Aufforstung Mischwald

IHK - Abschlussprüfung

Mathematisch-technischer Softwareentwickler

Anton Herzog

IHK-Nummer 107

Prüflingsnummer 95976

Inhaltsverzeichnis

1 Aufgabenanalyse ..... - 5 -
2 Programmkonzeption ..... - 7 -
2.1. Zentraler algorithmischer Ansatz ..... - 7 -
2.2 Verbale Verfahrensbeschreibung ..... - 8 -
2.3. Abweichungen von erster Programmkonzeption ..... - 9 -
2.4. Technische Produktumgebung ..... - 11 -
2.5. UML – Klassendiagramm ..... - 12 -
2.6. UML Use-Case-Diagramm ..... - 13 -
2.7. UML Sequenzdiagramm ..... - 13 -
2.8. Nassi_Schneidermann Diagramme wichtiger Methoden ..... - 15 -
3 Benutzerhandbuch ..... - 17 -
3.1 Starten des Programms ..... - 17 -
3.2 Muster Eingabedatei ..... - 17 -
3.3 Weiterverarbeiten der Ausgabedatei ..... - 18 -
4 Verifikation ..... - 19 -
4.1 Übersicht - Testfälle ..... - 19 -
4.2 Diskussion der Testergebnisse ..... - 19 -
5 Zusammenfassung und Ausblick ..... - 37 -
5.1 Zusammenfassung ..... - 37 -
5.2 Mögliche Erweiterungen ..... - 38 -
6 Abbildungsverzeichnis ..... - 39 -
7 Anhang(Quellcode) ..... - 41 -
7.1 Klasse Program ..... - 41 -
7.2 Klasse FileReader ..... - 43 -
7.3 Klasse FileValidator ..... - 44 -
7.4 Klasse FileWriter ..... - 46 -
7.5 Klasse Utils ..... - 48 -

7.6 Enum ValidationResult .....- 49 -
7.7 Klasse Baum.....- 50 -
7.8 Klasse Baumart.....- 51 -
7.9 Klasse Position.....- 52 -
7.10 Klasse Simulation .....- 53 -
8 Eigenhändigkeitserklärung: .....- 61 -

1 Aufgabenanalyse

Zentraler Punkt der vorliegenden Aufgabe ist die Entwicklung eines Programmes mit deren Hilfe ein Rechteckiges Waldstück einer bestimmten Größe neu zu bepflanzen.
Hierzu ist weiterhin eine Menge von Baumarten gegeben, welche sich einzig im Radius ihrer kreisförmigen Grundfläche unterscheiden. Ein Baum wird hier also durch einen Kreis der Fläche abgebildet, die er für eine gute Entwicklung benötigt.
Weiterhin sind zweierlei Indizes gegeben, um das erhaltene Ergebnis einzuordnen bzw. zu bewerten. Einerseits ist dies der Simpson-Index(D), der die Diversität der vorhandenen Bäume bezüglich ihrer Art abbildet. Andererseits wird der Simpson-Index mit der prozentualen Abdeckung der Gesamtfläche Multipliziert(B).
Die Parameter für diese Simulation sind in einem vorgegebenen Format in Textdateien gegeben. Die Ausgabe soll in einer ebenfalls vorgegebenen Form geschehen. Diese beinhaltet den gegebenen Namen des Waldstückes, dessen Größe und eine Auflistung aller errechneten Bäume, repräsentiert durch deren Position, ihren Radius und ihren Artenindex aus der Eingabedatei. Es soll ein Gnuplot-Script generiert werden, um die Ergebnisse einfach visualisieren zu können.
Ein gegebener Lösungsansatz schlägt das Anordnen der Bäume auf Basis von Dreiecke an, welche durch die Mittelpunkte der abstrakten Baumflächen abgebildet werden. Durch diesen Ansatz hat man 2 Punkte und alle Seitenlängen des entstehenden Dreiecks vorliegen und kann durch wenige algebraische Operationen einen dritten Punkt, und dessen an der Gerade durch die gegebenen Mittelpunkte gespiegeltes Gegenstück, errechnen. Dieser Vorgang kann für daraus entstehende neue Baumpaare so oft wiederholt werden, bis keine validen neuen Positionen mehr vorhanden sind.

2 Programmkonzeption

2.1. Zentraler algorithmischer Ansatz

Mein Algorithmus stützt sich auf die Wirksamkeit der Generierung einer Startkonstellation, beschrieben in der Aufgabenstellung. Danach ist ein Baumpaar vorhanden, aus dessen Mittelpunkten sich zwei mögliche neue Positionen für einen weiteren Baum generieren lassen. Dieses Paar wird in einer Menge von Baumpaaren abgelegt.
Die Wahl welche Baumart man zuerst für die Konstruktion der neuen Position testet, wird durch eine Sortierung derer Menge vorgegeben. Hierzu wird erst aufsteigend nach der Anzahl der bereits errechneten Bäume der fraglichen Art sortiert und bei einem Gleichheitsfall absteigend nach deren Radien Bäume. Zusammengefasst wird also immer eine Baumart bevorzugt die möglichst selten vorhanden ist und eine möglichst große Fläche abdeckt.
Hat man eine solche valide Position für eine Art gefunden, wird der Baum positioniert. Anschließend werden der Menge der zu untersuchenden Baumpaare die Kombination des entstandenen Baumes mit seinen jeweiligen „Elternbäumen" hinzugefügt und das erzeugende Paar aus der Menge entfernt.
Dieser Vorgang wird nun so lange fortgesetzt bis keine Baumpaare mehr vorhanden sind.
Die Entscheidung welche Baumpaare zuerst genutzt werden um weitere Positionen zu ermitteln, ist in dieser Implementierung zufällig gestaltet, sodass in 50% der Fälle der neueste und in 50% der Fälle der älteste Knoten benutzt werden. Dies führ zu einer größeren Erfolgschance, da es die Wahrscheinlichkeit, dass der Algorithmus vorzeitig terminiert, dass sich die entstandenen Bäume gegenseitig verstellen, verringert.

Da dieser Fall immer noch auftritt und die zeitig terminierenden Fälle stets eine kurze Laufzeit haben, werden hier bis zu 10 Simulationen durchgeführt, bis eine davon wenigstens 50% der Gesamtfläche abdeckt.

Für den Normalfall der Aufgabenstellung hat dieser Ansatz eine gute Performance und ist sehr stabil. Sonderfälle werden im Abschnitt über die Testfälle diskutiert.

2.2 Verbale Verfahrensbeschreibung

Außerhalb des unter 2.1. beschriebenen Algorithmus ist die Programmstruktur sehr einfach. Hier wurde auf den Aspekt der funktionalen Trennung Wert gelegt, d.h. die Ein-/Ausgabe der Daten ist streng von der verarbeitenden Schicht, hier der Simulation, getrennt. Dadurch wird eine einfachere Wartbarkeit erreicht und die Verständlichkeit im Allgemeinen verbessert.
Ein- und Ausgabeformat waren durch die Aufgabenstellung vorgegeben und wurden so übernommen bzw., im Fall des Ausgabeformates, erweitert. (Laufzeit in Sekunden und D_{Max} wurden hinzugefügt um besser auswerten zu können)
Die Datenhaltung innerhalb des Algorithmus ist vergleichsweise simpel. Die Menge der bereits ermittelten Bäume wird in einer einfachen Liste vorgehalten. Ebenso die Menge aller Baumpaare, allerdings wird diese als eine Kombination aus Stack und Queue verwendet. Dieser wird zur leichten Randomisierung des Verfahrens und der damit verbunden Verbesserung der Erfolgschancen verwendet.

2.3. Abweichungen von erster Programmkonzeption

Funktionale Abweichungen in der Programmlogik sind die Optimierung der Validierung der möglichen neuen Pflanzungspositionen und die Randomisierung der Auswahl des nächsten zu untersuchenden Baumpaares.

Ersteres lässt sich dadurch begründen, dass im Vorfeld nicht klar war das die eigentlich äquivalente Umformung einen so gravierenden Performanceunterschied verursachen würde.

Ursprüngliche Version:

```
double abstand = Math.Sqrt(Math.Pow(pos.X -
 baum.Position.X, 2) + Math.Pow(pos.Y -
       baum.Position.Y, 2));
double diff = abstand - (radius + baum.Art.Radius);
```

Überarbeitete Version:

```
double abstandsQuadrat = (pos.X -
 baum.Position.X) * (pos.X baum.Position.X) +
                      (pos.Y - baum.Position.Y) * (pos.Y -
 baum.Position.Y);
double diff = abstandsQuadrat -
 (radius + baum.Art.Radius) * (radius + baum.Art.Radius);
```

Der rein mathematische Wert von „diff" hat sich hier nicht geändert, die Performance allerdings ist in etwa verdoppelt worden.

Die Randomisierung der Auswahl des nächsten zu untersuchenden Baumpaares verringert die Wahrscheinlichkeit, dass die Entwicklung der Baumstruktur des Waldstückes vorzeitig terminiert. Dies trat bei ungünstigen Verhältnissen zwischen den Radien der Baumarten auf. Realisiert ist diese Randomisierung durch folgende Konstruktion:

```
int index = _random.Next(0, 2) - 1;
if (index == -1)
{
    index = _zuTestendeBaumpare.Count - 1;
}
Tuple<Baum, Baum> untersuchtesTuple = _zuTestendeBaumpare[index];
```

_r ist hier vom Typ Random der pro Instanz des Typs Simulation auch nur einmal initialisiert wird und daher tatsächlich ausreichend zufällige Ergebnisse für diesen Anwendungsfall liefert.

Ebenfalls zur Qualitätssicherung bestimmt, ist die bis zu zehnmalige Durchführung der Simulation. Dies führt wie oben schon beschrieben zu einer höheren Erfolgschance, was die Suche nach „guten" Ergebnissen angeht. Die „schlechten" Simulationsläufe sind zu vernachlässigen, da ihre Laufzeit stets klein bleibt.

Eine weitere kleine Erweiterung das das Hinzufügen der Klasse Utils um eine Duplikation der Funktionalität zum Entfernen von Kommentarblöcken aus der Eingabedatei.

2.4. Technische Produktumgebung

Programmiersprache/Entwicklungsumgebung

Das zu entwickelnde Produkt wurde in der Sprache C# und in der Entwicklungsumgebung Visual Studio 2013 erstellt.

