Benutzerhandbuch
Simulatoren für Wilde Raumzeit

Betti Österholz

Betti_Oesterholz@gmx.de

www.BioKom.info

Potsdam, 3. Mai 2011

Inhaltsverzeichnis

1 Einleitung

Zu meinem Universum „Wilde Raumzeit“, dessen Dokumentation ([1]) unter `www.BioKom.info/Projekt_Wilde_Raumzeit.html` zu finden ist, habe ich auch drei Simulatoren (zu finden auf der gleichen Internetseite) geschrieben. Die vorliegende Arbeit ist als Dokumentation bzw. Benutzerhandbuch für diese Simulatoren zu sehen.

Die Simulatoren berechnet dabei nicht das Universum, sondern gültige Folgen von Raumuniversen. Sie unterscheiden sich nur in den Wahrscheinlichkeiten mit der eine bestimmte Raumuniversumsfolge berechnet wird und der Zeit die Berechnung benötigt. Der Unterschied kommt durch die unterschiedlichen Wahlen der Transition zustande.

Der Simulator univers wählt die Transitionen gleichwahrscheinlich aus der Menge der möglichen Transitionen aus und ist der wohl langsamste der drei Simulatoren.

Beim Simulator universD liegt der Wahl der Transitionen die Anzahl der ausgehenden Distanzen eines Punktes (plus Eins) zugrunde.

Im Simulator universP wird zuerst ein Punkt und dann zwei Distanzen (plus Eins) dieses Punktes gleichwahrscheinlich ausgewählt. Dieser Simulator ist wohl der schnellste der drei Simulatoren, dafür sind die Wahrscheinlichkeiten, mit der Raumuniversumsfolgen erzeugt werden, am weitesten von dem Universum „Wilde Raumzeit“ entfernt.

In diesem Benutzerhandbuch werden alle drei Simulatoren gleichzeitig behandelt, da ihre Funktionsweise sich nur in unterschiedlichen Wahlen der Transition unterscheidet. Nur wo dieser Unterschied zum Tragen kommt, werden die drei Simulatoren separat besprochen.

Diese Arbeit ist in zwei Teile untergliedert:

Im ersten Teil werden die Grundlagen und Parameter der Simulatoren beschrieben. Er dient Anwendern die nur die vorhandene Funktionalität der Simulatoren nutzen wollen ohne diese zu erweitern.

Im zweiten Teil wird die Implementierung der Simulatoren beschrieben. Er soll Anwendern die Erweiterung des Simulators erleichtern.

Die Simulatoren sind für erfahrene Anwender gedacht die sich mit der Theorie „Wilde Raumzeit“ auskennen. Daher ist der Umgang mit den Simulatoren nicht so einfach, wie z.B. mit normalen Windowsprogrammen. Für Jemanden der die Grundlagen der Theorie nicht kennt, haben die Simulatoren wohl auch keinen Verwendungszweck.

Des weiteren sind die Simulatoren eine reine Konsolenanwendungen.

Die Simulatoren unterliegt zur Zeit auch einem Weiterentwicklungsprozess, so dass nicht garantiert werden kann, dass diese Dokumentation auf dem aktuellen Stand der Dinge ist.

Im Nachfolgendem werden Quelltextabschnitte im `Schreibmaschinenstil` geschrieben.

2 Installation

Für die Simulatoren gibt es keine direkte Installation. Das erhalten einer ausführbaren Datei für die Simulatoren ist bei allen drei gleich. Die Dateien ausführbaren Dateien für die Simulatoren unterscheidet sich nur in ihren Namen (genauer: in der Endung des Namens ”,’D’ und ’P’). Alle Versionen sind in C++ geschrieben und liegen (auch) im Quelltext vor.

Bei den Linux Versionen muss der Quellcode kompiliert werden. Dazu speichern sie die Quellcode Dateien in ein Verzeichnis, öffnen eine Konsole, wechseln auf der Konsole in das Verzeichnis und führen „make“ aus (mittels „`./make`“).

Die entstandene Simulatordatei z.B. „univers“ kann nun mit „`./univers parameter`“ ausgeführt werden, wobei „parameter“ die Datei ist, in der die Parameter gespeichert sind. Wird keine Parameterdatei angegeben, so werden die Standartparametern aus der Datei „standart_Parameter.txt“ geladen.

Wenn der Simulator öfter verwendet werden soll, ist es sicherlich sinnvoll, den Pfad der Simulatordatei (z.B. „univers“) zum Standartsuchpfad hinzuzufügen. Dafür kann beispielsweise in der versteckten Datei „.bashrc“ im Stammverzeichnis des Benutzers die Zeile „`PATH=$PATH:/username/simulator/`“ hinzugefügt werden, wobei „/username/simulator/“ das Verzeichnis des Simulators ist.

Bei den Windows Versionen muss nur die ausführbare Datei heruntergeladen werden. Sie wurden von Linux portiert und sollten zumindest unter Windows XP laufen.

3 Schnellstart

Vor der ersten Ausführung eines Simulators ist die Simulatordatei, z.B. „univers“, zu erzeugen, siehe hierzu Abschnitt 2 . Im Nachfolgendem wird als angenommen das die Simulatordatei „univers“ heißt, wenn der Simulator „universD“ oder „universP“ verwendet wird, ist „univers“ entsprechend zu ersetzen.

Die Simulatoren sind alle reine Konsolenanwendungen. Gestartet wird der Simulator mit „`./univers parameter`“, wobei „parameter“ die Datei ist, in der die Parameter gespeichert sind. Wird keine Parameterdatei angegeben, so wird die Datei „standart_Parameter.txt“ mit den standart Parametern geladen. Eine Parameterdatei kann z.B. dadurch erstellt werden, indem die Parameterdatei „standart_Parameter.txt“ mit einem Texteditor geöffnet, entsprechend modifiziert und unter neuem Namen abgespeichert wird.

Was die einzelnen Parameter bedeuten wird im Nachfolgendem näher erläutert.

Mit der Eingabe von „E“ (End) oder „C“ (Chancel), beides als Großbuchstaben, kann die aktuelle Testreihe beendet werden. Dabei wird noch die aktuelle Iteration/Runde beendet. Die Eingaben führen nur zu unterschiedlichen Ergebnissen,

wenn die Testreihe fortgesetzt werden soll (siehe 4.4.6).

4 Simulatorbeschreibung

In den Nachfolgenden Unterabschnitten werden meist zuerst die für das angesprochene Thema relevanten Methoden, Daten, Klassen und/oder Parameter aufgeführt.

4.1 Raumuniversen

Methoden:
```
void start(unsigned long trans=0);
bool endCondition();
unsigned long getInternalTime();
list<TPoint*>* getUniversStructur();
TUPart getUnivers();
unsigned long getNumberOfPoints();
unsigned long getLastName();
unsigned long getNumberOfPDistance();
unsigned long getNumberOfNDistance()
```
Daten: `list<TPoint*> univers`
Klassen: `TPoint; TUPart`

Zentrales Objekt der Theorie „Wilde Raumzeit“ sind Raumuniversen. Die Simulatoren bilden aus einem Raumuniversum durch (eine) Transition(en) ein neues Raumuniversum.

Ein Raumuniversum wird repräsentiert durch eine List von Punkten des Raumuniversums (`list<TPoint*> univers`). Jeder dieser Punkte `TPoint` enthält einen Namen in Form einer ganzen Zahl (`unsigned long name`) und zwei Listen von Zeigern auf Punkte, die seine positiven (`list<TPoint*> pPoint`) und negativen (`list<TPoint*> nPoint`) Nachbarpunkten sind (der Punkt ist verbunden mit dem Nachbarpunkt durch je eine positiven bzw. negative Distanz pro Eintrag). Auf diese Weise kann einfach eine Transition ausgeführt werden, indem ein Punkt und von ihm zwei Nachbarpunkte (inklusive neuen Punkt) ausgewählt werden. Es kann leicht durch das Raumuniversum „gewandert“ werden, indem von einem Punkt über einem Zeiger aus seinen Nachbarpunktenlisten zu einem Nachbarpunkt gesprungen wird. Außerdem können die Punkte nach ihren Namen in der Ordnung der ganzen Zahlen eindeutig geordnet werden.

Im Programmsystem wird nur ein Raumuniversum (das aktuelle) im Arbeitsspeicher gehalten, um Arbeitsspeicher zu sparen. Dies ist sinnvoll, da die Raumuniversen schnell größer werden. Es kann aber dafür gesorgt werden, dass das jeweils aktuelle Raumuniversum auf Festplatte persistent gespeichert wird.

Die Funktion `getNumberOfPoints()` liefert die Anzahl der Punkte, die Funktion `getNumberOfPDistance()` die Anzahl der positiven Distanzen und `getNumberOfNDistance()` die Anzahl der negativen Distanzen im aktuellen

Raumuniversum. Mit der Funktion `getLastName()` wird der Name des letzten erzeugten Punktes zurückgegeben, bzw. der Name, der die höchste Nummer darstellt, da bei Transitionen entstehende Punkt nach der Ordnung der natürlichen Zahlen mit 1 beginnend benannt werden.

In der Klasse `TUPart` können die zu einem Raumuniversum gehörigen Daten abgespeichert werden. Dazu gehören die einzelnen Punkte (`TPoint`) des Raumuniversums plus die Anzahl der positiven, negativen Distanzen und der zuletzt vergebene Name im Raumuniversum.

Durch die Funktion `getUniversStructur()` wird ein Zeiger auf das aktuelle Raumuniversum (`list<TPoint*>*`) zurückgegeben. Jede Veränderung des zurückgegebenen Raumuniversums wirkt sich also direkt auf das aktuelle Raumuniversum aus. Mit der Funktion `getUnivers()` werden die Daten zu dem aktuellen Raumuniversum zurückgegeben. Dabei wirken sich nur Veränderungen in den Punkten der zurückgegebenen Struktur auf das aktuelle Raumuniversum aus. Wenn Beispielsweise nur neue Punkte im Raumuniversum der zurückgegebenen Struktur hinzugefügt werden, hat dies keinen Einfluss auf das aktuelle Raumuniversum.

4.2 Transitionen

Methoden:
```
void tick(unsigned long trans); void tick();
    void evalueTransition(TPoint* poi);
    void evalueTransition(TPoint* poi, unsigned long wich);
    bool endCondition();
    unsigned long getInternalTime();
    unsigned long* getNumberOfLastTransitions()
```

Die Simulatoren unterstützt natürlich nur die eine-Welt-Sicht. Es wird also eine Raumuniversenfolge berechnet. Das Berechnen aller Nachfolger nur eines Raumuniversums ist normalerweise viel zu aufwändig, da es nur für einen Punkt $n_a^2 = (n_p + n_n + 1)^2$ mögliche Transitionen gibt (n_p Anzahl der positiven und n_n Anzahl der negativen Nachbarn). Für die Wahl des nächsten Transitionspunktes gibt es jeweils zwei unterschiedlich schnelle (`tick()`) Methoden

Eine Transition wird durch die Methode `evalueTransition(poi)` bzw. die Methode `evalueTransition(poi, wich)` ausgeführt. Transitionenfolgen werden durch die `tick()` Methoden realisiert. In diesen Methoden liegt der Unterschied zwischen den drei Simulatoren.

