt das Ergebnis in eine Ausgabedatei. Intern werden durch einen objektorientierten Ansatz Ein-/Ausgabe und Funktionalität sauber voneinander getrennt und über eine Abstrahierung der Klasse Berechnung ist es leicht möglich weitere Klassen zur Berechnung abzuleiten.
Das Programm liesse sich sehr gut mit einer grafischen Benutzeroberfläche erweitern. Diese könnte unter anderem einen Plotter beinhalten, der die berechneten Daten direkt anzeigt. Dies würde den bisher notwendigen Zwischenschritt einsparen, nach der Berechnung durch mein Programm noch Gnuplot aufrufen zu müssen, um den Output zu visualisieren. Der Plotter könnte zusätzlich die Funktion beinhalten, die Landkarte zu exportieren oder noch manuelle

Änderungen durchzuführen. Performance-technisch lässt sich das Programm ebenfalls noch verbessern. Dies könnte über ein effizienteres Iterationsverfahren oder möglicherweise auch durch Parallelisierung realisiert werden. Allerdings wäre letzteres erst ab einer hohen Zahl an Iterationen bzw. bei einer deutlich höheren Anzahl Staaten erforderlich, damit das Programm nicht ungewollt durch Overhead und Synchronisierung beim parallelen Aufbau verlangsamt wird.

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 den Betriebssystemen Windows 10 Education, OpenSUSE 13.1 und Ubuntu Bionic Beaver 18.04 LTS auf 64-bit getestet und 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 <folder>`
Linux: `python ./run_all_tests.py ./Main.py <folder>`

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

In diesem Kapitel werden alle Tests behandelt. Da die Ergebnisse je nach Anzahl der Iterationen und mit anderen Skalierungsfaktoren anders ausfallen können, werden folgende Parameter grundsätzlich für alle Tests angenommen:

- Iterationen: 100
- Skalierungsfaktor Mittelpunkte: $\delta = 4$
- Skalierungsfaktor Radien: $\phi = 2/rMinundrMin = kleinsterRadius$

6.1 Äquivalenzklassen

Im folgenden sind die gültigen und ungültigen Äquivalenzklassen dargestellt, auf denen die Tests aufbauen:

Wert	Gültige Äquivalenzklassen	Ungültige Äquivalenzklassen
Längengrad	1) Längengrad ist in (-180;180]	2) Längengrad ist in (180;inf) 3) Längengrad ist in (-inf;-180]
Breitengrad	4) Breitengrad ist in [-90;90]	5) Breitengrad ist in (-inf;-90) 6) Breitengrad ist in (90;inf)
Kennzeichen	7) nur verschiedene Kennzeichen bestehend aus max. Großbuchstaben	8) doppelte Kennzeichen 9) Kennzeichen zu lang 10) Kennzeichen enthält ungültige Zeichen
(Längengrad, Breitengrad)	11) nur verschiedene Positionen	12) doppelte Positionen
Nachbarschaften	15) doppelte Länderbeziehungen	16) Nachbarschaften von nicht definierten Ländern 17) Land ohne Nachbarschaftsbeziehung
Kennwert	18) Kennwert ist in [1;maxInt-1]	19) Kennwert ist in (-inf;0] oder [maxInt;inf)

6.2 IHK Tests

6.2.1 IHK_Beispiel_01

Eingabedatei:

```
Flaeche der Staaten
# Staat Fläche Längengrad Breitengrad
D 357 10.0 51.3
NL 42 5.3 52.2
B 33 4.8 50.7
L 3 6.1 49.8
F 544 2.8 47.4
CH 41 8.2 46.9
A 84 14.2 47.6
CZ 79 15.3 49.8
PL 313 18.9 52.2
DK 43 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
```

Erwartete Ausgabe:

```
reset
set xrange[700.2517724484413:827.7002402768413]
set yrange[856.8647214556818:984.3131892840818]
set size ratio 1.0
set title "Flaeche der Staaten, Iteration: 100"
unset xtics
unset ytics
$data << EOD
766.7834981016138 930.3185429627034 21.81742422927143 D 0
742.555557198933 946.8167634812557 7.4833147735478835 NL 1
738.4945723703775 933.3089756697666 6.6332495807108 B 2
743.4807756786114 925.5870070124199 2.0 L 3
727.1837856174069 901.7272715950642 26.93201316896554 F 4
761.4598380217082 901.6197768434589 7.393691004272945 CH 5
779.3567011368073 900.4377222425624 10.583005244258363 A 6
794.7022000752384 914.5003211738591 10.263202878893768 CZ 7
807.271502348782 942.4728444996451 20.42873792805942 PL 8
759.5115694505225 958.8107745192644 7.571877794400365 DK 9
EOD
plot \
'$data' using 1:2:3:5 with circles lc var notitle, \
'$data' using 1:2:4:5 with labels font ärial,9"tc variable notitle
```

Erklärung:
Das erste IHK Beispiel. Die erwarteten Werte der Ausgabe entsprechen jedoch den Werten, die man für die Ausgabe meines Programms erwartet. Das Beispiel sollte funktionieren.