Software-/Hardwareanforderungen

Das Produkt ist für Windowsversionen ab Windows 7 geeignet und benötigt das .Net-Framework Version 4.0.
Downloadlink:
https://www.microsoft.com/de-de/download/details.aspx?id=17718

Zur Visualisierung der Ausgabedateien ist das Programm Gnuplot notwendig. Beides kann kostenfrei heruntergeladen werden. (Links im Anhang)
Downloadlink:
http://www.gnuplot.info/download.html

2.5. UML – Klassendiagramm

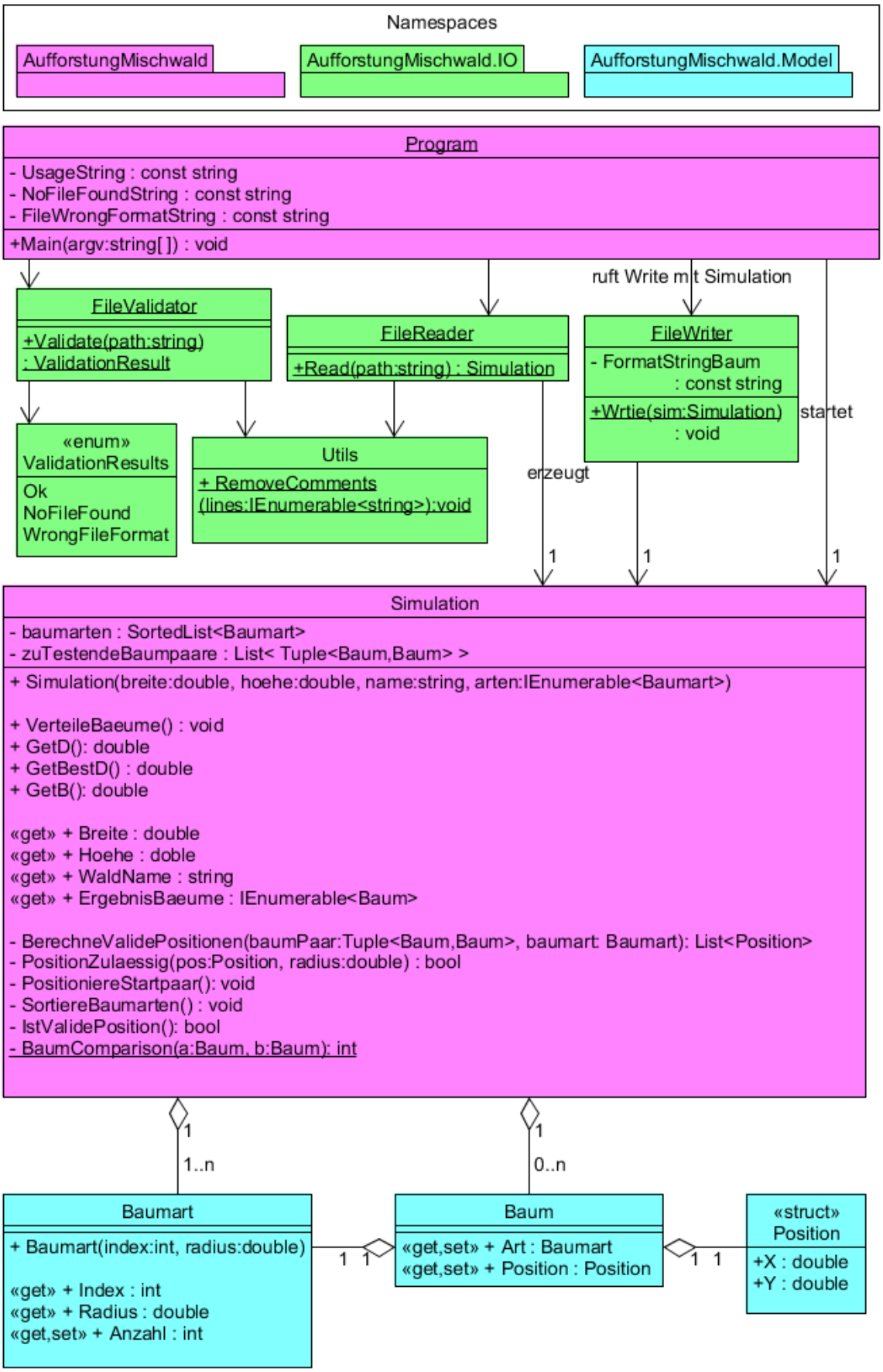

Abbildung 1 - UML-Klassendiagramm

2.6. UML Use-Case-Diagramm

Der Benutzer hat bei dieser Programmlösung die Möglichkeit die Eingabeparameter zu verändern und eine Simulation zu starten.

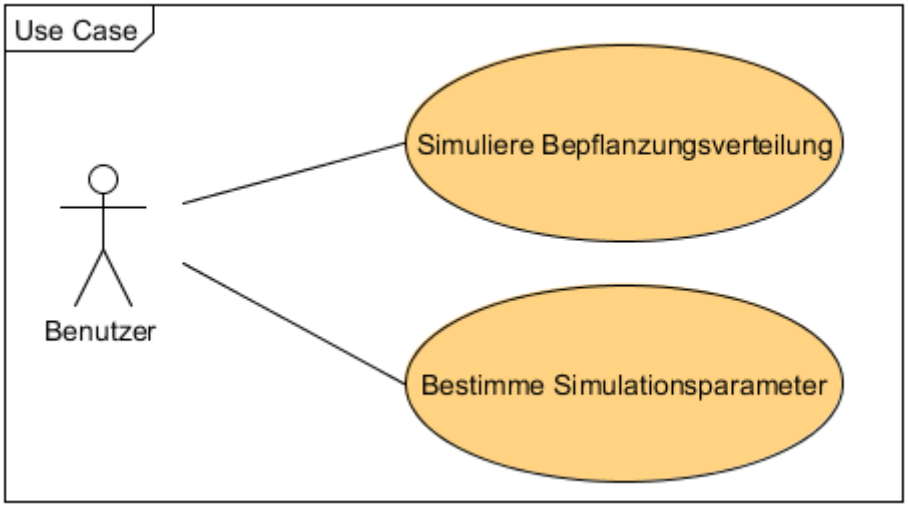

Abbildung 2 - UML Use Case Diagramm

2.7. UML Sequenzdiagramm

Das Sequenzdiagramm spiegelt den üblichen Programmablauf wieder. Die Modellklassen, welche von FileReader und Simulation benutzt werden sind hierbei vernachlässigt.

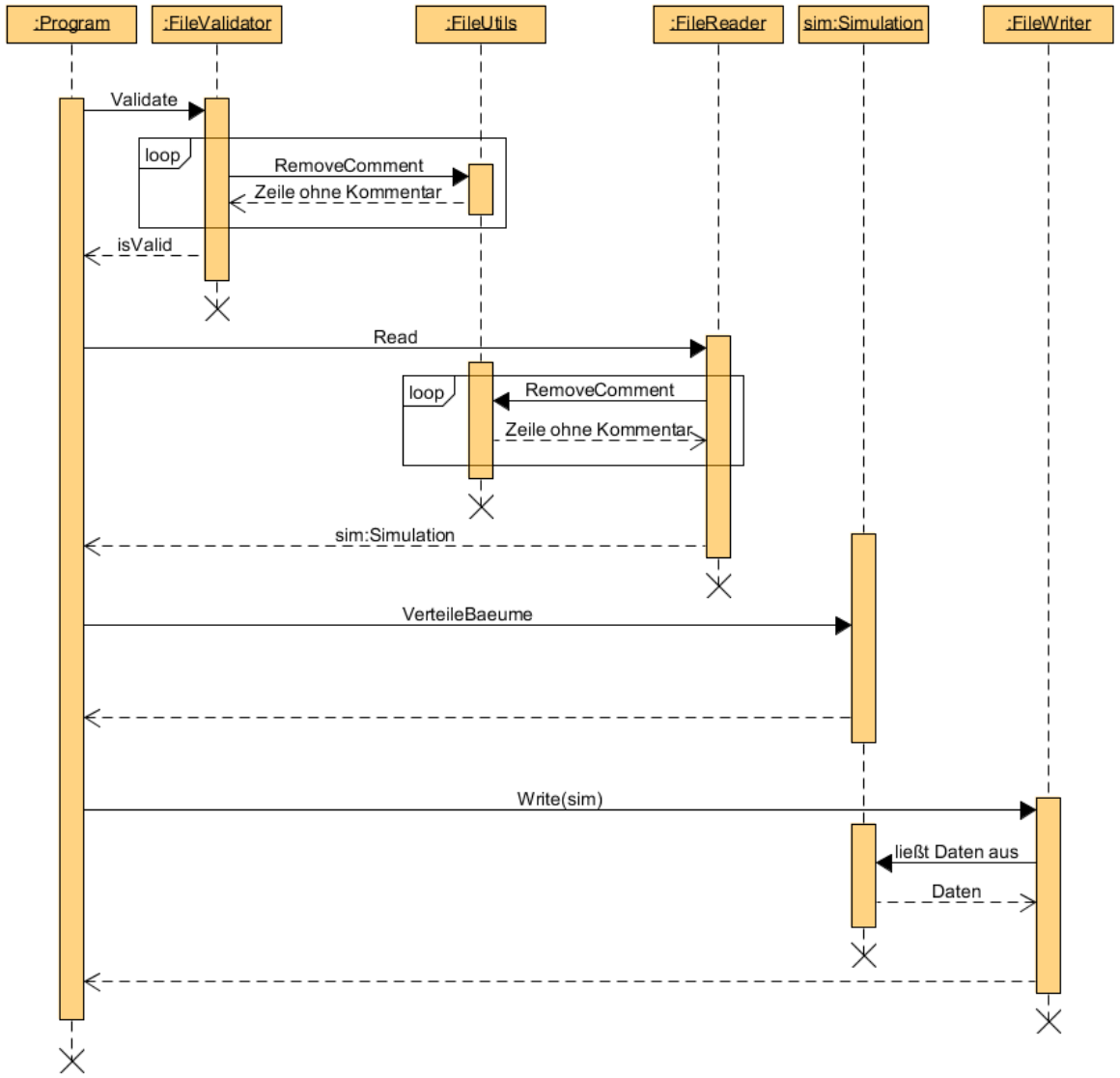

Abbildung 3 - UML Sequenzdiagramm

2.8. Nassi Schneidermann Diagramme wichtiger Methoden

2.8.1 Struktogramm Simulation.VerteileBaeume

Das folgende Struktogramm beschreibt die zentrale Simulationsmethode.