Die Methode `tick()` ohne Parameter ist dabei jeweils die Standartmethode, die Methode `tick(trans)` ist nur von Interesse, wenn das Programmsystem abgeändert werden soll. Des weiteren ist die Art der Wahl der jeweiligen `tick(trans)` Methode näher an der Wahl der Theorie „Wilde Raumzeit“, ist aber bei der gleichen Transitionsanzahl langsamer, als die Wahl der jeweiligen Methode `tick()` ohne Parameter. In wie weit die jeweils unterschiedlichen Arten der Wahl beider `tick()` Methoden aber Einfluss auf die entstehenden Raumuniver-

senfolgen und Wahrscheinlichkeiten haben, ist ungewiss. Nur bei der `tick(trans)` Methode des Simulators „univers“, entspricht die Auswahlwahrscheinlichkeit einer Transition für das Raumuniversum der in der Theorie „Wilde Raumzeit“ geforderten.

Im Nachfolgenden werden die `evalueTransition()` und `tick()` Methoden der einzelnen Simulatoren beschrieben.

4.2.1 univers

In diesem Simulator ist jedem Punkt und dem ganzem Raumuniversum eine Zahl für die Zahl der möglichen Transitionen zugeordnet. Die Anzahl der möglichen Transitionen für einen Punkt ist dabei (wie in der Theorie) $n_a^2 = (n_p + n_n + 1)^2$, mit n_p Anzahl der positiven und n_n Anzahl der negativen Nachbarn. Die Anzahl der möglichen Transitionen für das Raumuniversum ist die Summe der Anzahlen der möglichen Transitionen aller Punkte des Raumuniversums.

Im Simulator „univers“ wird mit der `evalueTransition(poi, wich)` eine Transition ausgeführt. Ihr wird der Zeiger zu dem Punkt übergeben, welcher der Entstehungspunkt der Transition sein soll. Der der Methode übergebene Parameter `wich` ($0 \leq wich < (n_p + n_n + 1)^2$) bestimmt die beiden Nachbarpunkte für die Transition ($t(Raumuniversum, p_1, p_2, p_3)$). Der erste Punkt p_1 entspricht der Zahl $wich/(n_p + n_n + 1)$ und der dritte Punkt p_3 der Zahl $(wich \pmod{n_p + n_n + 1})$ (der zweite Punkt p_2 der Transition ist der übergebene Punkt `poi`). Ist die Zahl P, die dem Punkt entspricht, kleiner n_p ($P < n_p$), ist der Punkt der P'te positive Nachbarpunkt (die Zählung beginnt bei 0). Wenn die Zahl P größer gleich n_p und kleiner als $n_p + n_n$ ist ($n_p \geq P < n_p + n_n$), ist der Punkt der $(P - n_p)$'te negative Punkt (die Zählung beginnt auch bei 0). Sonst, wenn $P = n_p + n_n + 1$ ist, ist der Punkt ein neuer Punkt. Wenn zwei neue Punkte gewählt werden, dann werden immer zwei verschiedene neue Punkte gewählt. Mit den gewählten Punkten wird nach der Transitionstabelle in [1] die Veränderungen im Raumuniversum durchgeführt. Wird ein neuer Punkt erzeugt, ist dessen Name der Name des zuletzt erzeugten neuen Punktes erhöht um Eins. Auf diese Weise sind die durch Transitionen vergebenen Namen eindeutig.

Die `tick()` Methode ohne Prameter führt für jeden Punkt im Raumuniversum eine Transition (mit `evalueTransition(poi, wich)`) aus. Um die Auswahl zu beschleunigen, wird nicht aus dem ganzen Raumuniversum der nächste Ursprungspunkt für die Transition ausgewählt, sondern das Raumuniversum wird von vorn nach hinten durchwandert, wobei nach dem letzten Punkt des Raumuniversums sein erster Punkt kommt. Dafür enthält das Universum einen aktuellen Punkt p_a, der beim Erzeugen, Beschneiden und Laden des Raumuniversums auf den ersten (ältesten) Punkt der Raumuniversumsliste (`list<TPoint*> univers`) gesetzt wird.

Nach dem Aufruf der `tick()` Methode wird die Anzahl der Punkte im Raumuniversum durch 256 ermittelt, diese Zahl wird in eine ganze Zahl S größer 1 umgewandelt (if ($Punkte/256 > 1$) Then $S = Punkte/256$ Else $S = 1$). Nun wird

eine Schleife die Anzahl der Punkte im Raumuniversum mal durchlaufen. In dieser Schleife wird für den nächsten Transitionsursprungspunkt jeweils eine Zahl P ermittelt, welche die Anzahl möglichen Transitionen für des aktuelle Raumuniversum durch die zuvor ermittelte Zahl S ist. Nun wird auf eine Zahl Z, die beim Aufruf der `tick()` Methode mit 0 initialisiert wurde, eine Zufallszahl zwischen 0 und P (im Bereich von $0...(P-1)$) addiert. Solange diese Zahl Z größer gleich die Anzahl P_a der möglichen Transitionen des aktuellen Punkts p_a ist, wird diese (P_a) von erstere Z abgezogen ($Z = Z - P_a$) und für den aktuellen Punkt p_a der nächste Punkt (in der Liste) genommen. Ist der aktuelle Punkt p_a der letzte Punkt der Liste (des aktuellen Raumuniversums) wird der erste Punkt der Liste als aktueller Punkt p_a genommen.

Wenn Zahl Z kleiner als die Anzahl P_a der möglichen Transitionen des aktuellen Punkts p_a ist, wird die `evalueTransition(`p_a`, Z)` aufgerufen. Es wird also eine Transition ausgeführt, mit p_a als Ursprungspunkt und Z als Zahl die die beiden anderen Nachbarpunkte bestimmt.

Durch diese Art der Wahl der Ursprungspunkte für eine Transition, werden im Durchschnitt rund 128 Punkte zwischen den Transitionen in der Raumuniversumsliste übersprungen, bevor wieder ein Punkt Ursprungspunkt einer Transition wird. Die Wahl der Transitionsentstehungspunkte entspricht nicht der in der Theorie „Wilde Raumzeit“ geforderten, verkürzt aber die Berechnungszeit.

Die Methode `tick(trans)` wählt demgegenüber aus allen möglichen Transitionen des aktuellen Raumuniversums eine zufällig Gleichwahrscheinlich als nächste Transition aus. Dies wird dann `trans` mal wiederholt. Diese Art der Wahl entspricht der Wahl, wie sie mit der Theorie „Wilde Raumzeit“ konsistent ist, ist aber bei der gleichen Transitionsanzahl langsamer, als die Wahl der Methode `tick()` ohne Parameter.

4.2.2 universD

Dieser Simulator funktioniert wie der Simulator “univers“ des vorhergehenden Abschnitts, nur dass der Anzahl der möglichen Transitionen nicht die Anzahl der Nachbarpunkte eines Punktes zugrunde liegt, sondern die Anzahl der ausgehenden Distanzen. Dadurch wird die Berechnung der möglichen Transitionen etwas einfacher.

In diesem Simulator ist jedem Punkt und dem ganzem Raumuniversum eine Zahl für die Zahl der möglichen Transitionen zugeordnet. Die Anzahl der möglichen Transitionen für einen Punkt ist dabei (entgegen der Theorie) $d_a^2 = (d_p + d_n + 1)^2$, mit d_p Anzahl der positiven und d_n Anzahl der negativen ausgehenden Distanzen. Die Anzahl der möglichen Transitionen für das Raumuniversum ist die Summe der Anzahlen der möglichen Transitionen aller Punkte des Raumuniversums.

Im Simulator „universD“ wird mit der `evalueTransition(poi, wich)` eine Transition ausgeführt. Ihr wird der Zeiger zu dem Punkt übergeben, welcher der Entstehungspunkt der Transition sein soll. Der der Methode übergebene Para-

meter `wich` ($0 \leq wich < (n_p + n_n + 1)^2$) bestimmt die beiden Nachbarpunkte für die Transition ($t(Raumuniversum, p_1, p_2, p_3)$). Ein Punkt p wird dadurch gewählt, dass eine von Ursprungspunkt p_2 ausgehenden Distanz d gewählt wird, die zu ihm geht.

Die erste Distanz d_1 (zu dem ersten Punkt p_1) entspricht der Zahl $wich/(d_p + d_n + 1)$ und die zweite Distanz d_2 der Zahl $(wich \pmod{d_p + d_n + 1})$, diese Distanz d_2 bestimmt den dritten Punkt p_3 (der zweite Punkt p_2 der Transition ist der übergebene Punkt). Ist die Zahl D, die der Distanz entspricht, kleiner d_p ($D < d_p$), ist die Distanz die D'te positive Distanz (die Zählung beginnt bei 0). Wenn die Zahl D größer gleich d_p und kleiner als $d_p + d_n$ ist ($d_p \geq D < d_p + D_n$), ist die Distanz die $(D - d_p)$'te negative Distanz (die Zählung beginnt auch bei 0). Sonst, wenn $D = d_p + d_n + 1$ ist, ist der entsprechende Punkt ein neuer Punkt. Wenn zwei neue Punkte gewählt werden, dann werden immer zwei verschiedene neue Punkte gewählt. Mit den durch die Distanzen gewählten Punkten wird nach der Transitionstabelle in [1] die Veränderungen im Raumuniversum durchgeführt. Wird ein neuer Punkt erzeugt, ist dessen Name der Name des zuletzt erzeugten neuen Punktes erhöht um Eins. Auf diese Weise sind die durch Transitionen vergebenen Namen eindeutig.

Die `tick()` Methode ohne Prameter führt für jeden Punkt im Raumuniversum eine Transition (mit `evalueTransition(poi, wich)`) aus. Um die Auswahl zu beschleunigen, wird nicht aus dem ganzen Raumuniversum der nächste Ursprungspunkt für die Transition ausgewählt, sondern das Raumuniversum wird von vorn nach hinten durchwandert, wobei nach dem letzten Punkt des Raumuniversums sein erster Punkt kommt. Dafür enthält das Universum einen aktuellen Punkt p_a, der beim Erzeugen, Beschneiden und Laden des Raumuniversums auf den ersten (ältesten) Punkt der Raumuniversumsliste (`list<TPoint*> univers`) gesetzt wird.

Nach dem Aufruf der `tick()` Methode wird die Anzahl der Punkte im Raumuniversum durch 256 ermittelt, diese Zahl wird in eine ganze Zahl S größer 1 umgewandelt (if ($Punkte/256 > 1$) Then $S = Punkte/256$ Else $S = 1$). Nun wird eine Schleife die Anzahl der Punkte im Raumuniversum mal durchlaufen. In dieser Schleife wird für den nächsten Transitionsursprungspunkt jeweils eine Zahl P ermittelt, welche die Anzahl möglichen Transitionen für des aktuelle Raumuniversum durch die zuvor ermittelte Zahl S ist. Nun wird auf eine Zahl Z, die beim Aufruf der `tick()` Methode mit 0 initialisiert wurde, eine Zufallszahl zwischen 0 und P (im Bereich von $0...(P-1)$) addiert. Solange diese Zahl Z größer gleich die Anzahl P_a der möglichen Transitionen des aktuellen Punkts p_a ist, wird diese (P_a) von erstere Z abgezogen ($Z = Z - P_a$) und für den aktuellen Punkt p_a der nächste Punkt (in der Liste) genommen. Ist der aktuelle Punkt p_a der letzte Punkt der Liste (des aktuellen Raumuniversums) wird der erste Punkt der Liste als aktueller Punkt p_a genommen.

Wenn Zahl Z kleiner als die Anzahl P_a der möglichen Transitionen des aktuellen Punkts p_a ist, wird die `evalueTransition(`p_a`, Z)` aufgerufen. Es wird also eine Transition ausgeführt, mit p_a als Ursprungspunkt und Z als Zahl, welche

die beiden anderen Nachbarpunkte bestimmt.