6.2.2 IHK_Beispiel_02

Eingabedatei:

```
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
```

Erwartete Ausgabe:

```
reset
set xrange[730.0918468719356:813.7048298686099]
set yrange[889.3127574556308:972.9257404523053]
set size ratio 1.0
set title "Bierkonsum, Iteration: 100"
unset xtics
unset ytics
$data << EOD
765.5758599792373 934.3733238909471 20.847062143141414 D 0
737.6944779954348 940.0307359422596 7.602631123499284 NL 1
739.702532976425 926.3254082357182 6.248999919987197 B 2
746.705927195248 921.4915070307936 2.0 L 3
747.2660866575683 909.3129805429031 10.190682018393076 F 4
762.1460577992617 909.0668195802234 4.690415759823429 CH 5
773.6474042905288 907.8537821312843 6.873863542433759 A 6
787.8317016409018 914.6836660759692 8.868483523128404 CZ 7
800.0593177858773 933.5882178735943 13.64551208273255 PL 8
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
```

Erklärung: Das zweite IHK Beispiel. Die erwarteten Werte der Ausgabe entsprechen jedoch den Werten, die man für die Ausgabe meines Programms erwartet. Das Beispiel sollte funktionieren.

6.2.3 IHK_Beispiel_03

Erwartete Ausgabe:

```
reset
set xrange[668.4407470958473:848.2649929297002]
set yrange[818.4754105753512:998.2996564092041]
set size ratio 1.0
set title "Flaeche der Staaten, Iteration: 100"
unset xtics
unset ytics
$data << EOD
759.9117024899185 946.1176442974413 21.81742422927143 D 0
733.0942744867059 957.8497368903509 7.4833147735478835 NL 1
731.840011226166 943.9370995079238 6.6332495807108 B 2
737.8400401507139 937.5561516306295 2.0 L 3
725.9243677924528 911.317812659253 26.93201316896554 F 4
761.0075586818753 916.6965557180389 7.393691004272945 CH 5
780.0538973319441 918.0048301407443 10.583005244258363 A 6
786.1546923528323 935.4322086939553 10.263202878893768 CZ 7
802.1041118876469 958.2429172958114 20.42873792805942 PL 8
752.1854360464978 974.4605381891093 7.571877794400365 DK 9
710.2859001757952 860.7169973194165 25.974346318370873 E 10
679.5162455797381 881.3404821943074 11.075498483890765 P 11
766.3352407050984 889.6593175574536 20.03330560175563 I 12
786.5257039557389 903.8961899441882 5.163977794943223 SLO 13
799.464057507478 927.9783749617561 8.082903768654761 SK 14
801.1512865930757 909.9497349607547 11.135528725660043 H 15
793.9873976605014 892.3417594347463 8.717797887081348 HR 16
796.7929994273657 876.4212667424503 8.24621125123532 BIH 17
812.9140750583823 891.1920093802989 10.832051206181282 SRB 18
805.982036441863 880.6881277561976 4.320493798938574 MNE 19
```

```
821.1323977915398 878.0028753209402 5.887840577551898 MK 20
811.3382172011133 872.7929306319606 6.21825270205921 AL 21
829.221989968858 914.5040528201025 17.435595774162696 RO 22
836.0994678691037 885.7968793173857 12.16552506059644 BG 23
827.5368916175953 861.1035066347839 13.2664991614216 GR 24
EOD plot \
'$data' using 1:2:3:5 with circles lc var notitle, \
'$data' using 1:2:4:5 with labels font ärial,9"tc variable notitle
```

Erklärung:
Das dritte IHK Beispiel. Die erwarteten Werte der Ausgabe entsprechen jedoch den Werten, die man für die Ausgabe meines Programms erwartet. Das Beispiel funktioniert je nach Skalierung besser oder schlechter. Hier ist eine gute Voriteration definitiv erforderlich.

6.3 Funktionierende Tests

6.3.1 F01 bliblablub etc.

6.4 Syntax-Fehler Tests

6.4.1 S01 lirumlarum

6.5 Logik Fehler Tests

6.5.1 L01 loremipsum

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
```