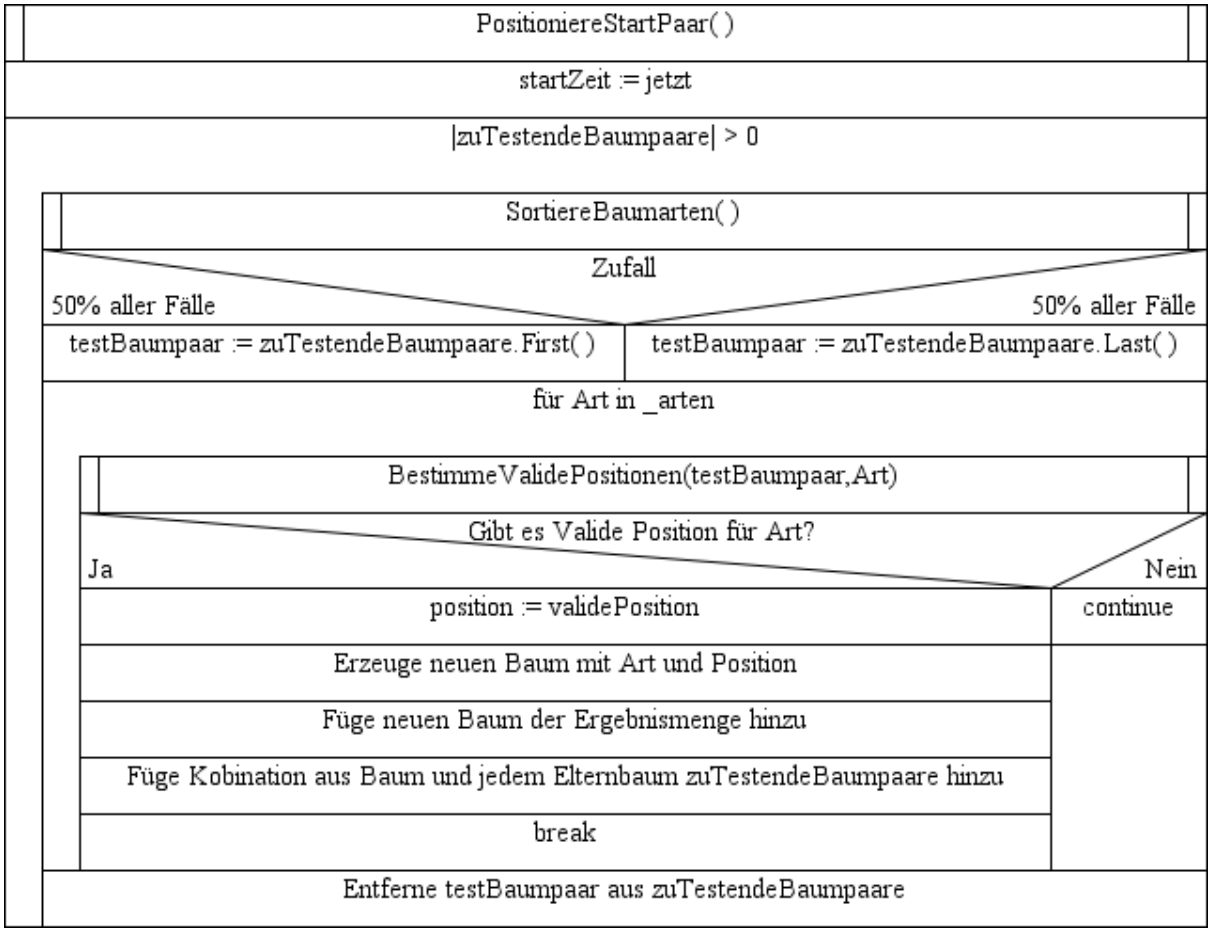

Abbildung 4 - Struktogramm Simulation.VerteileBaeume

2.8.2 Struktogramm IstValidePosition

Das folgende Struktogramm beschreibt die Validierung einer errechneten Baumposition gegen die Ausmaße des Waldstücks und die bereits vorhandenen Bäume.

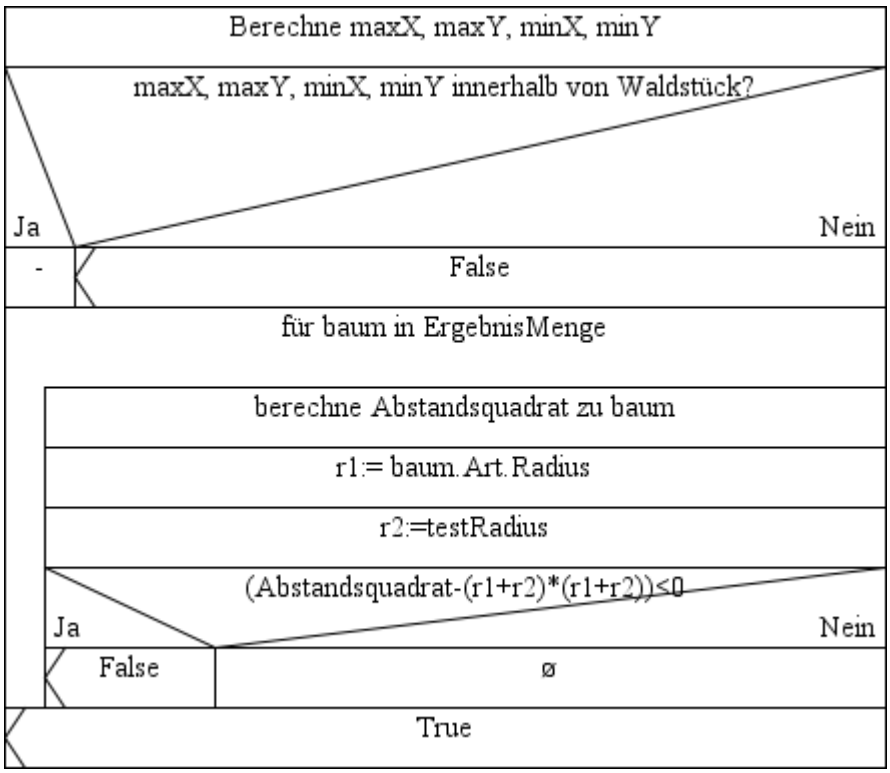

Abbildung 5 - Struktogramm IstValidePosition

3 Benutzerhandbuch

3.1 Starten des Programms

Das Programm wird über die Kommandozeile mit dem Aufruf „AufforstungMischwald.exe <Eingabedateipfad>" gestartet.
Alternativ kann das beigefügte Batch-Skript genutzt werden um das Programm mit allen Dateien des Typs „*.in" im aktuellen Verzeichnis auszuführen. Hierfür muss sich das Programm im gleichen Ordner wie Skript und Eingabedateien befinden.

3.2 Muster Eingabedatei

Eine Eingabedatei muss folgendem Schema entsprechen:

<Name des Waldstücks>
<Breite> <Länge> % <möglicher Kommentar>
<Radius Baumart1>
<Radius Baumart2>
<Radius Baumart3>
…
<Radius BaumartN>

Kommentare können hinter dem Zeichen „%" angefügt werden, aber erst nach den Daten der Zeile.

3.3 Weiterverarbeiten der Ausgabedatei

Das Programm liefert und dem Pfad „<Eingabepfad>.plt“ ein Gnuplot-Skript. Dieses Skript kann mit Hilfe des freien Programmes Gnuplot geladen werden und liefert eine visuelle Repräsentation der Simulationsergebnisse.

Der Befehl in Gnuplot lautet „load <Dateiname>“.

Das Ergebnis sieht z.B. wie folgt aus:

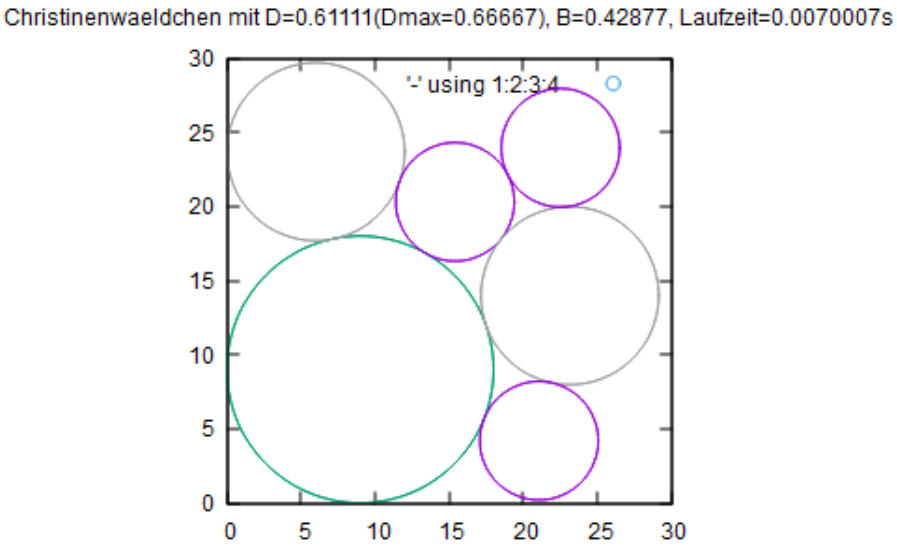

Abbildung 6 - Beispielausgabeplot

4 Verifikation

4.1 Übersicht - Testfälle

Nr.	Test-Name	Testinhalt	Erwartetes Ergebnis	Erfolgtes Ergebnis	Fazit
1	IHK-Beispiel 1	Kleines Feld 3 Arten	Gute Lösung	Gute Lösung	Korrekt
2	IHK-Beispiel 2	Großes Feld 5 Arten	Gute Lösung	Gute Lösung	Korrekt
3	Extremfall 1	Radien zu groß	Leerer Wald	Leerer Wald	Korrekt
4	Extremfall 2	Sehr flaches Feld 2 Radien	Ansatz greift nicht	Nur ein Baum	Erwartete Schwäche
5	Extremfall 3	Sehr schmales Feld 2 Radien	Ansatz greift nicht	Nur zwei Bäume	Erwartete Schwäche
6	Extremfall 4	Schmales Feld 4 Radien	Gute Lösung	Gute Lösung	Korrekt
7	Extremfall 5	Flaches Feld 4 Radien	Problem	3 Bäume	Erwartete Schwäche
8	Extremfall 6	Flaches Feld 6 Radien	Problem	10 Bäume	Erwartete Schwäche
9	Extremfall 7	Schmales Feld 6 Radien	Gute Lösung	Gute Lösung	Korrekt
10	Extremfall 8	Sehr großer Wald	Lange Laufzeit	Gute Lösung 18,5 sec.	Sehr Gut
11	Extremfall 9	2 Arten Verhältnis 1/20	Schlechtes Ergebnis	Schlechtes Ergebnis	Unrealistischer Fall
12	Falsches Format	Keine Länge	Fehlerausgabe Konsole	Fehlerausgabe Konsole	Korrekt
13	Keine Datei unter Pfad	Keine Datei	Fehlerausgabe Konsole	Fehlerausgabe Konsole	Korrekt
14	Normalfall 1	Großer Wald Viele Arten	Gute Lösung	Gute Lösung	Korrekt
15	Normalfall 2	Großer Wald Wenig Arten	Gute Lösung	Gute Lösung	Korrekt
16	Normalfall 3	Kleiner Wald Wenig Arten	Gute Lösung	Gute Lösung	Korrekt
17	Normalfall 4	Kleiner Wald Viele Arten	Gute Lösung	Gute Lösung	Korrekt

Die Testfälle sowie die jeweiligen Ausgabedateien sind im Ordner „Tests" zu finden. Durch die Randomisierung können erneute Testläufe ein anders Ergebnis haben.

4.2 Diskussion der Testergebnisse

Zur besseren Bildhaftigkeit, werden die Testergebnisse anhand der visuellen Repräsentation diskutieret. Die Ausgabedateien liegen vor.

4.2.1 IHK Beispiel 1

Eingabe:

```
Christinenwäldchen
30 30 %Länge und Breite in Metern
6      %Esche
4      %Erle
9      %Buche
```

Dies ist ein typisches Beispiel eines kleinen und quadratischen Waldstückes. Die Radien der Baummenge sind in einem realistischen Verhältnis zueinander.