Durch diese Art der Wahl der Ursprungspunkte für eine Transition, werden im Durchschnitt rund 128 Punkte zwischen den Transitionen in der Raumuniversumsliste übersprungen, bevor wieder ein Punkt Ursprungspunkt einer Transition wird.

Die Methode `tick(trans)` wählt demgegenüber aus allen möglichen Transitionen des aktuellen Raumuniversums einen zufällig Gleichwahrscheinlich als nächste Transition aus. Dies wird dann `trans` mal wiederholt.

4.2.3 universP

Im Simulator „universP“ wird mit der `evalueTransition(poi)` Methode eine Transition ausgeführt. Ihr wird der Zeiger zu dem Punkt übergeben, welcher der Entstehungspunkt der Transition sein soll. Die Methode wählt dann zufällig, Gleichwahrscheinlich aus den Nachbarpunkten plus einem neuem Punkt zwei Punkte aus (wenn zwei neue Punkte gewählt werden, dann werden immer zwei verschiedene neue Punkte gewählt) und führt nach der Transitionstabelle in [1] die Veränderungen im Raumuniversum durch. Wird ein neuer Punkt erzeugt, ist dessen Name der Name des zuletzt erzeugten neuen Punktes erhöht um Eins. Auf diese Weise sind die durch Transitionen vergebenen Namen eindeutig.

Die `tick()` Methode ohne Prameter führt für jeden Punkt im Raumuniversum nacheinander eine Transition (mit `evalueTransition(poi)`) aus. Begonnen wird beim ersten (ältesten) Punkt der Raumuniversumsliste (`list<TPoint*> univers`) und geendet bei dem Punkt, der beim Aufruf der `tick()` Methode der letzte in der Liste war. Durch diese Art der Wahl ist diese `tick()` Methode die schnellst aller Simulatoren.

Die Methode `tick(trans)` wählt demgegenüber aus allen Punkten des aktuellen Raumuniversums einen zufällig Gleichwahrscheinlich für die erste Transition aus, dann wird auf den nächsten `trans` Punkten jeweils eine Transition ausgeführt.

4.2.4 Weitere Methoden

Hier werden weitere Methoden der Simulatoren im Zusammenhang mit Transitionen aufgeführt.

Wenn der `start(trans)` Methode 0 oder kein Wert übergeben wird, wird die `tick()` Methode benutzt, sonst wird die `tick(trans)` Methode benutzt, wobei der übergebene Wert `trans` gleich dem der `start()` Methode übergebene Wert `trans` ist.

Das Universum enthält eine interne Zeit `iTime`, die angibt, wie oft seit dem erzeugen des Universums eine der `tick()` Methoden aufgerufen wurde. Sie kann mit `getInternalTime()` abgefragt werden.

Die Methode `getNumberOfLastTransitions()` liefert ein Array mit 10 Elementen zurück. Der erste Wert (mit Index 0) liefert die gesamt Anzahl der Transitionen zurück, der Wert mit Index i die Anzahl der Transition t_i (nach der

Transitionstabelle in [1]). Die Anzahlen beziehen sich jeweils entweder auf die Anzahl der Transitionen, die seit dem letzten Aufruf von `tick()` (ohne Parameter) gemacht wurden, oder, wenn `tick()` nicht verwendet wurde, auf die gesamte Anzahl der Transitionen.

Die Methode `endCondition()` gibt genau dann Wahr zurück, wenn die weitere Berechnung des Universums (weitere Ticks bzw. `tick()` Aufrufe) abgebrochen werden soll (siehe 4.4.2 auf Seite 11).

4.3 Ein- und Ausgabe von Raumuniversen

Methoden: `bool printUnivers(ostream &ostream);`
`bool readUnivers(istream &stream)`
Klassen: `TOPoint`

Das aktuelle Raumuniversum kann mit der Methode `printUnivers()` in einen Stream (z.B. zur Festplatte) gespeichert und mit der Methode `readUnivers()` durch ein Raumuniversum aus dem Stream (z.B. aus einer Datei) ersetzt werden. Dafür wird den Methoden jeweils ein Zeiger auf den Stream übergeben. Die Methoden liefern `false` zurück, wenn die Ein- bzw. Ausgabe nicht erfolgreich war.

Im Stream werden die Punkt und ihre Verbindungen mithilfe ihrer Namen abgespeichert. Jede Zeile wird durch ein „.“ beendet. In der ersten Zeile steht die gesamte Anzahl der Punkte des im Stream abgespeicherten Raumuniversum. Danach folgt für jeden Punkt des Raumuniversums eine eigene Zeile, wobei die Punkte nach ihren Namen (bzw. Entstehungszeit) aufsteigend geordnet sind. Als erstes steht dabei in der Zeile der Name des Punktes gefolgt von einem Semikolon „;“, dann folgen jeweils die Nachbarpunkte, wobei vor jedem Nachbarpunkt ein Komma steht „,“. Zuerst werden die Nachbarn aufgeführt zu denen eine positive Distanz geht und dann die Nachbarn zu denen eine negative Distanz geht, diese beiden Nachbarlisten werden, auch wenn sie leer sind, durch ein Semikolon „;“ getrennt.

Beispiel eines gespeicherten Raumuniversums:

```
4.
2;,3;,6.
3;,2;,6.
6;;,2,3.
10;;.
```

4.4 Parameter

Methoden: `bool restoreParameter(char* pFile);`
`bool storeParameter(char* pFile)`
Daten: `list<TParameter> parameter`
Klassen: `class TParameter`
Dateien: standart_Parameter.txt; Null_Parameter.txt

Die meisten Parameter dienen zur Ausgabe und Formatierung von Statistiken, nur sehr wenige (z.B. das Anfangsraumuniversum) haben einen Einfluss auf die Entwicklung der Raumuniversumsfolge.

Für die einzelnen Parameter gibt es jeweils zwei mögliche Methoden, „`get`“- und „`set`“-Methoden. Die Methoden die mit „`get`“ beginnen dienen zur Ausgabe des Wertes des Parameters. Dagegen kann mit Methoden die mit „`set`“ beginnen der Wertes des Parameters gesetzt werden. Daneben werden von „`set`“-Methoden manchmal noch zugehörige Operationen ausgeführt, z.B. wird eine zum Parameter gehörende Datei geöffnet. Ist eine der beiden oder sind beide Methoden nicht vorhanden, soll auf dem zugehörigen Parameter nicht die Operation(-en) ausgeführt werden können.

Ist für einzelne Statistiken, die Daten für Raumuniversen zu verschiedenen Zeiten in einer Datei ausgeben, der Parameter zur Ausgabe gesetzt und wird ein neuer Dateiname gesetzt oder es wird nur der Parameter zur Ausgabe (auf größer 0 oder wahr) gesetzt, so wird automatisch die Datei erzeugt. Deshalb ist beim setzen dieser Parameter mit etwas Vorsicht vorzugehen um Fehler oder zusätzliche, leere und überflüssige Dateien zu vermeiden.

Jedem Parameter ist neben seinem Wert noch eine Nummer, eine Zeilennummer und eine Beschreibung bzw. ein Name zugeordnet. Die Nummer, die Zeilennummer und die Beschreibung zu einem Parameter können in einem Objekt der Klasse `TParameter` gespeichert werden, wobei auf diesen Objekten eine Ordnung nach ihrer Zeilennummer (Parameter mit der niedrigerer Zeilennummer kommen zuerst) definiert ist. In der Parameterliste `parameter` (`list<TParameter> parameter`) sind diese Zusatzinformationen der Parameter des Programmsystems zusammengefasst.

Die Parameter können in Parameterdateien abgespeichert und aus diesen wieder geladen werden. Dabei identifiziert die Parameternummer den zu der Zeile zugehörigen Parameter. Die Parameter können also in beliebiger Reihenfolge und Zeil geschrieben werden und eine beliebige Parameterbeschreibung haben, nur die richtige Parameternummer ist wichtig. Mit der Methode `restoreParameter( pFile)` werden die Parameter mit Zusatzinformationen aus der Datei mit dem (Pfad-)Namen `pFile` gelesen. Mit der Methode `storeParameter(pFile)` werden die aktuellen Parameter mit Zusatzinformationen in die Datei mit dem (Pfad-)Namen `pFile` geschrieben. Dafür werden zuerst die Parameter ihrer Zeilennummer nach geordnet.

In der Paramerterdatei sind die Daten für die Parameter in einer lesbaren und manipulierbaren Form abgespeichert. Jede Zeile steht dabei für die Informationen über einen Parameter. Die Zeile beginnt dabei mit der Nummer des Parameters gefolgt von einem Punkt „.“, dann kommt die Parameterbeschreibung gefolgt von einem Doppelpunkt „:“, als letztes kommt der Parameterwert, die Zeile wird durch ein Semikolon beendet. Die Parameternummer 0 oder undefinierte Parameternummern stehen für Kommentare. Zeilen für Kommentare enthalten nur ihre Paramerternummer gefolgt von der Beschreibung (z.B. „0. Kommentar;“).

Ist der Parameterwert ein Wahrheitswert, werden Zeichenketten die mit „j“, „t“, „w“, „y“ und „1“ beginnen als wahr angesehen, sonst werden sie als falsch angesehen. Wenn der Parameterwert ein Zeichenketten ist, so sind die „:“ und „;“ Zeichen als Begrenzer der Zeichenkette anzusehen (z.B. bei „12. Dateinamen :univers;“ ist der Parameterwert „univers“).

Die Datei „standart_Parameter.txt“ enthält die Standartwerte und Zusatzinformationen der Parameter, diese werden geladen, wenn keine andere Parameterdatei angegeben wurde. Sie kann auch als Anfangsgerüßt für neue Parameterdateien dienen, indem sie unter anderen Namen gespeichert und verändert wird.

Die Datei mit Namen „Null_Parameter.txt“ dient einer Initialisierung des Programmsystems ohne Nebeneffekte und sollte deshalb nicht verändert werden. Sie ist aber nur von Interesse, wenn das Programmsystem verändert werden soll.

4.4.1 Anfangswerte

Methoden: `long getTimeBase(); void setTimeBase(long bTim)`

Nr.	Klassenvariable	Standartparameterbeschreibung
4	`long bTime`	Anfangszeit:
5	`bool appendUnivers`	Universum fortsetzen:
6		Dateiname des Anfangsuniversums:

Diese Parameter bestimmen die Anfangswerte des Universums.

Die Anfangszeit `bTime` (die dazugehörige Methoden: `getTimeBase()` und `setTimeBase()`) ist der Anfangswert bei welchem mit der Zählung der Zeitschritte begonnen werden soll. Sie wirkt sich auf die Ausgabe der Statistiken aus.

Wenn bei Parameter 6 („Dateiname des Anfangsuniversums:“) ein gültiger Pfad einer Datei für ein Raumuniversum angegeben wurde, wird dieses automatisch mit der Parameterdatei geladen. Dieses dient dann als aktuelles (Anfangs-) Raumuniversum.

Wenn neue Parameter in das Programmsystem geladen werden, kann angegeben werden, dass ein bestehendes Universum fortgesetzt werden soll. Dafür dienen Parameter 5 und 6. In diesem Fall wird das angegebene Universum geladen und die Statistikdateien an der aktuellen Universumszeit fortgesetzt. Mehr dazu im Abschnitt 4.4.6 auf Seite 16 .

4.4.2 Endbedingung

Methoden:
```
unsigned long getMaxNumberOfTime();
    void setMaxNumberOfTime(unsigned long mTm);
    unsigned long getMaxNumberOfPoints();
    void setMaxNumberOfPoints(unsigned long mPoi);
    unsigned long getCheckMaxPoints();
    void setCheckMaxPoints(unsigned long cMP)
```

Nr.	Klassenvariable	Standartparameterbeschreibung
1	`unsigned long maxTime`	Maximale Zeit:
2	`unsigned long maxPoints`	Maximale Anzahl von Punkten:
3	`unsigned long checkMaxPoints`	Ticks zwischen dem Prüfen der maximale-Punkte-Bedingung
51	`time_t timeMaxTimeIteration`	Zeit die eine Iteration maximal dauern darf
	`time_t timeOfIterationBegin`	Zeit wann die aktuelle Iteration began
52	`time_t timeMaxTime`	Zeit die die Berechnung maximal dauern darf
	`time_t timeOfBegin`	Zeit wann die erste Iteration began

Diese Parameter geben an, wann die Berechnung weiterer Ticks abgebrochen werden soll. Wenn die Zeit größer als oder gleich `maxTime` (zugehörige Methoden: `getMaxNumberOfTime()` und `setMaxNumberOfTime()`) ist, werden keine weiteren Ticks berechnet. Weiterhin wird alle `checkMaxPoints` (zugehörige Methoden: `getCheckMaxPoints()` und `setCheckMaxPoints()`) Ticks überprüft, ob das aktuelle Raumuniversum weniger als `maxPoints` (zugehörige Methoden: `getMaxNumberOfPoints()` und `setMaxNumberOfPoints()`) Punkte enthält, nur wenn dies der Fall ist, werden weiterer Ticks berechnet.

Die Parameter 51 und 52 dienen dazu die (reale) Zeit, welche die Berechnung benötigt, zu begrenzen. Mit `timeMaxTimeIteration` kann angegeben werden, wie lange eine einzelle Iteration maximal dauern darf. Wenn also „2 Minuten“ angegeben wird, wird bei der ersten Iteration, welche mehr als zwei Minuten zur Berechnug benötigte, abgebrochen. Der Parameter `timeMaxTime` bezieht sich dem gegenüber auf die Zeit, welche alle Iterationen zusammen benötigen dürfen. Als Werte für `timeMaxTimeIteration` und `timeMaxTime` in der Parameterdatei sind dabei ein Zahl gefolgt von einer Zeichenkette zulässig. Beginnt die Zeichenkette mit „m“ oder „M“ wird die Zahl als Anzahl von Minuten interpretiert, bei „s“, „S“, „h“ oder „H“ als Stunden und bei „t“, „T“, „d“ oder „D“ als Tage. Folgt keine Zeichenkette, wird die Zahl als Anzahl von Sekunden interpretiert.

Für alle Endbedingungsparameter gilt: Ist der Wert eines Parameters gleich 0 wird die zugehörige Bedingung nicht überprüft.

4.4.3 Grundstatistiken

Methoden:
```
bool getCommentsActiv();
void setCommentsActiv(bool comAc);
unsigned long getOutStatisticCycle();
void setOutStatisticCycle(unsigned long outS);
char* getStatisticFileName();
bool setStatisticFileName(char* statF)
```

Daten: `bool outComent; unsigned long outStat;`
`char* statFileName; fstream *statFile`

Nr.	Klassenvariable	Standartparameterbeschreibung
7	`bool outComent`	Kommentare Ausgeben:
8	`unsigned long outStat`	Ausgaben der Grundstatistiken:
9	`char* statFileName`	Name der Grundstatistikendatei:

Diese Parameter regeln die Ausgabe von einfachen statistischen Werten.

Der Parameter `outComent` (zugehörige Methoden: `getCommentsActiv()` und `setCommentsActiv()`) gibt an, ob Statistiken und Meldungen in die Standardausgabe (Bildschirm) ausgegeben werden sollen (wenn er `true` ist, erfolgt die Ausgabe).

Bei jedem Tick-Schritt erfolgt dafür die Ausgabe der aktuellen Zeit („`time :`"), der aktuellen Iteration (bzw. internen Zeit „`iteration :`"), der Anzahl der Punkte („`Points :`"), positiven („`pos distances :`"), negativen Distanzen („`neg distances :`") und aller Distanzen zusammen durch die Anzahl der Punkte („`distances/points :`"). Bei den Simulatoren „univers" und „universD" wird zusätzlich noch die Anzahl der möglichen Transitionen („`posible transitions :`") im aktuellen Raumuniversum ausgegeben.

Wenn das Raumuniversum beschnitten wird, wird ausgegeben, wie viele zusammenhängende Teile das Raumuniversum enthielt („cutparts") und dann, bis auf die Zeit und Iterationszahl, die Daten, die sonst bei jedem Tick ausgegeben werden, noch einmal für das beschnittene Raumuniversum.

Beim Einlesen eines Raumuniversums aus einer Datei wird, wenn `outComent true` ist, ausgegeben, wie viele Punkte das eingelesene Raumuniversum enthält („`univers size :`"), für den Einlesefortschritt (bei der Konvertierung der Daten) wird alle 256 Punkte die Nummer des aktuellen Punktes (nicht sein Name) ausgegeben und am Ende wird der erfolgreiche Abschluss des Einlesevorgangs bestätigt.

Der Parameter `outStat` (Methoden dazu: `getOutStatisticCycle()` und `setOutStatisticCycle()`) regelt die Ausgabe von Statistikdaten in eine Statistikdatei. Die Statistik bezieht sich dabei auf das Raumuniversum nach dem Beschneiden. Ist der Parameter größer 0 (bei 0 erfolgt keine Ausgabe) werden alle `outStat` Ticks die Werte des aktuellen Raumuniversums und die Werte der Anzahlen der Transitionen, seit dem letzten Aufruf der `tick()` Methode (ohne Parameter), in eine Datei mit dem Namen bzw. Pfad `statFileName` (zugehörige Methoden: `getStatisticFileName()` und `setStatisticFileName()`) ausgegeben. Die erste Zeile beinhaltet den Tabellenkopf. Danach werden die Werte für das jeweils aktuelle Raumuniversum in eine neue Zeile geschrieben, wobei hinter jedem Wert ein Semikolon „;" kommt. Zu diesen Werten gehören der Reihenfolge nach die Zeit (interne Zeit plus Zeitbasis), die Anzahl der Punkte, positiven und negativen Distanzen im aktuellen Raumuniversum, sowie der Quotient der Anzahl aller Distanzen durch die Anzahl der Punkte im aktuellen Raumuniversum.

Danach werden die Werte der Transitionen ausgegeben, dafür wird zuerst die Anzahl aller Transitionen zusammen ausgegeben und dann, mit der ersten Transition t_1 beginnend, der Reihenfolge nach jeweils die Anzahl der Transition und der entsprechende Anteil, an allen Transitionen, der einzelnen neun Transitionen. Bei den Simulatoren „univers“ und „universD“ wird am Ende zusätzlich noch die Anzahl der möglichen Transitionen („`number of posible transitions`“) im aktuellen Raumuniversum ausgegeben.

4.4.4 Realzeitstatistik

Methoden: `bool getOutRealTime();`
`void setOutRealTime(bool rT);`
`char* getRTimeFileName();`
`bool setRTimeFileName(char* rTimeF)`
Daten: `bool outRealTime; char* rTimeFileName;`
`fstream *rTimeFile`

Nr.	Klassenvariable	Standartparameterbeschreibung
16	`unsigned long outRealTime`	Realzeitstatistik ausgeben:
17	`char* rTimeFileName`	Realzeitstatistik Datei:

Diese Parameter regeln die Ausgabe von der Statistiken für die reale Zeit. Diese Statistiken beinhalten die Uhrzeit und die Zeit, in Sekunden, welche die einzelnen Schritte benötigen.

Wenn der Parameter `outRealTime` (die zu ihm gehörigen Methoden sind: `getOutRealTime()` und `setOutRealTime()`) größer als 0 ist, wird alle `outRealTime` Runden/Iterationen die Realzeitstatistik in die Datei mit dem in `rTimeFileName` (die dazugehörigen Methoden: `getRTimeFileName()` und `setRTimeFileName()`) stehenden Pfad ausgegeben. Die erste Zeile beinhaltet den Tabellenkopf. Danach werden die Werte für das aktuelle Raumuniversum jeweils in eine neue Zeile geschrieben, wobei hinter jedem Wert ein Semikolon „;“ kommt. Zu diesen Werten gehören der Reihenfolge nach die Zeit (interne Zeit plus Zeitbasis), das aktuelle Datum und die aktuelle Zeit, dann kommen die Sekunden, die die Iteration insgesamt benötigt hat, danach kommen die Werte für die Anzahlen der Sekunden, die die einzelnen Schritte benötigt haben. Zu diesen Werten gehören der Reihe nach die Zeit für das erzeugen des neuen Raumuniversums (bzw. für die `tick()` Methode), für das Beschneiden des Raumuniversums, für die Ausgabe des Raumuniversums, für die direkte Rasterstatistik plus, wenn vorhanden, für das erstellen des Raster, für die Nachbarschaftsstatistik, für die Abstandsstatistik und am Ende für die Statistik für die maximale Ausdehnung des Raumuniversums.

4.4.5 Ausgabe der Raumuniversen

Methoden: `unsigned long getOutUniversCycle();`

```
    void setOutUniversCycle(unsigned long outU);
    char* getUniversFolderName();
    void setUniversFolderName(char* uFol);
    char* getUniversFileBaseName();
    void setUniversFileBaseName(char* uFB)
Daten: char* uFolder; char* bUName; unsigned long outUniv
```

Nr.	Klassenvariable	Standartparameterbeschreibung
10	`unsigned long outUniv`	Universen ausgeben:
50	`time_t timeMaxBetweanSaves` `time_t timeOfLasteSave`	Sicherheitskopie des Raumuniversums mindestens alle:
11	`char* uFolder`	Verzeichnis in dem die Universen gespeichert werden:
12	`char* bUName`	Erster Teil der Dateinamen für die Universen:

Während eines Laufes können die Raumuniversen zwischen den einzelnen Ticks ausgegeben werden. Der Parameter `outUniv` (die dazugehörige Methoden sind: `getOutUniversCycle()` und `setOutUniversCycle()`) gibt dabei an, alle wie vielen Ticks das dann aktuelle Raumuniversum ausgegeben werden soll. Ist `outUniv` gleich 0 wird kein Raumuniversum ausgegeben. In dem Parameter `uFolder` (die dazugehörige Methoden sind: `getUniversFolderName()` und `setUniversFolderName()`) ist der Name des Verzeichnisses abgespeichert, in dem die Raumuniversen ausgegeben werden. Der Parameter `bUName` (`getUniversFileBaseName()` und `setUniversFileBaseName()`) enthält den Basisdateinamen unter dem die Raumuniversen ausgegeben werden, an diesem Basisdateinamen wird dann die Nummer der aktuellen Zeit und die Endung „.txt“ angehängt. Der vollständige Pfad der ausgegebenen Raumuniversums lautet: `uFolder` + `bUName` + Zeit + „.txt“, ein möglicher Pfad wäre also: „./universen/univ12.txt“. In den Dateien werden die Daten der Raumuniversen in der im Abschnitt 4.3 auf Seite 9 beschriebenen Form ausgegeben, bzw. zur Ausgabe wird die Methode `printUnivers()` verwendet.

Der Parameter `timeMaxBetweanSaves` gibt an, nach wie viel Zeit das Universum wieder ausgegeben werden soll. Wenn `timeMaxBetweanSaves` größer 0 ist und der Zeitpunkt an dem das letzte mal ein Raumuniversum gespeichert wurde (`timeOfLasteSave`) länger als `timeMaxBetweanSaves` her ist, wird das aktuelle Raumuniversum, wie oben beschrieben, wieder ausgegeben. Als Werte für `timeMaxBetweanSaves` in der Parameterdatei sind dabei ein Zahl gefolgt von einer Zeichenkette zulässig. Beginnt die Zeichenkette mit „m“ oder „M“ wird die Zahl als Anzahl von Minuten interpretiert, bei „s“, „S“, „h“ oder „H“ als Stunden und bei „t“, „T“, „d“ oder „D“ als Tage. Folgt keine Zeichenkette, wird die Zahl als Anzahl von Sekunden interpretiert.