Ausgabe:

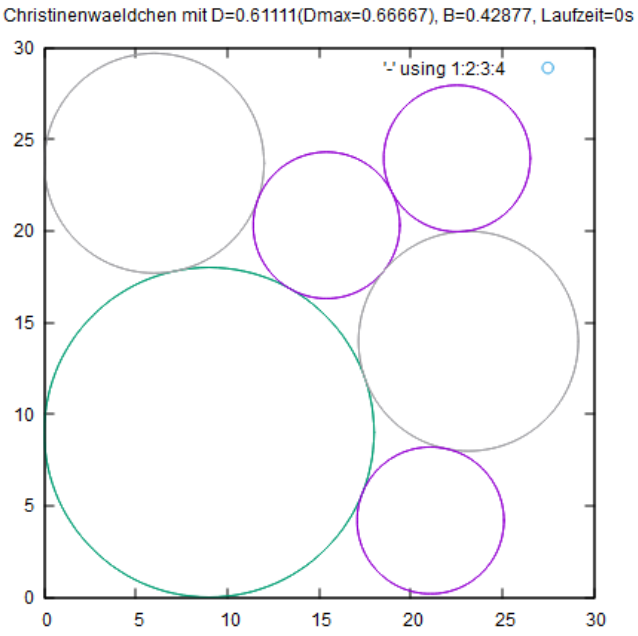

Abbildung 7 - IHK Beispiel 1

Auswertung:

Ein gutes Ergebnis, da die Diversität ihrem Optimum nahe kommt und eine gute Abdeckung erreicht wurde. Ein typischer günstiger Fall für den Algorithmus.

4.2.2 IHK Beispiel 2

Eingabe:

```
Snowdenforst
500 200      %Länge und Breite in Metern
12      %Eiche
6       %Esche
4       %Erle
9       %Buche
3       %Birke
```

Dies ist ein typisches Beispiel eines großen Waldstückes. Die Radien der Baummenge sind in einem realistischen Verhältnis zueinander und der Algorithmus sollte gut funktionieren.

Ausgabe:

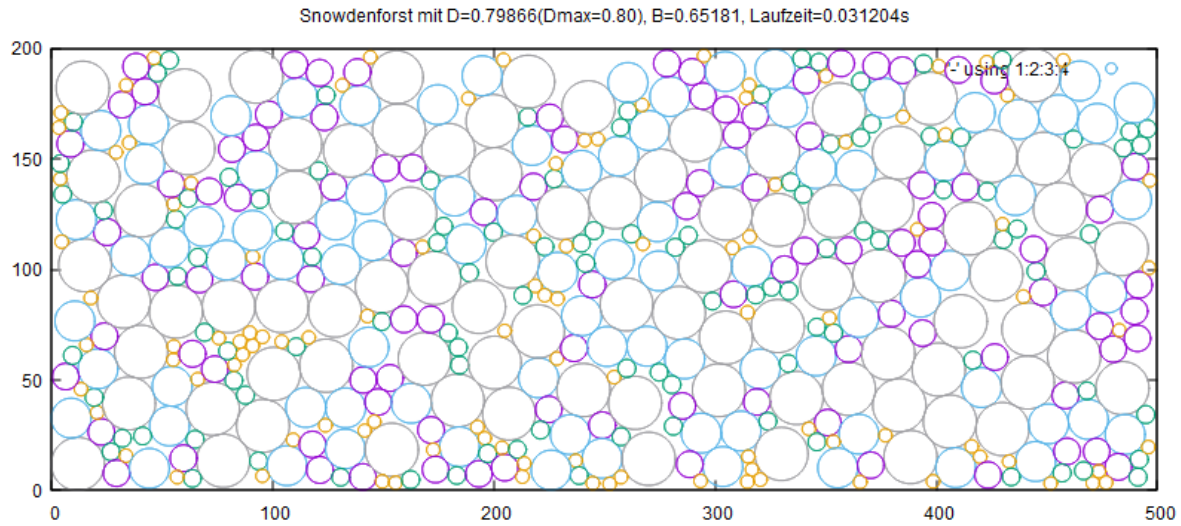

Abbildung 8 - IHK Beispiel 2

Auswertung:

Ein gutes Ergebnis, da die Diversität ihrem Optimum nahe kommt und eine gute Abdeckung erreicht wurde. Ein typischer günstiger Fall für den Algorithmus.

4.2.3 Extremfall 1

Eingabe:

```
Extremfall 1
10 10 % Länge und Breite in Metern
5,1    % Esche
```

Ein unlösbarer Fall, da der minimale Radius keinen gültigen Kreis ergeben kann. Programm sollte leeres Waldstück ausgeben.

Ausgabe:

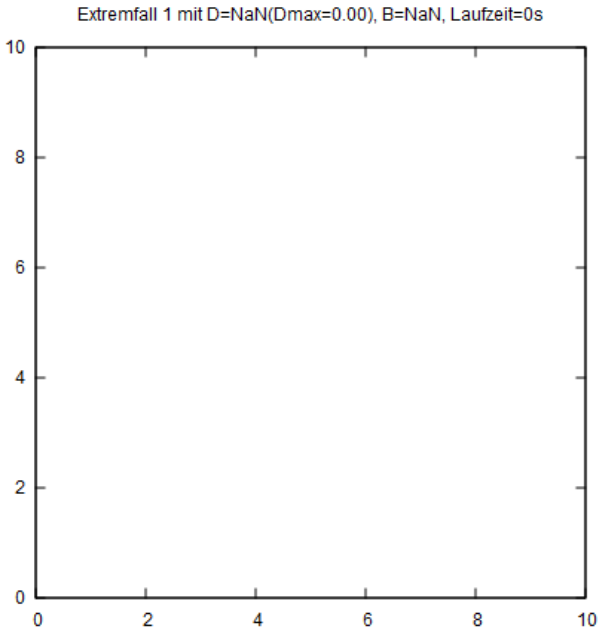

Abbildung 9 - Extremfall 1

Auswertung:

Programm läuft durch und liefert erwartetes Ergebnis.

4.2.4 Extremfall 2

Eingabe:

Extremfall 2	
100 10	**% Länge und Breite in Metern**
5	**% Esche**

Ausgabe:

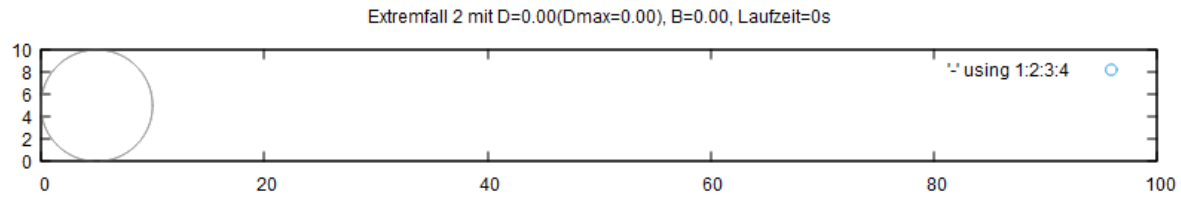

Abbildung 10 - Extremfall 2

Auswertung:

Der Algorithmus terminiert, da keine Paare nach dem Dreiecksansatz konstruiert werden können. Allerdings liegt hier kein praxisrelevanter Fall vor.

4.2.5 Extremfall 3

Eingabe:

Extremfall 3	
10 100	**% Länge und Breite in Metern**
5	**% Esche**

Ausgabe:

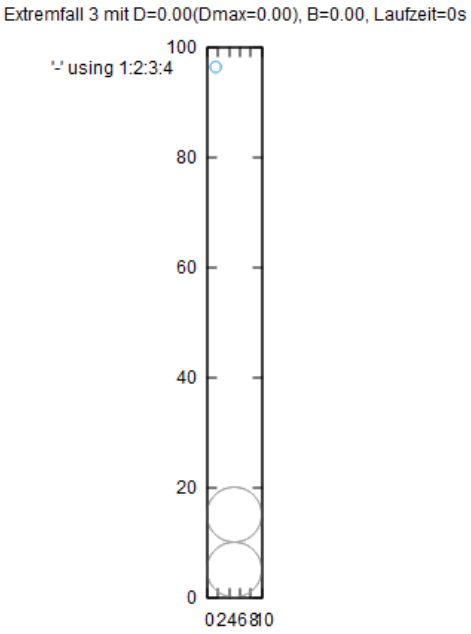

Abbildung 11 - Extremfall 3

Auswertung:

Der Algorithmus terminiert, da keine Paare nach dem Dreiecksansatz konstruiert werden können. Allerdings liegt hier kein praxisrelevanter Fall vor. Ähnliches Problem wie bei Extremfall 2.

4.2.6 Extremfall 4

Eingabe:

Extremfall 4	
20 100	**% Länge und Breite in Metern**
5	**% Esche**

Ausgabe:

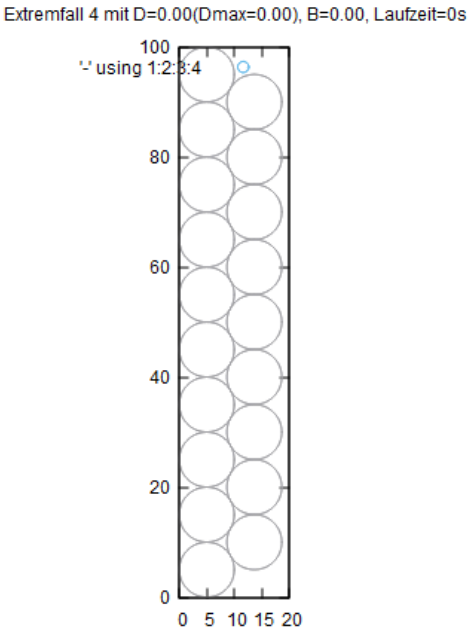

Abbildung 12 - Extremfall 4

Auswertung:

Hier ist der Algorithmus wirksam, da valide Positionen aus Dreiecken konstruiert werden können.

4.2.7 Extremfall 5

Eingabe:

Extremfall 5	
100 20	**% Länge und Breite in Metern**
5	**% Esche**

Ausgabe:

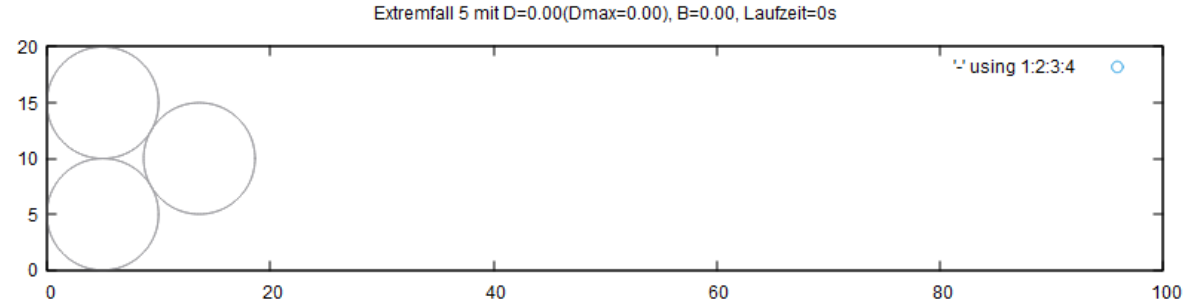

Abbildung 13 - Extremfall 5

Auswertung:

Der Algorithmus terminiert, da die Waldfläche sehr flach ist. Der Fall ist kaum praxisrelevant, außerdem kann man durch vertauschen von Länge und Breite den Extremfall 4 erhalten und dieser ist gut lösbar.