4.4.6 Universum fortsetzen

Daten: `bool appendUnivers; char* parameterFile`

Nr.	Klassenvariable	Standartparameterbeschreibung
4	`long bTime`	Anfangszeit:
5	`bool appendUnivers`	Universum fortsetzen:
6		Dateiname des Anfangsuniversums:

Es existiert die Möglichkeit die Simulation einer Raumuniversumsfolge zu unterbrechen und später fortzusetzen. Dazu muss in der Parameterdatei der Parameter `appendUnivers` auf wahr gesetzt werden. Ist `appendUnivers` wahr, wird versucht ein vorhandenes Raumuniversum aus der Datei für das Anfangsuniversum zu laden, wenn dies nicht möglich ist, wird mit einem neuen leeren Raumuniversum eine neue Testreihe begonnen. Wird ein Raumuniversum geladen, so werden auch die vorhandenen Statistikdateien, wenn möglich, bei dem Zeitpunkt `bTime` fortgesetzt. Die interne Zeit wird auf `bTime` gesetzt.

Wenn das Universum fortgesetzt wird, kann mit „E“ (Großbuchstabe) die aktuelle Testreihe beendet werden. Wird „E“ eingegeben, wird die aktuelle Iteration bzw. Runde noch beendet, dann wird, wenn nötig, das aktuelle Raumuniversum, wie im Abschnitt 4.4.5 beschrieben, gespeichert und das aktuelle Raster wird, wenn nötig, in eine Datei „grid.txt“ im Statistikverzeichnis gespeichert. In der zuletzt geladenen Parameterdatei (`parameterFile`) wird die Anfangszeit auf die aktuelle Zeit gesetzt und der Pfadname, des zuletzt gespeicherten Raumuniversums, unter „Dateiname des Anfangsuniversums“ angegeben.

Wenn nun die vorher zuletzt geladenen Parameterdatei (`parameterFile`) erneut beim Start des Simulators angegeben wird, wird der begonnene Lauf automatisch fortgesetzt.

Wie gesagt wird das Raumuniversum wie im Abschnitt 4.4.5 beschrieben ausgegeben. Sollten die Parameter 11 `uFolder` und 12 `bUName` also leer sein, wird als Dateiname nur die aktuelle Zeit gefolgt von „.txt“ genommen (z.B. „34.txt“). Wenn die Raumuniversen im allgemeinen nicht ausgegeben werden sollen, wäre es Beispielsweise sinnvoll `uFolder` leer zu lassen und für `bUName` „tmp“ anzugeben, so dass die zwischengespeicherten Universen im aktuellen Verzeichnis liegen und mit „tmp“ beginnen.

Die zwischengespeicherten Raumuniversen werden nicht automatisch gelöscht, wenn sie nicht mehr gebraucht werden. Diese Aufgabe sollte der Benutzer übernehmen. Für die Fortsetzung einer Testreihe wird nur das zuletzt gespeicherte Raumuniversum benötigt, also das mit der höchsten Zahl für die Zeit.

Wird während des Testlaufes „C“ eingegeben, so wird der Testlauf am Ende der Iteration abgebrochen, ohne den aktuellen Stand zu speichern. Der Testlauf wird dann nicht an der aktuellen Stelle fortgesetzt, wenn die zuletzt geladene Parameterdatei erneut geladen wird.

4.4.7 Universum beschneiden

Methoden: `unsigned long getCutCycle();`
`void setCutCycle(unsigned long cutA);`
`double getCutNearDimension();`
`void setCutNearDimension(double cND);`
`double getCutMaxClassRatio();`
`void setCutMaxClassRatio(double cMCR);`
`double getCutRemainingParts();`
`void setCutRemainingParts(double dCutRemainingParts);`
`void cutUnivers(double dim, double maxCl, double dInCutRemainingParts);`

Daten: `unsigned long cutAfter;double cutNearDimension;`
`double cutMaxClassRatio;double cutRemainingParts;`

Nr.	Klassenvariable	Standartparameterbeschreibung
13	`unsigned long cutAfter`	Ticks zwischen dem Beschneiden des Universums:
14	`double cutNearDimension`	Dimension dem der verbleibende Raumuniversumsteil nahe kommen soll:
15	`double cutMaxClassRatio`	Größe der zusammenhängenden Raum -universen, im Bezug auf den größten Raumuniversumsteil, die in der Auswahlmenge sein dürfen:
60	`double cutRemainingParts`	Anteil (bei < 1) oder Anzahl (bei >=1) der Raumuniversumsteile die nicht herrausgeschnitten werden sollen:

Bei Transitionen kann es vorkommen, dass das Universum in nicht zusammenhängende Teile aufgespalten wird. Meist wird dabei nur ein kleiner Teil des Universums, der wenige Punkte und Distanzen enthält und schnell expandiert, vom Rest abgetrennt. Um Rechenzeit zu sparen können alle zusammenhängende Teile des Universums, bis auf einige, gelöscht werden, dies geschieht mit Hilfe der Methode `cutUnivers()`.

Dabei werden in der Methode `cutUnivers(dim, maxCl, dInCutRemainingParts)` zuerst alle zusammenhängenden Teile, diese sind selbst wieder Raumuniversen, des aktuellen Raumuniversums ermittelt, einige dieser werden ausgewählt und zum aktuellen Raumuniversum gemacht, die restlichen Teile werden gelöscht. Die Auswahl der zusammenhängenden Raumuniversen, die nicht gelöscht werden soll, wird durch drei Parameter bestimmt `dim`, `maxCl` und `dInCutRemainingParts`.

Ausgewählt wird aus der Menge der zusammenhängenden Raumuniversen, deren Anzahl von Distanzen mindestens `maxCl` mal der Anzahl der Distanzen im größten zusammenhängenden Raumuniversum (dem mit dem meisten Distanzen) beträgt. Auf diese Weise enthalten die ausgewählten Raumuniversen mindestens

einen Anteil von maxCl Distanzen des größten Raumuniversums. So kann vermieden werden, dass nur sehr kleine zusammenhängende Raumuniversen ausgewählt werden.

Bei der Auswahl werden die Raumuniversumen gewählt, deren Dimension am nächsten zu dim ist. Die Dimension ergibt sich dabei aus der Anzahl der Distanzen (beider Arten zusammen) durch die Anzahl der Punkte im betrachteten Raumuniversum (Dimension=(positive Distanzen + negative Distanzen)/Punkte).

Mit dem Parameter dInCutRemainingParts wird die Anzahl der auszuwählenden Raumuniversen bestimmt. Ist er kleiner 1, werden von der Menge der zusammenhängende Raumuniversumsteile der Anteil dInCutRemainingParts ausgewählt. Gibt es also im ganzen Raumuniversum N zusammenhängende Raumuniversumsteile, werden davon $\lfloor N * dInCutRemainingParts \rfloor$ ausgewählt. Ist dInCutRemainingParts größer oder gleich 1, werden von den zusammenhängende Raumuniversumsteilen $\lfloor dInCutRemainingParts \rfloor$ ausgewählt. Der Parameter dInCutRemainingParts gibt dann also direkt die Anzahl der auszuwählenden zusammenhängenden Raumuniversumsteile an.

In der start() Methode, also bei einem Lauf, wird alle cutAfter (zugehörige Methoden: getCutCycle() und setCutCycle()) Ticks das Universum beschnitten. Dabei wird beim Aufruf der Methode cutUnivers(dim, maxCl, dInCutRemainingParts), der Parameter maxCl auf den Wert des Parameters cutMaxClassRatio (zum Parameter maxCl gehörigen Methoden sind: getCutMaxClassRatio() und setCutMaxClassRatio()), der Parameter dim wird auf den Wert von cutNearDimension (die dazugehörige Methoden sind: getCutNearDimension() und setCutMaxClassRatio()) und der Parameter dInCutRemainingParts auf cutRemainingParts (die zum diesem Parameter zugehörige Methoden sind: getCutRemainingParts() und setCutRemainingParts()) gesetzt.

4.4.8 Beschneidungsstatistik

Methoden:
```
unsigned long getOutCutStatistic();
void setOutCutStatistic(unsigned long cut);
char* getCutFileName();
bool setCutFileName(char* cutF)
```
Daten:
```
unsigned long outCutStatistic; char* cutFileName;
fstream *cutFile
```

Nr.	Klassenvariable	Standartparameterbeschreibung
18	unsigned long outCutStatistic	Beschneidungsstatistik ausgeben:
19	char* cutFileName	Beschneidungsstatistik Dateiname:

Zu dem Beschneiden der Raumuniversen kann eine Statistik in eine Datei ausgegeben werden.

Wenn der Parameter `outCutStatistic` (die zugehörigen Methoden sind: `getOutCutStatistic()` und `setOutCutStatistic()`) größer 0 ist, wird alle `outCutStatistic` Ticks/Iterationen beim Beschneiden eines Raumuniversums die Beschneidungsstatistik in die Datei mit dem in `cutFileName` (zugehörige Methoden: `getCutFileName()` und `setCutFileName()`) stehenden Pfad ausgegeben. Die erste Zeile beinhaltet den Tabellenkopf. Danach werden die Werte für das aktuelle Raumuniversum und der Beschneidung jeweils in eine neue Zeile geschrieben, wobei hinter jedem Wert ein Semikolon „;“ kommt.

Zu diesen Werten gehören, der Reihenfolge nach, die Zeit (interne Zeit plus Zeitbasis), dann kommt die Anzahl der Punkte, positiven und negativen Distanzen des Raumuniversums vor dem Beschneiden, danach folgt die Anzahl der zusammenhängenden Raumuniversumsteile und am Ende folgt eine Liste mit den Daten zu den einzelnen Raumuniversumsteilen. In dieser Liste steht, nach der Größe bzw. Anzahl von Punkten geordnet, für jeden zusammenhängende Raumuniversumsteil ein Klammerausdruck. Zwischen den Klammern für einen Raumuniversumsteil steht der Reihe nach, durch Komma „,“ getrennt, die Anzahl der Punkte, positiven und negativen Distanzen in ihm.

Beispiel für eine Zeile der Beschneidungsstatistik:

```
54; 11; 4; 5; 3; ( 6, 2, 4 ) ( 4, 2, 1 ) ( 1, 0, 0 ) ;
```

4.4.9 Maximale Ausdehnung des Raumuniversums

Methoden:
```
unsigned long getPartSizeStatistics();
void setGetPartSize(unsigned long pSize);
char* getPartSizeFileName();
void setPartSizeFileName(char* partN)
```

Nr.	Klassenvariable	Standartparameterbeschreibung
25	`unsigned long getPartSize`	Maximale Ausdehnung des Raumuniversums vom ältesten Punkt aus ausgeben:
26	`char *partSizeName`	Dateiname der maximalen Ausdehnungsstatistik:

Um eine Vorstellung von der Größe des aktuellen Raumuniversums zu bekommen, kann ein Wert für die Ausdehnung eines Teils des Raumuniversums ermittelt werden. Der Wert für die maximale Ausdehnung gibt den Abstand zwischen den ersten/ältesten Punkt des Raumuniversums und den Punkt, der von diesem am weitesten entfernt, aber noch im gleichen zusammenhängenden Teil des Raumuniversums ist, wieder.

Dieser Wert wird alle `getPartSize` (zum Parameter gehörende Methoden: `getPartSizeStatistics()` und `setGetPartSize()`) Ticks ermittelt oder gar nicht, wenn `getPartSize` gleich 0 ist. Der ermittelten Werte wird in die Datei mit Namen `partSizeName` (die zum Parameter gehörenden Methoden sind:

getPartSizeFileName() und setPartSizeFileName()) geschrieben. In der Datei sind die Daten für die einzelnen Raumuniversen Zeilenweise getrennt (pro Tick eine Zeile). In einer Zeile kommt dabei als erstes die aktuelle Zeit und dann die maximale Ausdehnung, beides gefolgt durch ein Semikolon „;“.

4.5 Parameter für die Rasterestatistiken

Um die lokalen Strukturen und Entwicklungsverläufe besser Untersuchen zu können, kann, wenn ein Raumuniversum der Folge eine bestimmte Größe überschritten hatte, ein Menge von näher benachbarten Punkten aus ihm als Raster ausgewählt werden. Das gleiche Raster von Punkten kann dann bei den nachfolgenden Raumuniversen auf verschiedene Weisen untersucht werden und die entsprechenden Statistiken ausgegeben werden.

Während die Parameter und Methoden aus dem vorhergehenden Abschnitt 4.4, bis auf die der Unterabschnitte 4.4.9 und 4.4.6, sich auf die Klasse Univers beziehen, beziehen sich die nachfolgenden Parameter und Methoden auf die Klasse UniversNeib.

4.5.1 Raster erzeugen

Methoden:
```
list<TPoint*> makeGrid(TPoint* sPoi,
 unsigned int dist, unsigned int number);
list<TPoint*> getSurface(list<TPoint*> poiL);
list<unsigned long> getGridNames();
void setGridNames(list<unsigned long> grN);
unsigned long getGridDistance();
bool setGridDistance(unsigned long grD);
unsigned long getNumberOfGridPoints();
bool setNumberOfGridPoints(unsigned long grP);
unsigned long getGridAfterPoints();
bool setGridAfterPoints(unsigned long grAP);
double getGridStartPoint();
bool setGridStartPoint(double grSP);
char* getStatisticFolderName();
void setStatisticFolderName(char* statFN)
```
Daten:
```
list<unsigned long> gridN;
unsigned long gridAfterPoints;
unsigned long gridPoints;
unsigned long gridDist; double gridStartPoint;
char *statisticFolderName
```

Nr.	Klassenvariable	Standartparameterbeschreibung
20	unsigned long gridAfterPoints	Raster erzeugen wenn das Universum mindestens soviel Punkt enthält:
21	unsigned long	Anzahl der Punkt im erzeugten Raster:

	gridPoints	
22	unsigned long gridDist	Mindest Abstand zweier Punkte im erzeugten Raster:
23	double gridStartPoint	Position des Rasterstartpunktes im Raumuniversum:
24	char *statisticFolderName	Name des Statistikverzeichnisses:

Ausgangspunkt für die Rasterstatistiken ist ein Raster (gridN) dessen Punkte im Raumuniversum, in dem es erzeugt wurde, bestimmte Vorgaben erfüllen. Das Raster wird automatisch (zwischen zwei Ticks) erzeugt, wenn kein Raster existiert (bzw. es (gridN) keine Punkte enthält), eines erzeugt werden soll (gridPoints>0) und das aktuelle Raumuniversum mehr als gridAfterPoints (zugehörige Methoden: getGridAfterPoints() und setGridAfterPoints()) Punkt enthält. Als Raster gridN (die dazugehörige Methoden sind: getGridNames() und setGridNames()) werden die Namen (bzw. Nummern) der Rasterpunkt abgespeichert. Bei der Auswertung des Rasters werden aber nur Punkte betrachtet die auch im aktuellen Raumuniversum enthalten sind.

Um das Raster zu erzeugen wird zuerst das Universum beschnitten, wenn es beschnitten werden soll (wenn cutAfter > 0). Dann wird der Startpunkt für das Raster ausgewählt. Dies geschieht mithilfe des Parameters gridStartPoint (Methoden: getGridStartPoint() und setGridStartPoint()). Um den Startpunkt für das Raster zu ermitteln wird die Anzahl der Punkte im aktuellen Raumuniversum mit gridStartPoint multipliziert, das abgerundete Ergebnis gibt die Position (begonnen mit den ältesten Punkt) des Startpunktes im aktuellen Raumuniversum an (Anzahl der Punkte im Raumuniversum $*$ gridStartPoint $=$ Position des Startpunktes). Danach wird der Startpunkt heraus gesucht und die Methode makeGrid() (makeGrid(Startpunkt,gridDist,gridPoints)) aufgerufen, die das Raster erzeugt. Zu dem Parameter gridDist gehören die Methoden getGridDistance() und setGridDistance() und zu dem Parameter gridPoints gehören die Methoden getNumberOfGridPoints() und setNumberOfGridPoints().

Die Methode makeGrid() arbeitet wie folgt. Der übergebene Startpunkt wird als erster Punkt ins Raster eingefügt. Der nächste Punkt wird aus der Punktmenge der Oberfläche im Abstand gridDist zum Raster gewählt. Für alle nachfolgenden Punkte wird, aus dem Schnitt von der Oberfläche im Abstand gridDist zum Raster und der Oberfläche im Abstand gridDist zur Menge des aktuellen gewählten Punktes, ein Punkt ausgewählt. Der aktuelle gewählte Punkt, ist der älteste Punkt im Raster, der mit diesem gemeinsame Oberflächenpunkte hat. Dies wird solange wiederholt bis gridPoints Punkte im Raster sind oder die Oberfläche leer ist. Das so erzeugte Raster wird zurückgegeben und als Universumsraster verwendet.

Die Oberfläche O zu einer Punktmenge M im Abstand a, ist dabei die Menge aller Punkte, welche zu einem Punkt der Punktmenge M den Abstand a haben

und zu keinem Punkt der Punktmenge M einen Abstand kleiner als a. ($O = \{P \mid \exists P_1 \in M \wedge a = n(P_1, P) \wedge \forall P_2 \in M \wedge \neg(a > n(P_2, P))\}$)

Beim Raster ist zu beachten, dass bei der Entwicklung der Raumuniversen (in der Raumuniversenfolge) nach und nach Punkte entfallen können, wenn das Universum beschnitten wird, oder bei Transitionen Punkte vom Rest des Rasters getrennt werden können.

Alle Rasterstatistiken bezogenen Daten werden in Dateien im Statistikverzeichnis (`statisticFolderName` Methoden: `getStatisticFolderName()` und `setStatisticFolderName()`) abgespeichert. Beim Setzen des Statistikverzeichnisses ist mit etwas Vorsicht vorzugehen, da eventuelle Dateien automatisch geöffnet werden und, wenn Statistiken ausgegeben werden sollen, das Statistikverzeichnis automatisch erzeugt wird. Die Dateien für die Ausgabe der allgemeinen Statistiken werden, wenn die zugehörige Statistik ausgegeben werden soll, automatisch erzeugt.

4.5.2 Direkte Rasterstatistik

Methoden:
```
unsigned long getDirectStatistics();
void setDirectStatistics(unsigned long gDi);
char* getDirectBaseFileName();
void setDirectBaseFileName(char* dircBN)
```
Daten: `unsigned long getDirect; char *directBaseName`

Nr.	Klassenvariable	Standartparameterbeschreibung
27	`unsigned long getDirect`	Direkte Rasterstatistik ermitteln:
28	`char *directBaseName`	Dateinamensanfang der direkten Statistik:

Die direkte Statistik ist die einfachste der auf das Raster bezogenen Statistiken. Wenn `getDirect` (zugehörige Methoden: `getDirectStatistics()` und `setDirectStatistics()`) größer 0 ist, wird sie alle `getDirect` Ticks ermittelt und ausgegeben. Die Ausgabe erfolgt dabei in zwei Dateien. In einer Datei wird die detailierter Statistik für das aktuelle Raumuniversum ausgegeben und in einer zweiten Datei wird, für alle betrachteten Raumuniversen, eine Zusammenfassung für die einzelnen Raumuniversen Zeilenweise ausgegeben. Beide Dateien werden in das Statistikverzeichnis (`statisticFolderName`) abgespeichert. Jeder Eintrag in den Dateien wird von einem Semikolon „;“ gefolgt.

Die Dateinamen der detailierten Statistik beginnen immer mit der Zeichenkette `directBaseName` (zugehörige Methoden: `getDirectBaseFileName()` und `setDirectBaseFileName()`), dann folgt die Zeit für das aktuelle Raumuniversum mit einem vorgestellten „U“ und am Ende die Endung „.txt“, ein Beispiel dafür ist „direktU13.txt“. In der Datei kommt nach einem Tabellenkopf die Daten für die einzelnen Punkte im Raster. In einer Zeile kommt dabei zuerst der Punktname auf dem sich die Daten der Zeile beziehen, dann die Anzahl der direkten Nachbarpunkten, aller, der negativen und der positiven ausgehenden Distanzen,

am Ende folgt der Unterschied zwischen den positiven und negativen Distanzen ((Anzahl positive Distanzen) - (Anzahl negative Distanzen)). In den letzten drei Zeilen stehen die Daten für den Durchschnitt, das Maximum und das Minimum aller Punkte.

Die Datei, in der die Zusammenfassungen für alle Raumuniversen gespeichert wird, beginnt mit `directBaseName`, dann folgt „UAll.txt“. In der Datei steht, nach dem Tabellenkopf, für jedes Raumuniversum, für das die direkte Statistik ermittelt wurde, eine Zeile. Die Zeile beginnt jeweils mit der Zeit für das Raumuniversum gefolgt von der Anzahl der Punkte im Raster. Danach folgen drei Blöcke nacheinander für die Durchschnitts-, Maximal- und Minimalwerte. Jeder Block besteht aus vier Einträgen. Der erste Eintrag ist dabei jeweils für alle direkten Nachbarpunkte der Punkte im Raster, dann folgen nacheinander die Einträge für die Anzahl aller, der negativen und der positiven ausgehenden Distanzen, am Ende folgt der Unterschied zwischen den positiven und negativen Distanzen.

4.5.3 Nachbarschaftsstatistik

Methoden:
```
unsigned long getNeibourStatistics();
void setNeibourStatistics(unsigned long gNeib);
unsigned long getNeibourDistance();
void setNeibourDistance(unsigned long neibD)
char* getNeibourBaseFileName();
void setNeibourBaseFileName(char* neibBN)
```
Daten:
```
unsigned long getNeib; unsigned long neibDist
char *neibBaseName;
```

Nr.	Klassenvariable	Standartparameterbeschreibung
29	`unsigned long getNeib`	Nachbarschaftsstatistik ermitteln:
30	`char *neibBaseName`	Dateinamensanfang der Nachbarschaftsstatistik:
31	`unsigned long neibDist`	Entfernung von den Rasterpunkten für die Nachbarschaftsstatistik:

Bei der Nachbarschaftsstatistik wird die Nachbarschaft bzw. die Umgebung der einzelnen Rasterpunke untersucht. Zur Nachbarschaft eines Punktes gehören dabei alle Punkte bis zum Abstand (siehe [1]) `neibDist` (zugehörige Methoden: `getNeibourDistance()` und `setNeibourDistance()`) um den Punkt. Die ausgegebenen Statistiken sind die gleichen wie für die direkte Statistik (aus dem vorhergehenden Abschnitt 4.5.2), nur beziehen sie sich auf die Punkte und Distanzen in der Nachbarschaft im Abstand `neibDist` und nicht nur um die direkten Nachbarschaft (Abstand $= 1$).