4.2.8 Extremfall 6

Eingabe:

Extremfall 6	
100 30	**% Länge und Breite in Metern**
5	**% Esche**

Ausgabe:

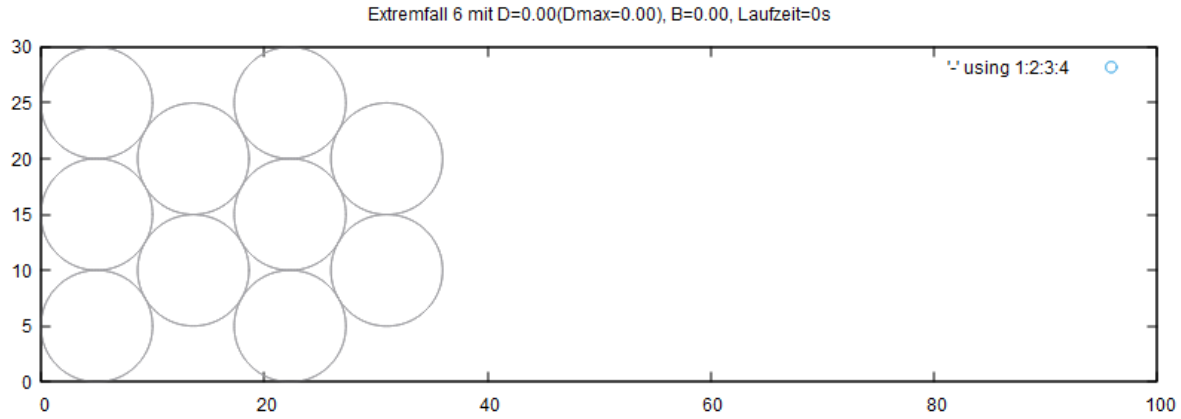

Abbildung 14 - Extremfall 6

Auswertung:

Ein ähnliches Problem wie in Extremfall 5. Der Algorithmus terminiert hier allerdings, obwohl theoretisch eine weitere Entwicklung möglich ist. Dies liegt daran das die letzten beiden entwickelten Bäume ein paar bilden, welches aber nicht registriert wird. Ein Vertauschen von Breite und Höhe liefert auch hier ein gutes Ergebnis.

4.2.9 Extremfall 7

Eingabe:

Extremfall 7	
30 100	**% Länge und Breite in Metern**
5	**% Esche**

Ausgabe:

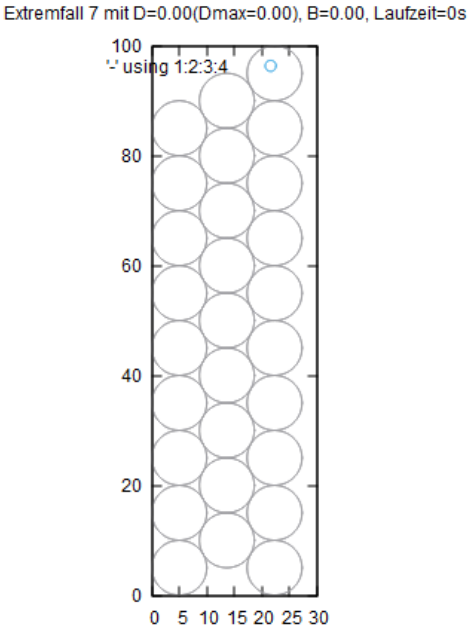

Abbildung 15 - Extremfall 7

Auswertung:

Beweis der Behauptung zu Extremfall 6.

4.2.10 Extremfall 8

Eingabe:

```
Extremfall 8
1500 1500
8
5
4
5
12
3
4
```

Ausgabe:

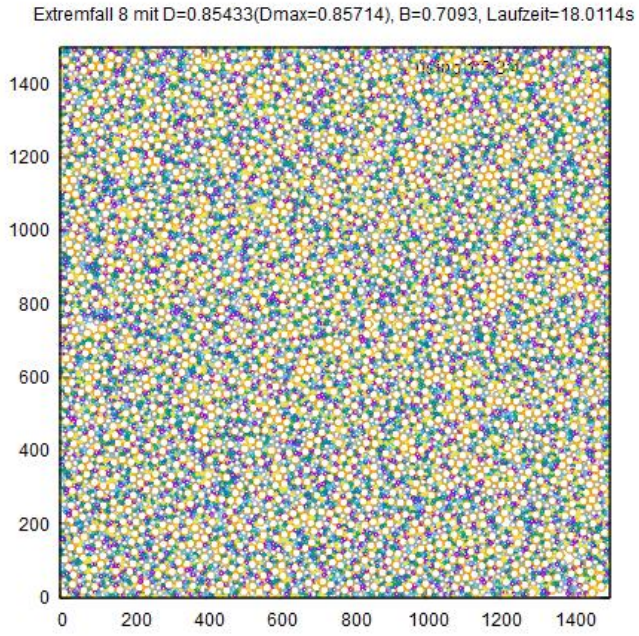

Abbildung 16 - Extremfall 8

Auswertung:

Eine sehr vielversprechende Lösung. Beide Indizes liegen in einem sehr guten Bereich, die Diversität konvergiert bei großen Testfeldern gegen ihr Optimum. Die Laufzeit ist signifikant länger als bei anderen Testfällen, aber immer noch in einem guten Rahmen.

4.2.11 Extremfall 9

Eingabe:

```
Extremfall 9
100 200
20
1
```

Ausgabe:

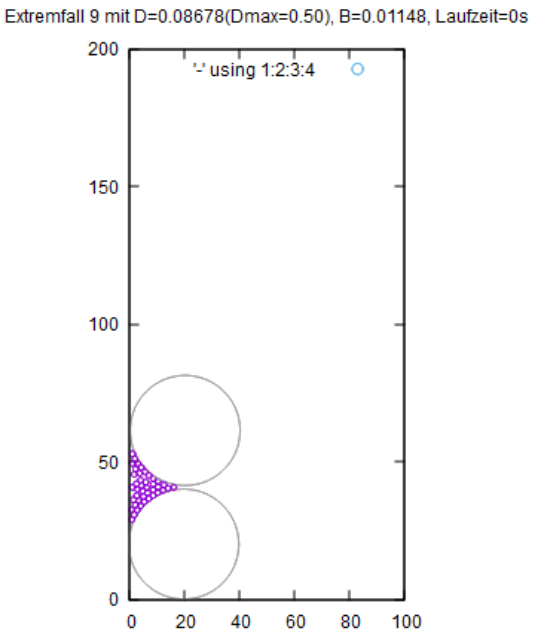

Abbildung 17 - Extremfall 9

Auswertung:

Nicht praxisrelevanter Fall. Durch einen großen Unterschied zwischen den Radien, führt die Bemühung, eine möglichst gleichmäßige Anzahl aller Baumarten zu erreichen, zu einer vorzeitigen Terminierung des Algorithmus.

4.2.12 Falsches Format Test

Eingabe:

```
FalschesFormatWald
30
1
2
3
4
```

Ausgabe:

Konsolenausgabe der Fehlerbeschreibung.

Auswertung:

Falsche Formatierungen werden abgefangen.

4.2.13 Keine Datei vorhanden Test

Eingabe:

Eingabe eines falschen Pfades in Konsole.

Ausgabe:

Konsolenausgabe der Fehlerbeschreibung.

Auswertung:

Invalider Pfad führt nicht zu Programmabsturz.

4.2.14 Normalfall 1

Eingabe:

```
NormalWald1
500 300
10
9
8
7
6
5
4
```

Ein großes Waldstück mit vielen Baumarten.

Ausgabe:

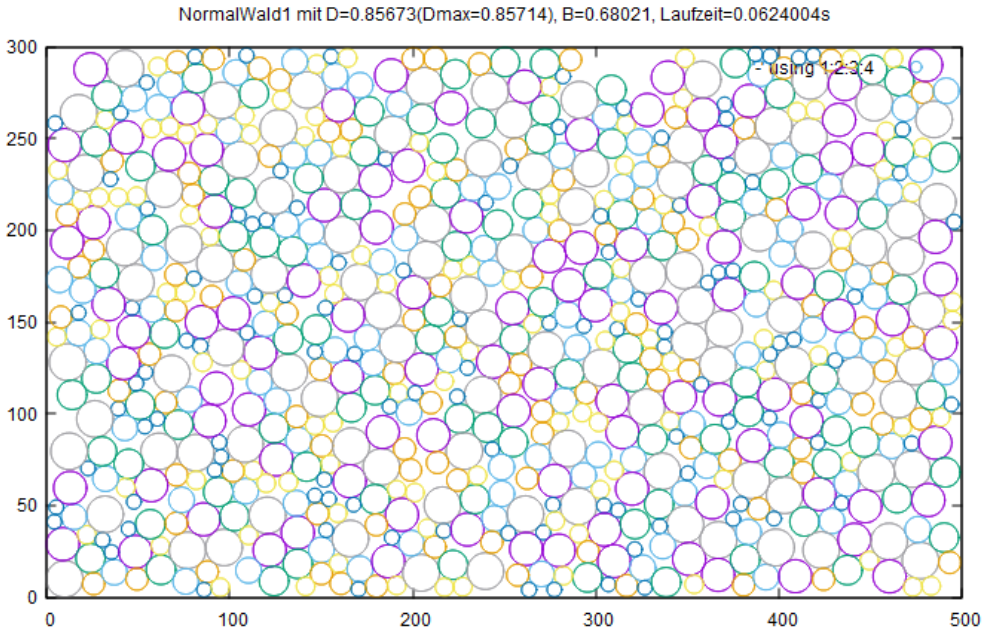

Abbildung 18 - Normalfall 1

Auswertung:

Ein gutes Ergebnis, da die Diversität ihrem Optimum nahe kommt und eine gute Abdeckung erreicht wurde. Ein typischer günstiger Fall für den Algorithmus.

4.2.15 Normalfall 2

Eingabe:

```
NormalWald2
500 300
6
5
4
```

Ein großes Waldstück mit wenigen Baumarten.

Ausgabe:

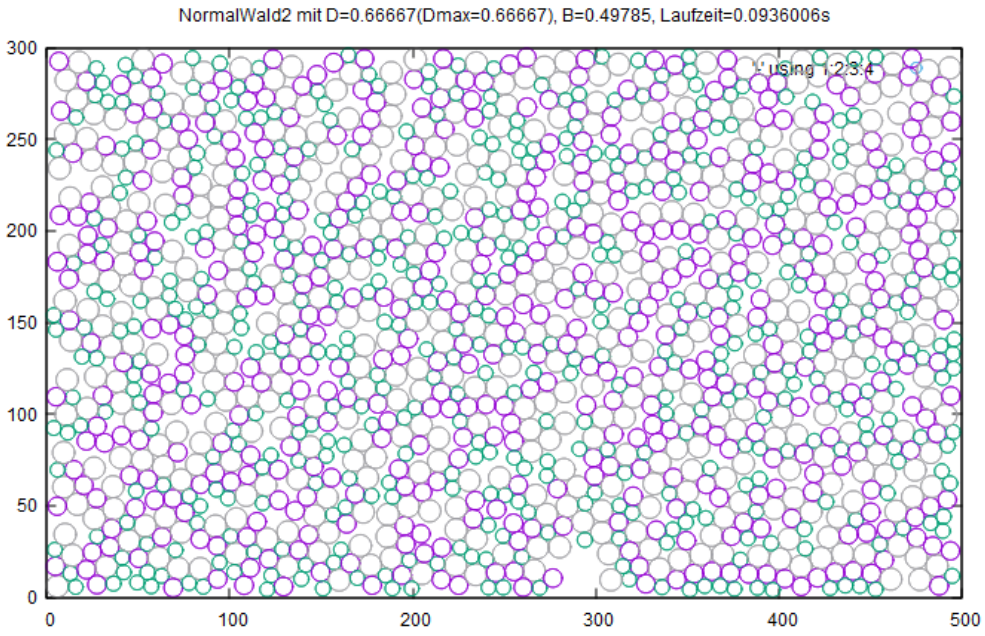

Abbildung 19 - Normalfall 2

Auswertung:

Ein gutes Ergebnis, da die Diversität ihrem Optimum nahe kommt und eine gute Abdeckung erreicht wurde. Ein typischer günstiger Fall für den Algorithmus. Die Anzahl der Arten spielt eine untergeordnete Rolle. Das Verhältnis der Radien zueinander und zur Größe des Waldstückes ist entscheidend.

4.2.16 Normalfall 3

Eingabe:

```
NormalWald3
50 30
8
5
4
```

Ein kleines Waldstück mit wenigen Baumarten.

Ausgabe:

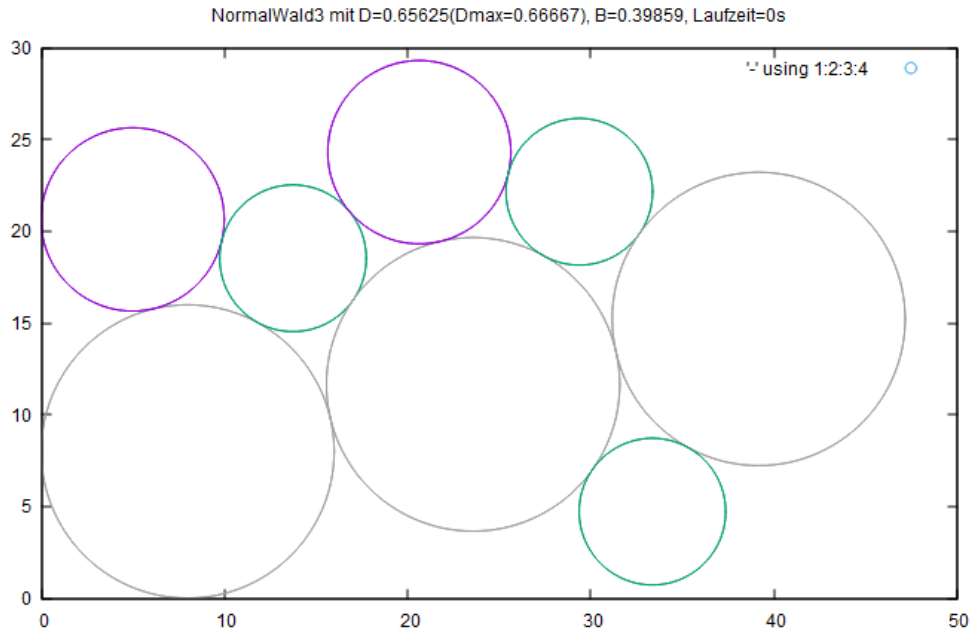

Abbildung 20 - Normalfall 3

Auswertung:

Ein gutes Ergebnis, da die Diversität ihrem Optimum nahe kommt und eine gute Abdeckung erreicht wurde. Ein typischer günstiger Fall für den Algorithmus.

4.2.16 Normalfall 4

Eingabe:

```
NormalWald4
50 30
8
5
4
5
12
3
4
```

Ein kleines Waldstück mit vielen Baumarten.

Ausgabe:

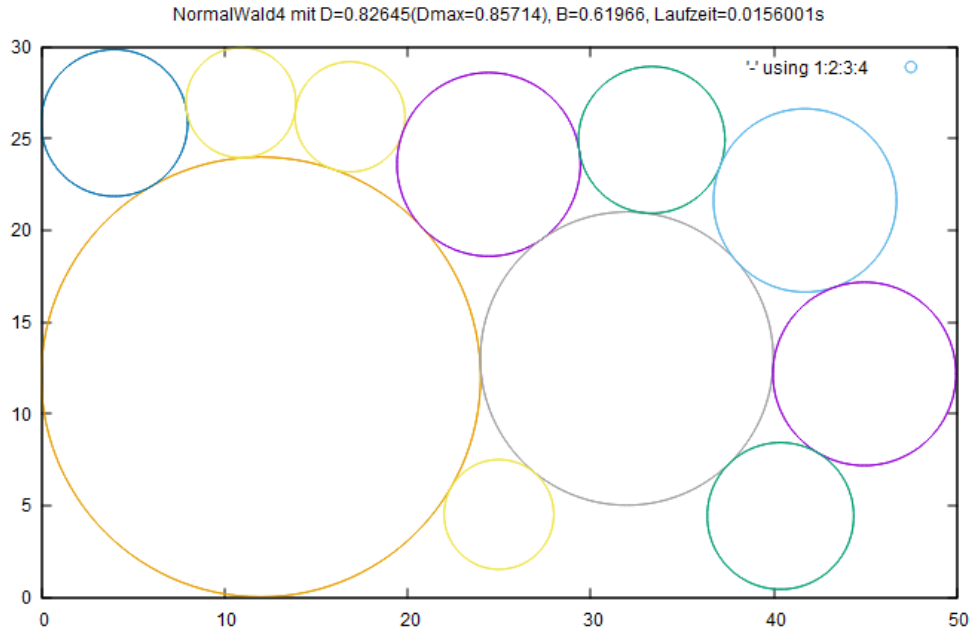

Abbildung 21 - Normalfall 4

Auswertung:

Ein gutes Ergebnis, da die Diversität ihrem Optimum nahe kommt und eine gute Abdeckung erreicht wurde. Ein typischer günstiger Fall für den Algorithmus.

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

Die Aufgabenstellung wurde für Normalfälle gelöst. Eine in etwa der Realität entsprechende Eingabedatei führt in der überwiegenden Mehrzahl der Fälle zu einem guten Ergebnis.
Probleme entstehen durch Extremfälle bezüglich Seitenverhältnis des Waldstückes oder Verhältnis der Radien der Baumarten zueinander. Diese Fälle müssen aber provoziert werden und treten im Anwendungsfall beinahe nie auf.
Bei der Bewertung der Indizes muss stets darauf geachtet werden, dass D zwar theoretisch im Intervall [0:1] angesiedelt ist, aber im konkreten Anwendungsfall eine obere Grenze von $D_{\text{Max}}=\frac{1}{m}$ hat, wobei m der Anzahl der verschiedenen Arten entspricht. Automatisch ergibt sich daraus eine obere Grenze $B_{\text{Max}}= D_{\text{Max.}}$ Berücksichtigt man dies, sind die Ergebnisse dieses Programmes, meiner Meinung nach, zufriedenstellend. Die Laufzeit wird erst bei sehr großen Simulationsparametern relevant groß und dieses Problem kann durch eine effektivere Speicherform behoben werden.

5.2 Mögliche Erweiterungen

Die einfachste Verbesserung des Programmes liegt in der besseren Datenverwaltung bezüglich der bereits ermittelten Bäume. Bei der Kollisionsabfrage wird in dieser Implementierung stets ein Vergleich des aktuellen Baumes gegen alle andern durchgeführt. Würde man hier ein geeignetes Raster wählen, und die Bäume darin einordnen müsste man immer nur gegen alle Bäume aus Nachbarfeldern testen. Dies würde die Auswirkung der Fläche des Waldstückes auf die Laufzeit beinahe aufheben. Diese Verbesserung war nicht in der Planung vorgesehen und wird daher nicht mit implementiert, da auch ohne sie eine gute Laufzeit erreicht wird.
Im Zusammenhang mit dieser Erweiterung könnte man auch einen Test auf nicht triviale neue Baumpaare einbauen. Hierzu könnte man die Menge aller Bäume in der eingegrenzten Nachbarschaft auf direkte Nachbarschaft zum gerade berechneten Baum überprüfen und dieses so entstehende Paar der zu untersuchenden Menge hinzufügen.

Eine weitere gute Erweiterung wäre das Hinzufügen von Algorithmischen Ansätzen für Sonderfälle die in den Testfällen beschrieben sind. (Sehr schmales Feld)
Da diese zumeist keinerlei praktische Relevanz haben, wurden sie hier vernachlässigt.

6 Abbildungsverzeichnis

Abbildung 1 - UML-Klassendiagramm .....- 12 -
Abbildung 2 - UML Use Case Diagramm.....- 13 -
Abbildung 3 - UML Sequenzdiagramm .....- 14 -
Abbildung 4 - Struktogramm Simulation.VerteileBaeume.....- 15 -
Abbildung 5 - Struktogramm IstValidePosition.....- 16 -
Abbildung 6 - Beispielausgabeplot .....- 18 -
Abbildung 7 - IHK Beispiel 1 .....- 20 -
Abbildung 8 - IHK Beispiel 2 .....- 21 -
Abbildung 9 - Extremfall 1 .....- 22 -
Abbildung 10 - Extremfall 2 .....- 23 -
Abbildung 11 - Extremfall 3 .....- 24 -
Abbildung 12 - Extremfall 4 .....- 25 -
Abbildung 13 - Extremfall 5 .....- 26 -
Abbildung 14 - Extremfall 6 .....- 27 -
Abbildung 15 - Extremfall 7 .....- 28 -
Abbildung 16 - Extremfall 8 .....- 29 -
Abbildung 17 - Extremfall 9 .....- 30 -
Abbildung 18 - Normalfall 1 .....- 32 -
Abbildung 19 - Normalfall 2 .....- 33 -
Abbildung 20 - Normalfall 3 .....- 34 -
Abbildung 21 - Normalfall 4 .....- 35 -

7 Anhang(Quellcode)

7.1 Klasse Program

```
using System;
using AufforstungMischwald.IO;

namespace AufforstungMischwald
{
    /// <summary>
    /// Startklasse.
    /// Leitet die Eingabeparameter in die entsprechenden Programmteile weiter.
    /// </summary>
    internal class Program
    {
        private const string Usagestring = "usage: RettetDenWald.exe <pathToInputF
ile>";
        private const string NoFileFoundString= "There was no file found under the
 given path. Maybe a typo?";
        private const string FileWrongFormatString= "The file has a wrong format.
See documentation for proper format informations.";
        private const int MaximaleWiederholungen = 10;

        private static void Main(string[] args)
        {
            //Überprüfung des Parameterarrays
            if (args.Length != 1)
            {
                Console.WriteLine(Usagestring);
                return;
            }

            string path = args[0];

            ValidationResult validationResult = FileValidator.Validate(path);
            if (validationResult == ValidationResult.Ok)
            {
                int durchgefuehrteSimulationen = 0;
                while (true)//Wiederholtes simulieren sichert ab das ein gutes Erg
ebnis gefunden wird, wenn es eines gibt.
                {
                        Simulation sim = FileReader.Read(path);

                        sim.VerteileBaeume();

                    durchgefuehrteSimulationen++;

                    if (sim.GetB()/sim.GetD() > 0.5||durchgefuehrteSimulationen==M
aximaleWiederholungen) //Ist die abgedeckte Fläche größer als 50% der Gesamtfläche
?
                    {
                        FileWriter.Write(sim,path);
                        break;
                    }
                }
            }
```

```
            else
            {
                FehlerAusgabe(validationResult);
            }
        }

        private static void FehlerAusgabe(ValidationResult validationResult)
        {
            if (validationResult==ValidationResult.NoFileFound)
            {
                    Console.WriteLine(NoFileFoundString);
            }
            else
            {
                    Console.WriteLine(FileWrongFormatString);
            }
        }
    }
}
```