Wenn `getNeib` (zugehörige Methoden: `getNeibourStatistics()` und `setNeibourStatistics()`) größer 0 ist, wird die Nachbarschaftsstatistik alle `getNeib` Ticks ermittelt und ausgegeben. Die Ausgabe erfolgt dabei in zwei

Dateien. In einer Datei wird die detailierte Statistik für das aktuelle Raumuniversum ausgegeben und in einer zweiten Datei wird für alle betrachteten Raumuniversen eine Zusammenfassung für die einzelnen Raumuniversen Zeilenweise ausgegeben. Beide Dateien werden in das Statistikverzeichnis (`statisticFolderName`) abgespeichert. Jeder Eintrag in den Dateien wird von einem Semikolon „;“ gefolgt.

Der Dateiname der detailierten Statistik beginnt mit der in `neibBaseName` (die dazugehörige Methoden sind hierbei: `getNeibourBaseFileName()` und `setNeibourBaseFileName()`) gespeicherten Zeichenkette, dann folgt die Zeit für das aktuelle Raumuniversum mit einem vorgestellten „U“ und am Ende die Endung „.txt“, ein Beispiel dafür ist „neibU13.txt“. In der Datei kommt nach einem Tabellenkopf die Daten für die einzelnen Punkte im Raster. In einer Zeile kommt dabei zuerst der Punktname auf dem sich die Daten der Zeile bezieht, dann die Anzahl der Punkte, der negativen und der positiven Distanzen, am Ende folgt der Unterschied zwischen den positiven und negativen Distanzen ((Anzahl positive Distanzen) - (Anzahl negative Distanzen)) in der Nachbarschaft im Abstand `neibDist`. In den Letzten drei Zeilen stehen die Daten für den Durchschnitt, das Maximum und das Minimum aller Rasterpunkte.

Die Datei, in der die Zusammenfassungen für alle Raumuniversen gespeichert wird, beginnt mit `neibBaseName` dann folgt „UAll.txt“. In der Datei steht, nach dem Tabellenkopf, für jedes Raumuniversum, für das die Nachbarschaftsstatistik ermittelt wurde, eine Zeile. Die Zeile beginnt mit der Zeit für das Raumuniversum gefolgt von der Anzahl der Punkte im Raster. Danach folgen drei Blöcke nacheinander für die Durchschnitts-, Maximal- und Minimalwerte. Jeder Block besteht aus vier Einträgen. Der erste Eintrag ist dabei jeweils für alle Nachbarpunkte im Abstand `neibDist` der Punkte im Raster, dann folgen nacheinander die Einträge für die Anzahl der negativen und der positiven ausgehenden Distanzen, am Ende folgt der Unterschied zwischen den positiven und negativen Distanzen.

4.5.4 Abstandsstatistik

Methoden:
```
unsigned long getDistanceStatistics();
void setDistanceStatistics(unsigned long gDist)
char* getDistanceBaseFileName();
void setDistanceBaseFileName(char* distBN)
```
Daten: `unsigned long getDist; char *distBaseName`

Nr.	Klassenvariable	Standartparameterbeschreibung
32	`unsigned long getDist`	Abstandsstatistik ermitteln:
33	`char *distBaseName`	Dateinamensanfang der Abstandsstatistik:

Mit der Abstandsstatistik werden die Abstände zwischen den einzelnen Rasterpunkten ermittelt.

Wenn `getDist` (die zugehörige Methoden: `getDistanceStatistics()` und `setDistanceStatistics()`) größer 0 ist, wird die Abstandsstatistik alle

`getDist` Ticks ermittelt und ausgegeben. Die Ausgabe erfolgt dabei in zwei Dateien. In einer Datei wird die detailierter Statistik für das aktuelle Raumuniversum ausgegeben und in einer zweiten Datei wird für alle betrachteten Raumuniversen eine Zusammenfassung für die einzelnen Raumuniversen Zeilenweise ausgegeben. Beide Dateien werden in das Statistikverzeichnis (`statisticFolderName`) abgespeichert. Jeder Eintrag in den Dateien wird von einem Semikolon „;“ gefolgt.

Der Dateiname der detailierten Statistik beginnt mit der in `distBaseName` (die zum Parameter gehörigen Methoden sind: `getDistanceBaseFileName()` und `setDistanceBaseFileName()`) gespeicherten Zeichenkette, dann folgt die Zeit für das aktuelle Raumuniversum mit einem vorgestellten „U“ und am Ende die Endung „.txt“, ein Beispiel dafür ist „distU13.txt“.

Die Daten in der Datei sind in drei Blöcke gegliedert.

Der erste Block enthält die Abstände der Rasterpunkte zueinander. Dafür stehen dort in der ersten Zeile und Spalte die einzelnen Namen der Rasterpunkte. In den restlichen Zellen steht jeweils der Abstand den die beiden Rasterpunkte zu dieser Spalte und Zeile besitzen.

Der zweite Block enthält jeweils nacheinander eine Spalte für den maximalen und durchschnittlichen Abstand, den die Rasterpunkte zu dem Rasterpunkt dieser Zeile haben.

Im dritten Block werden die Distanzen, welche die Rasterpunkte zu dem Punkt dieser Zeile haben, noch einmal geordnet aufgeführt.

Zwischen dem ersten und zweiten Block gibt es eine Freispalte und zwischen dem zweiten und dritten Block gibt es zwei Freispalten (in der Datei stehen dort jeweils zwei „;;“ oder drei „;;;“ Semikolons direkt hintereinander).

Im Tabellenkopf steht in der erste Spalte „name“, danach folgen die Namen der einzelnen Rasterpunkte, dann „;;; maximum; average ;;;dist sorted : ; number of grid points :“ und danach folgt die Anzahl der Punkte im Raster.

Beispiel für die detailierte Statistik:

```
name; 4 ;17 ;19 ;; maximum; average ;;;dist sorted : ;
number of grid points :3 ;
4 ;0;3;3; ;3 ;2 ;;;0;3;3;
17 ;3;0;6; ;6 ;3 ;;;0;3;6;
19 ;3;6;0; ;6 ;3 ;;;0;3;6;
```

Die Datei, in der die Zusammenfassungen für alle Raumuniversen gespeichert wird, beginnt mit `distBaseName` dann folgt „UAll.txt“. In der Datei steht, nach dem Tabellenkopf, für jedes Raumuniversum, für das die Abstandsstatistik ermittelt wurde, eine Zeile. Die Zeile beginnt mit der Zeit für das Raumuniversum gefolgt von der Anzahl der Punkte im Raster. Danach folgen drei Spalten für den durchschnittlichen Abstand aller Rasterpunkte zu allen Rasterpunkten, dann für den maximalen Abstand zwischen allen Rasterpunkte und als letztes für den kleinsten der zweit kleinsten Abstände zwischen den Rasterpunkten (der kleinste Abstand ist immer 0).

4.6 Statistik über die Raumuniversenteile

Methoden: `unsigned long getOutUniversPartStatistic();`
`void setOutUniversPartStatistic(unsigned long uniPart);`
`char* getUniversPartFileName();`
`long setUniversPartFileName(char* uniPartF)`
Daten: `unsigned long lOutUniversPartStatistic;`
`char* szUniversPartStatisticFileName;`
`fstream *universPartStatisticFile;`
`bool bUniversPartStatisticParameterChanged;`

Nr.	Klassenvariable	Standartparameterbeschreibung
34	`unsigned long`	Teiluniversumsstatistik ausgeben:
34	`lOutUniversPartStatistic`	
35	`char* cutFileName`	Teiluniversumsstatistik Dateiname:

Zu den zusammenhängenden Teilraumuniversen des Raumuniversums kann eine Statistik in eine Datei ausgegeben werden.

Wenn der Parameter `lOutUniversPartStatistic` (die Methoden sind: `getOutUniversPartStatistic()` und `setOutUniversPartStatistic()`) größer 0 ist, wird alle `lOutUniversPartStatistic` Ticks/Iterationen die Statistik über die zusammenhängenden Teilraumuniversen in die Datei mit dem in `szUniversPartStatisticFileName` (`getUniversPartFileName()` und `setUniversPartFileName()`) stehenden Pfad ausgegeben. Die erste Zeile beinhaltet den Tabellenkopf. Danach werden die Werte für das aktuelle Raumuniversum und der Teilraumuniversen jeweils in eine neue Zeile geschrieben, wobei hinter jedem Wert ein Semikolon „;“ kommt.

Zu diesen Werten gehören, der Reihenfolge nach, die Zeit (interne Zeit plus Zeitbasis), dann kommt die Anzahl der Punkte, positiven und negativen Distanzen des Raumuniversums, danach folgt die Anzahl der zusammenhängenden Raumuniversumsteile und am Ende folgt eine Liste mit den Daten zu den einzelnen Raumuniversumsteilen. In dieser Liste steht, nach der Größe bzw. Anzahl von Punkten geordnet, für jeden zusammenhängende Raumuniversumsteil ein Klammerausdruck. Zwischen den Klammern für einen Raumuniversumsteil steht der Reihe nach, durch Komma „,“ getrennt, die Anzahl der Punkte, positiven und negativen Distanzen in ihm.

Beispiel für eine Zeile der Teiluniversumsstatistik:

```
54; 11; 4; 5; 3; ( 6, 2, 4 ) ( 4, 2, 1 ) ( 1, 0, 0 ) ;
```

Die ausgegebenen Daten sind identisch zur Beschneidungsstatistik (siehe 4.4.8 auf Seite 18) und sie wird nach dem Beschneiden des Raumuniversums erstellt (dann gibt es nur ein zusammenängendes Teilraumuniversum).

4.7 Standartparameter

Die Standartparameter sind in der Datei „standart_Parameter.txt“ zu finden. Diese wird geladen, wenn beim Start des Simulators keine Parameterdatei angegeben wird.

In der Datei „Null_Parameter.txt“ ist ein Parametersatz angegeben, der keine Nebenwirkungen hat, bei dem also nicht automatisch Ausgabedateien erzeugt werden. Diese Datei sollte nicht verändert werden. Sie liefert die Anfangswerte für den Simulator, die dann später durch spezifische Werte aus einer anderen Parameterdatei überschrieben werden können. Wenn in einer Paramerterdatei für einen Parameter keine Angaben bzw. Zeile existiert, kommen die Werte der Datei „Null_Parameter.txt“ zum Einsatz.

5 Klassenbeschreibung

Die Systemimplementation umfasst drei Klassen: `Univers`, `UniversTools` und `UniversNeib`.

Die Klasse `Univers` enthält den Basissimulator. In ihr sind die Methoden zur Erzeugung von Raumuniversenfolgen und für einfachen Statistiken für die Raumuniversen implementiert.

In der Klasse `UniversTools` sind Methoden implementiert mit denen Erzeugung und Auswertungen von Raumuniversen und Punktstrukturen möglich sind. Dazu gehören Beispielsweise, dass Erzeugen eines Rasters, den Durchschnitt zweier Punktmengen ermitteln oder die Berechnung der Entfernung zweier Punkte. `UniversTools` erbt von `Univers`.

Die Klasse `UniversNeib` erbt von der Klasse `UniversTools` und dient zur Ermittlung weiterer Statistiken. Mit ihr können insbesondere die Rasterstatistiken für die Raumuniversenfolge ausgegeben werden.