7.2 Klasse FileReader

```
using System.Collections.Generic;
using System.IO;
using AufforstungMischwald.Model;

namespace AufforstungMischwald.IO
{
    /// <summary>
    /// Utlility-
Klasse mit deren Hilfe eine Eingabedatei des vorgegebenen Formats eingelesen werde
n kann.
    /// Aus dieser Datei wird eine Simulations-Instanz erzeugt.
    /// Diese Klasse ist nur unter vorherigem verwenden der FileValidator Klasse z
ulässig, da ein unzulässiges Format oder nicht Vorhandensein der Datei Programmabs
türze verursachen könnte.
    /// </summary>
    internal static class FileReader
    {
        public static Simulation Read(string path)
        {
            var arten = new List<Baumart>();

            var lines = new List<string>(File.ReadAllLines(path));

            lines = new List<string>(Utils.RemoveComments(lines));

            //Parse Waldname
            string name = lines[0];

            //Parse Breite und Höhe
            string[] lineOneSplit = lines[1].Split(' ');
            double breite = double.Parse(lineOneSplit[0]);
            double hoehe = double.Parse(lineOneSplit[1]);

            //Erzeuge Baumarten
            for (int i = 2; i < lines.Count; i++)
            {
                double radius=double.Parse(lines[i]);
                arten.Add(new Baumart(i - 2, radius));
            }

            return new Simulation(breite, hoehe, name, arten);
        }
    }
}
```

7.3 Klasse FileValidator

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;

namespace AufforstungMischwald.IO
{
    /// <summary>
    /// Klasse die das Format und das Vorhandensein einer Eingabedatei überprüft.
    /// Liegt ein Problem mit einem dieser Aspekte vor wird eine entsprechende Nac
hricht an die Benutzende Klasse zurückgegeben.
    /// </summary>
    internal static class FileValidator
    {
        public static ValidationResult Validate(string path)
        {
            if (!File.Exists(path))
            {
                return ValidationResult.NoFileFound;
            }

            var lines = new List<string>(File.ReadAllLines(path));

            lines = new List<string>(Utils.RemoveComments(lines));

            var result = ValidationResult.Ok;

            if (lines.Count() < 3)
            {
                result = ValidationResult.WrongFormat;
            }

            try
            {
                string[] lineOneSplit = lines[1].Split(' ');
                double dummy;
                if (lineOneSplit.Length != 2)
                {
                    result = ValidationResult.WrongFormat;
                }
                foreach (string s in lineOneSplit)
                {
                    if (!double.TryParse(s, out dummy))
                    {
                        result = ValidationResult.WrongFormat;
                    }
                    if (dummy <= 0)
                    {
                        result = ValidationResult.WrongFormat;
                    }
                }

                for (int i = 2; i < lines.Count; i++)
                {
```

```
                    if (!double.TryParse(lines[i], out dummy))
                    {
                        result = ValidationResult.WrongFormat;
                    }
                    if (dummy <= 0)
                    {
                        result = ValidationResult.WrongFormat;
                    }
                }
            }
            catch (Exception)
            {
                result = ValidationResult.WrongFormat;
            }

            return result;
        }
    }
}
```

7.4 Klasse FileWriter

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Threading;

namespace AufforstungMischwald.IO
{
 /// <summary>
 /// Klasse zum umwandeln einer erfolgten Simulation in das geforderte Ausgabeform
at.
 /// </summary>
 internal static class FileWriter
 {
  private const string FormatStringBaum = "{0:0.0######} {1:0.0######} {2:0.0#####
#} {3:0.0######}";

  public static void Write(Simulation sim, string path)
  {
   try
   {
    Thread.CurrentThread.CurrentCulture = CultureInfo.InvariantCulture;
    var lines = new List<string>
                {
                 "reset",
                 string.Format("set yrange[0:{0:0.0######}]", sim.Hoehe),
                 string.Format("set xrange[0:{0:0.0######}]", sim.Breite),
                 string.Format("set size ratio {0:0.0######}", (sim.Hoehe/sim.Brei
te)),
                 string.Format(
                                "set title \"{0} mit D={1:0.00###}(Dmax={2:0.00###}
), B={3:0.00###}, Laufzeit={4:0.#######}s\"",
                                sim.WaldName,
                                sim.GetD(),
                                sim.GetBestD(),
                                sim.GetB(),
                                sim.Laufzeit.TotalSeconds),
                 "plot '-' using 1:2:3:4 with circles lc var"
                };

    lines.AddRange(
                   sim.ErgebnisBaeume.Select(
                                             baum =>
                                             string.Format(FormatStringBaum,
                                                           baum.Position.X,
                                                           baum.Position.Y,
                                                           baum.Art.Radius,
                                                           baum.Art.Index)));

    File.WriteAllLines(string.Format("{0}.plt", path), lines);
   }
   catch (IOException)
   {
```

```
    Console.WriteLine(
                        "Es ist ein fehler während des Schreibens der Ausgabedatei a
ufgetreten. Überprüfen sie mögliches Fehlen von Schreibrechten.");
   }
   catch (Exception)
   {
    Console.WriteLine("Ein unerwarteter Fehler beim schreiben der Ausgabedatei ist
 aufgetreten.");
   }
  }
 }
}
```

7.5 Klasse Utils

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace AufforstungMischwald.IO
{
 /// <summary>
 ///Utility-
Klasse die aus einer Zeile der Eingabedatei alle Kommentare entfernen kann.
 /// </summary>
 internal static class Utils
 {
  public static IEnumerable<string> RemoveComments(IEnumerable<string> lines)
  {
   var result = new List<string>();
   foreach (string s in lines)
   {
    result.Add(RemoveCommentFromLine(s));
   }
   return result.Select(s => s.Trim());
  }

  private static string RemoveCommentFromLine(string line)
  {
   if (line.Contains("%"))
   {
    int indexOfComment = line.IndexOf("%", StringComparison.Ordinal);
    return line.Remove(indexOfComment);
   }
   return line;
  }
 }
}
```

7.6 Enum ValidationResult

```
namespace AufforstungMischwald.IO
{
    /// <summary>
    /// Rückgabewerte der FileValidation Klasse
    /// </summary>
    internal enum ValidationResult
    {
        Ok,
        WrongFormat,
        NoFileFound
    }
}
```

7.7 Klasse Baum

```
namespace AufforstungMischwald.Model
{
    /// <summary>
    /// Klasse die einen konkreten Baum repräsentiert.
    /// </summary>
    internal class Baum
    {
        public Baumart Art { get; set; }
        public Position Position { get; set; }

        public override string ToString()
        {
            return string.Format("{0} | {1}", Position, Art);
        }
    }
}
```

7.8 Klasse Baumart

```
namespace AufforstungMischwald.Model
{
    /// <summary>
    /// Klasse deren Instanzen verschiedene Baumarten repräsentieren.
    /// </summary>
    internal class Baumart
    {
        private readonly int _index;
        private readonly double _radius;

        /// <summary>
        /// Konstruktor.
        /// </summary>
        /// <param name="index">Der Index der Baumart in der Eingabedatei.</param>
        /// <param name="radius">Der Radius der Bepflanzungsfläche der Baumart.
</param>
        public Baumart(int index, double radius)
        {
            _index = index;
            _radius = radius;
        }

        /// <summary>
        /// Bisherige Menge an Bäumen dieser Art.
        /// </summary>
        public int Anzahl { get; set; }

        public int Index
        {
            get { return _index; }
        }

        public double Radius
        {
            get { return _radius; }
        }

        public override string ToString()
        {
            return string.Format("r={0}, n={1}, i={2}", _radius, Anzahl, _index);
        }
    }
}
```

7.9 Klasse Position

```
namespace AufforstungMischwald.Model
{
    /// <summary>
    /// Container der eine Position im zweidimensionalen Raum abbildet.
    /// </summary>
    internal struct Position
    {
        public double X { get; set; }
        public double Y { get; set; }

        public override string ToString()
        {
           return string.Format("X={0} Y={1}", X, Y);
        }
    }
}
```

7.10 Klasse Simulation

```
using System;
using System.Collections.Generic;
using System.Linq;
using AufforstungMischwald.Model;

namespace AufforstungMischwald
{
 /// <summary>
 /// Instanzen dieser Klasse berechnen eine möglichst günstige Bepflanzungs-
Ordnung für ein Waldstück gegebener Größe und eine Menge von verwendbaren Baumarte
n.
 /// Wichtige Kriterien sind die Diversität der Baumarten und ein möglichst hoher
Nutzungsgrad der Gesamtfläche.
 /// </summary>
 internal class Simulation
 {
  private const double Toleranz = 1e-5;
  private readonly double _breite;
  private readonly double _hoehe;
  private readonly string _waldName;
  private readonly List<Baumart> _arten;
  private readonly List<Baum> _ergebnisBaeume;
  private readonly List<Tuple<Baum, Baum>> _zuTestendeBaumpare;
  private readonly Random _random = new Random();
  private TimeSpan _laufzeit;

  /// <summary>
  /// Konstruktor für zentrale Simulations-Engine
  /// </summary>
  /// <param name="breite">Die Breite der Waldfläche.</param>
  /// <param name="hoehe">Die Höhe der Waldfläche.</param>
  /// <param name="path">Der Pfad unter dem die Eingabedatei liegt.</param>
  /// <param name="waldName">Der Name des Waldstückes.</param>
  /// <param name="arten">Die Menge aller pflanzbaren Baumarten.</param>
  public Simulation(double breite, double hoehe, string waldName, IEnumerable<Baum
art> arten)
  {
   _breite = breite;
   _hoehe = hoehe;
   _waldName = waldName;
   _arten = new List<Baumart>(arten);
   _ergebnisBaeume = new List<Baum>();
   _zuTestendeBaumpare = new List<Tuple<Baum, Baum>>();
  }

  public IEnumerable<Baum> ErgebnisBaeume
  {
   get { return _ergebnisBaeume; }
  }

  public double Breite
  {
   get { return _breite; }
  }
```

```
  public double Hoehe
  {
   get { return _hoehe; }
  }

  public string WaldName
  {
   get { return _waldName; }
  }

  public TimeSpan Laufzeit
  {
   get { return _laufzeit; }
  }

  /// <summary>
  /// Gibt Simpson-Index zurück. Dieser gibt die Diversität der Baumarten an.
  /// </summary>
  public double GetD()
  {
   double result = 1.0;
   foreach (Baumart baumart in _arten)
   {
    result -= Math.Pow(baumart.Anzahl/(double) _ergebnisBaeume.Count, 2);
   }
   return result;
  }