6 Unterschiede der Simulatoren

Die drei Simulatoren unterscheiden sich untereinander nur in der Implementation der Methoden `tick()` und `evalueTransition()`, sowie durch die implementation der Klasse `TPoint`. Die TPoint implementation für den Simulator univers ist `TPoint.cpp`, für universD ist `TPointD.cpp` und für universP ist `TPointP.cpp`. Durch die Definition des jeweiligen Makros `UNIVERS`, `UNIVERS_D` und `UNIVERS_P` kann bei der Komplierung der Klasse `Univers` der jeweilige Simulator erzeugt werden. Dann muss noch die richtige TPoint Implementation/das richtige TPoint Objekt hinzugelinkt werden.

6.1 Unterschiede zwischen Linux und Windows

Die Windows-Versionen der Simulatoren unterscheiden sich von den Linux-Versionen dadurch, dass die Eingaben über Tastatur unter Linux mithilfe der `keyboard`-

Klasse geschieht und unter Windows mit Windows spezifischen Funktionen aus `conio.h`. Dadurch fehlen unter Windows die „keyboard.*“ Dateien und in der Datei „UniversNeib.cpp“ wird die Eingabe umgestellt. Diese Umstellung in „UniversNeib.cpp“geschieht dadurch, dass `conio.h` eingebunden/includiert und die `keyboard` Instanz gelöscht wird (`keyboard kb;` und `kb.` in der `start()`-Methode). (Die Umstellung geschieht automatisch durch `#ifdef` .)

7 Implementierungskonventionen

Im Nachfolgendem sind einige Implementierungskonventionen aufgeführt. Da sie erst im Nachhinein aufgestellt wurden, sind sie teilweise in der tatsächlichen Implementation noch nicht umgesetzt.

7.1 Allgemeines

Zur Dokumentierung des Quelltextes wird **doxygen** im Java Stil verwendet.

Nur Kommentare die sich direkt auf spezifische Codeabschnitte beziehen erfolgen nicht im doxygen Stil.

Beim Quellcode wird großer Wert auf Verständlichkeit und Übersichtlichkeit gelegt.

7.2 Einleitung des Quelltextes

Jede Quelltextdatei beginnt Informationsheader. In ihm steht der Name der Datei, des Autors und das Erstellungsdatum der Datei, gefolgt einer Kurzbeschreibung und den Kopierrechten(Copyright). Nach den Kopierrechten wird der Inhalt der Datei näher beschrieben.

Am einde der Dateieinleitung folgt ein Änderungsprotokoll, welches nicht in doxygen Stiel geschrieben wird.
Beispiel:

```
/**
 * @class cDomains
 * file name: cDomains.cpp
 * @author Betti Oesterholz
 * @date 09.06.2009
 * @mail webmaster@BioKom.info
 *
 * System: C++
 *
 * This class represents a list of domains.
 * Copyright (C) @c LGPL3 2009 Betti Oesterholz
 *
 * This program is free software: you can redistribute it and/or modify
```

```
 * it under the terms of the GNU Lesser General Public License (LGPL) as
 * published by the Free Software Foundation, either version 3 of the
 * License, or any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public Licen
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 *
 *
 * This class represents a list of domains for an root -element.
 * The domain list consists of a number of domains with a type.
 *
 */
/*
History:
09.06.2009  Oesterholz  created
12.07.2009  Schmidt     new method to get the size of an domainelement
*/
```

7.3 Formatierung

Gearbeitet wird mit Tabulatoren. Jeder Block, außer Methodenimplementationen und einzeilige Blöcke, wird durch ein Tab eingerückt.

Die einzelenen Elemente in einer zeile werden nach möglichkeit durch Leerzeichen seperiert. Die Länge einer Zeile sollte möglichst 75 Zeichen nicht überschreiten.
Beispiel:

```
if ( a == b ){
//it makes something
do_it;
call( parameter );
}
```

7.4 Kommentare

Alle Kommentare sind in Englisch gehalten.

Jede Methode beginnt mit einem Informationsheader im **doxygen** im Java Stil, in dem eine kurze Beschreibung der Methode steht.

Kommentare zu eine Quellcodesegment stehen in einer Zeile direkt vor diesem.

7.5 Namen

Namen sind Bezeichner von Klassen, Methoden, Variablen, Konstanten oder Makros.

7.5.1 Klassennamen

Alle Klassennamen beginnen mit einem Kleinen „c“ für „class“, die Teilworte des Klassennamen beginnen mit einem großen Buchstaben. (Camelcase) Beispiel: cUnivers; cUniversNeib

7.5.2 Methodennamen

Methoden beginnen mit einem Kleinbuchstaben. Teilnamen im Methodennamen beginnen mit einem großen Buchstaben und grenzen unmittelbar aneinander. (Camelcase)

Methoden die dazu dienen Werte zurückzugeben beginnen mit einem „get“, demgegenüber beginnen Methoden mit denen Werten gesetzt werden sollen mit einem „set“.
Beispiel:

- eineMethode();
- getIhrgendWas();

7.5.3 Variablennamen

Variablennamen beginnen kleingeschriebenen Teilnamen im Variablennamen beginnen mit einem großen Buchstaben und grenzen unmittelbar aneinander. (Camelcase) Namen von Variablen von grundlegenden Datentypen, wie Beispielsweise `int`, beginnen mit einem Kürzel für diesen Datentypen (Siehe Tabelle 1). Im Allgemeinen sollte am Variablennamenanfang der Typ der Variable erkennbar sein. Beispiel: variablenNamen, uiGanzeZahl

- variablenNamen
- uiGanzeZahl;

8 Zusatztools

8.1 Raumuniversen erzeugen

8.1.1 cutUniversPart

Das Tool cutUniversPart ist in einer eigenen Implementation realisiert. Nachdem der zugehörige Quelltext heruntergeladen wurde, kann er durch „./make“ kompiliert und durch „cut“ gestartet werden.

Kürzel	Datentyp
i	int
s	shor
c	char
l	long
u	unsigned
p	pointer / Zeiger
sz	string / Zeichenketten
li	list / Listen
vec	vector / Vektoren
str	stream / Datenstrom

Tabelle 1: Prefixe für Datentypen

Das Tool cutUniversPart dient dazu, aus einem existierenden Raumuniversum einen Teil bzw. Bereich auszuschneiden und in ein eigenes Raumuniversum umzuwandeln. Dazu wird um einem Punkt als Mittelpunkt bis zu einem vorgegebenen Abstand alle sein Nachbarpunkte zu dem Bereich hinzugefügt, dieser wird dann zu einem neuen Raumuniversum transformiert.

Nach dem Start von „cut“ wird der Benutzer zuerst aufgefordert den Pfad für das Raumuniversum anzugeben aus dem ein Teil ausgeschnitten werden soll, dieses wird dann geladen. Danach wird ein Punktname (bzw. Punktnummer) abgefragt, der als Mittelpunkt dienen soll, als nächstes wird dann der Abstand angegeben, bis zu welchem ein Bereich um den Punkt ausgeschnitten werden soll. Dann wird der Pfad abgefragt unter dem der bearbeitete Teil des Raumuniversums am Ende abgespeichert werden soll. Nun kann noch der Bearbeitungsmodus gewählt werden.

Im den Modi 0 wird gleich der Teil in ein Raumuniversum umgewandelt. Bei den Modi 1 bis 4 werden vorher noch die Distanzen der Arten verändert. Im Modi 1 bzw. 2 werden alle negativen bzw. positiven Distanzen aus dem Bereich gelöscht. Im Modi 3 bzw. 4 werden alle negativen bzw. positiven Distanzen in dem Bereich durch positiven bzw. negativen Distanzen ersetzt, negativen bzw. positiven Distanzen werden also in die andere Distanzart umgewandelt. Im Modi 5 werden nur die Statistiken (Anzahl Punkte, positive und negative Distanzen) über die Größe des ausgeschnittenen Teils und seiner Oberfläche (Punkte und Distanzen die sowohl mit Punkten im wie außerhalb des Teils verbunden sind) ausgegeben, ohne das der Teil zu einem neuen Raumuniversum gemacht wird.

Bei der Umwandlung des Teils in ein eigenes Raumuniversum werden Distanzpaare gesucht, die beide nur einen Endpunkt im Teil haben, die Endpunkt im Teil müssen verschieden sein, das Distanzpaare muss der gleichen Art (positiv oder negativ) zugehören und deren Endpunkte im Teil möglichst genauso viele positive und negative Distanzen haben, wie der jeweils andere Endpunkt der nicht im Teil liegt. Diese Distanzpaare werden dann durch ein Distanzpaar der gleichen Art ersetzt, wobei nun aber die beiden Punkte im Bereich durch eine Distanz verbunden

wird. Dies wird solange wiederholt wie es solche Distanzpaare gibt, die restlichen Distanzen die nur einen Endpunkt im Bereich haben werden gelöscht. Am Ende sind alle Punkte des Teils nur mit Punkten des Teils verbunden und der Teil ist ein eigenständiges Raumuniversum.

Dieses Raumuniversum wird dann ausgegeben.

Weiterhin werden im Laufe der Bearbeitung noch weitere Statistiken ausgegeben. Dazu gehören die Anzahl der Punkte, positiven und negativen Distanzen im Teil vor und nach der Bearbeitung. Die Anzahl der Punkte in der Oberfläche des Bereichs, wobei die Oberfläche all die Punkte enthält, die Distanzen besitzen welche nur einen Endpunkt im Bereich haben. Weiterhin wird ausgegeben, wie viele positive und negative Distanzen bei der Bearbeitung gelöscht oder versetzt wurden, sowie wie groß die Summe der Unterschiede der Anzahlen der positiven und negativen Distanzen der Punkte im Teil war, zwischen denen die Distanzen versetzt wurden.

9 GNU Free Documentation License

Eine deutsche Übersetzung der „GNU Free Documentation License“ ist unter `de.wikipedia.org/wiki/GNU_FDL/Text` zu finden. Diese ist eine inoffizielle deutsche Übersetzung der GNU Free Documentation License. Sie ist nicht von der Free Software Foundation herausgegeben und erläutert nicht die Bedingungen der GNU FDL – Dies tut nur der nachfolgend original englische Text der GNU FDL.

PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document „free“ in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of „copyleft“, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The „Document“, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as „you“. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A MModified Versionöf the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A SSecondary Sectionïs a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The „Invariant Sections“ are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The „Cover Texts“ are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A „Transparent“ copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not „Transparent“ is called „Opaque“.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The „Title Page“ means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, „Title Page“ means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section „Entitled XYZ“ means a named subunit of the Docu-

ment whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as „Acknowledgements“, „Dedications“, „Endorsements“, or „History“.) To „Preserve the Title“ of such a section when you modify the Document means that it remains a section „Entitled XYZ“ according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition.

Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled „History“, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled „History“ in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the „History“ section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled „Acknowledgements“ or „Dedications“, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled „Endorsements“. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled „Endorsements“ or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled „Endorsements“, provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as

Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled „History“ in the various original documents, forming one section Entitled „History“; likewise combine any sections Entitled „Acknowledgements“, and any sections Entitled „Dedications“. You must delete all sections Entitled „Endorsements“.

COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an „aggregate“ if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled „Acknowledgements“, „Dedications“, or „History“, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version“ applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (c)  YEAR  YOUR NAME.

Permission is granted to copy, distribute
and/or modify this document under the
terms of the GNU Free Documentation License,
Version 1.2 or any later version published
by the Free Software Foundation; with no
Invariant Sections, no Front-Cover Texts,
and no Back-Cover Texts. A copy of the
license is included in the section entitled
"`GNU Free Documentation License"'.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the „with...Texts.“ line with this:

```
with the Invariant Sections being LIST
THEIR TITLES, with the Front-Cover
Texts being LIST, and with the
Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Literatur

[1] Betti Österholz, Wilde Raumzeit

Index