  /// <summary>
  /// Gibt Erweiterten Simpson-
Index zurück. Hier liegt das Gewicht stärker auf der abgedeckten Fläche.
  /// </summary>
  public double GetB()
  {
   double d = GetD();
   double summeZaehler = 0;
   foreach (Baumart baumart in _arten)
   {
    summeZaehler += baumart.Anzahl*Math.PI*Math.Pow(baumart.Radius, 2);
   }
   double gesamtFlaeche = Breite*Hoehe;
   double b = d*summeZaehler/gesamtFlaeche;
   return b;
  }

  /// <summary>
  /// Startet die Simulation mit den im Konstruktor definierten Parametern.
  /// Zentrales Element ist die Menge aller Baumpaare, aus welchen sich eventuell
eine weitere Position eines Baumes errechnen lässt.
  /// Diese Methode kann pro Instanz dieser Klasse nur einmal gerufen werden.
  /// </summary>
  public void VerteileBaeume()
  {
   PositioniereStartpaar();

   //Zeitmessung
```

```
   DateTime start = DateTime.Now;

   //Solange eine Möglichkeit auf eine weiter Pflanzposition besteht, rechnet der
Algorithmus.
   while (_zuTestendeBaumpare.Count > 0)
   {
    SortiereBaumarten();

    //Ein Auswürfeln ob man mit den "alten" oder den "neuen" Expansionspunkten for
tfährt ist sinnvoll, da die wahrscheinlichkeit für ein terminieren des Algorithmus
, vor einer Abdeckung des gesamten Feldes stark sinkt.
    int index = _random.Next(0, 2) - 1;
    if (index == -1)
    {
     index = _zuTestendeBaumpare.Count - 1;
    }
    Tuple<Baum, Baum> testBaumpaar = _zuTestendeBaumpare[index];

    //Da die Baumarten sortiert sind wird immer ein möglichst seltener Baum mit mö
glichst großem Radius gewählt, da dies am besten den geforderten Kriterien entspri
cht.
    foreach (Baumart baumart in _arten)
    {
     //Mögliche Positionen werden bestimmt und validiert.
     List<Position> validePositionen = BerechneValidePositionen(testBaumpaar, baum
art);
     if (validePositionen.Count < 1)
     {
      continue;
     }

     //Der Aufbau des Algorithmus hat zur Folge das stetts nur eine der möglichen
Positionen valide ist, da die andere Entweder außerhalb des Waldes liegt oder inne
rhalb einer bereits bepflanzten Fläche.
     var neuerBaum = new Baum
                     {
                      Art = baumart,
                      Position = validePositionen[0]
                     };

     //Die Menge der zu untersuchenden Bäume wird erweitert.
     _zuTestendeBaumpare.Add(new Tuple<Baum, Baum>(testBaumpaar.Item1, neuerBaum))
;
     _zuTestendeBaumpare.Add(new Tuple<Baum, Baum>(testBaumpaar.Item2, neuerBaum))
;
     _ergebnisBaeume.Add(neuerBaum);
     baumart.Anzahl++;
     break;
    }

    //Das Baumpaar wurde untersucht und da jedes Baumpaar nur eine valide neue Pos
ition liefern kann, wird es aus der Untersuchungsmenge entfernt.
    _zuTestendeBaumpare.Remove(testBaumpaar);
   }

   //Zeitmessung
   _laufzeit = DateTime.Now - start;
```

```
  }

  /// <summary>
  /// Gibt bestmöglichen Wert des Simpson-Index zurück. Zu Vergleichszwecken.
  /// </summary>
  public double GetBestD()
  {
   return 1.0 - 1/(double) _arten.Count;
  }

  /// <summary>
  /// Berechnet valide Positionen für einen neuen Baum in Nachbarschaft zu einem B
aumpaar.
  /// </summary>
  /// <param name="untersuchtesTuple">Das Baumpaar.</param>
  /// <param name="baumart">Die Art des möglichen neuen Baumes.</param>
  /// <returns>Die möglichen Positionen.</returns>
  private List<Position> BerechneValidePositionen(Tuple<Baum, Baum> untersuchtesTu
ple, Baumart baumart)
  {
   Baum baum1 = untersuchtesTuple.Item1; //M1
   Baum baum2 = untersuchtesTuple.Item2; //M2

   Position m1 = baum1.Position;
   Position m2 = baum2.Position;

   double r1 = baum1.Art.Radius;
   double r2 = baum2.Art.Radius;
   double r3 = baumart.Radius;

   //Bestimme die Länge aller Seiten des entstehenden Dreiecks
   double l1 = r2 + r3;
   double l2 = r1 + r3;
   double l3 = r1 + r2;

   //Bestimme Winkel an M1
   double phi = Math.Acos((l2*l2 + l3*l3 - l1*l1)/(2*l2*l3));

   //Bestimme Punkt S welcher das rechtwinklige Dreieck M1-S-M3 bildet
   double betragSM3 = Math.Sin(phi)*(l2);
   double betragSM1 = Math.Cos(phi)*(l2);
   double sX = m1.X + (m2.X - m1.X)*(betragSM1/l3);
   double sY = m1.Y + (m2.Y - m1.Y)*(betragSM1/l3);
   var s = new Position
           {
            X = sX,
            Y = sY
           };

   //Bestimme genormten Richtungsvektor der von S auf M3 bzw. M3' zeigt
   double richtungsVektorX = (m2.Y - m1.Y);
   double richtungsVektorY = -(m2.X - m1.X);
   double laengeRichtungsVektor = Math.Sqrt(richtungsVektorX*richtungsVektorX + ri
chtungsVektorY*richtungsVektorY);
   double normRichtungsVektorX = richtungsVektorX/laengeRichtungsVektor;
   double normRichtungsVektorY = richtungsVektorY/laengeRichtungsVektor;
```

```
 //Bestimme mögliche Lösungen
 var moeglichePosition1 = new Position
                          {
                           X = s.X + normRichtungsVektorX*betragSM3,
                           Y = s.Y + normRichtungsVektorY*betragSM3
                          };
 var moeglichePosition2 = new Position
                          {
                           X = s.X - normRichtungsVektorX*betragSM3,
                           Y = s.Y - normRichtungsVektorY*betragSM3
                          };

 //Überprüfe mögliche Lösungen
 return new List<Position>
        {
         moeglichePosition1,
         moeglichePosition2
        }.Where(position => IstValidePosition(position, r3))
         .ToList();
}

/// <summary>
/// Sortiert die Baumarten.
/// </summary>
private void SortiereBaumarten()
{
 _arten.Sort(BaumComparison);
}

private void PositioniereStartpaar()
{
 SortiereBaumarten();
 PositioniereBaumEins();

 SortiereBaumarten();
 PositioniereBaumZwei();
}

/// <summary>
/// Positioniert den ersten Baum in der linken unteren Ecke. (wenn möglich)
/// Hierfür wird ein möglichst großer Baum genutzt.
/// </summary>
private void PositioniereBaumEins()
{
 foreach (Baumart baumart in _arten)
 {
  double radius = baumart.Radius;
  var pos = new Position
            {
             X = radius,
             Y = radius
            };
  if (!IstValidePosition(pos, radius))
  {
   continue;
  }
```

```
    _ergebnisBaeume.Add(new Baum
                        {
                         Art = baumart,
                         Position = pos
                        });
    baumart.Anzahl++;
    break;
   }
  }

  /// <summary>
  /// Positioniert den zweiten Baum zwischen dem Rand des Waldes und Baum 1(so vor
handen).
  /// Hierfür wird ein möglichst großer Baum genutzt.
  /// </summary>
  private void PositioniereBaumZwei()
  {
   if (_ergebnisBaeume.Count != 1)
   {
    return;
   }

   Baum ersterBaum = ErgebnisBaeume.First();

   foreach (Baumart baumart in _arten)
   {
    double radius = baumart.Radius;
    double radiusBaumEins = ersterBaum.Art.Radius;
    double y = radiusBaumEins + Math.Sqrt(Math.Pow(radiusBaumEins + radius, 2) -
 Math.Pow(radiusBaumEins - radius, 2));

    var pos = new Position
              {
               X = radius,
               Y = y
              };
    if (!IstValidePosition(pos, radius))
    {
     continue;
    }

    var zweiterBaum = new Baum
                      {
                       Art = baumart,
                       Position = pos
                      };
    _ergebnisBaeume.Add(zweiterBaum);
    baumart.Anzahl++;
    _zuTestendeBaumpare.Add(new Tuple<Baum, Baum>(ersterBaum, zweiterBaum));
    break;
   }
  }

  /// <summary>
  /// Sortiert zuerst nach der Anzahl der vorhandenen Ausprägungen einer Baumart.(
aufsteigend)
  /// Danach wird nach der größe des Radius sortiert.(absteigend)
```

```
  /// </summary>
  private static int BaumComparison(Baumart a, Baumart b)
  {
   int diffAnzahl = a.Anzahl - b.Anzahl;
   if (diffAnzahl == 0)
   {
    try
    {
     return Convert.ToInt32(b.Radius - a.Radius);
    }
    catch (OverflowException)
    {
     //Leer da einfach auf Differenz der Anzahl zurückgefallen werden kann.
    }
   }
   return diffAnzahl;
  }

  /// <summary>
  /// Überprüft ob eine Position ein valider Platz für einen Baum mit einem bestim
mten Radius liefert.
  /// </summary>
  private bool IstValidePosition(Position pos, double radius)
  {
   if (!IstInnerhalbDesWaldes(pos, radius))
   {
    return false;
   }

   if (!UeberschneidetSichMitAnderenBaeumen(pos, radius))
   {
    return false;
   }
   return true;
  }

  private bool UeberschneidetSichMitAnderenBaeumen(Position pos, double radius)
  {
   foreach (Baum baum in _ergebnisBaeume)
   {
    double abstandsQuadrat = (pos.X - baum.Position.X)*(pos.X - baum.Position.X) +
                             (pos.Y - baum.Position.Y)*(pos.Y - baum.Position.Y);
    double diff = abstandsQuadrat -
 (radius + baum.Art.Radius)*(radius + baum.Art.Radius);

    //Der Versuch diesen Codeblock zu optimieren ,indem man das Ziehen der Wurzel
durch ein Quadrieren der Gesamtgleichung auflöst und die Framework Funktion Math.P
ow durch eine triviale Quadrierung
    //ersetzt, erbrachte ca. 100% Leistungssteigerung

    //double abstand = Math.Sqrt(Math.Pow(pos.X -
 baum.Position.X, 2) + Math.Pow(pos.Y - baum.Position.Y, 2));
    //double diff = abstand - (radius + baum.Art.Radius);

    if (diff < -Toleranz)
    {
     return false;
```

```
    }
   }
   return true;
  }

  private bool IstInnerhalbDesWaldes(Position pos, double radius)
  {
   double minX = pos.X - radius;
   double minY = pos.Y - radius;
   double maxX = pos.X + radius;
   double maxY = pos.Y + radius;

   if (minX < -Toleranz || minY < -Toleranz || Breite - maxX < -
Toleranz || Hoehe - maxY < -Toleranz)
   {
    return false;
   }
   return true;
  }
 }
}
```

8 Eigenhändigkeitserklärung:

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

_____ _____

Datum, Ort Unterschrift Prüfungsteilnehmer