**Dorian Reineccius**

# Snake

## Eine grafische Visualisierung mit Java

**Informatik LK ‘14/‘15**

**Fachlehrer: Herr Faßbender**

# Inhaltsverzeichnis

# 1. Vorwort

Als ich noch sehr jung war, ungefähr 11 Jahre als, hab ich sehr gern und auch viel Computerspiele an meinem erstem Computer gespielt, alles andere wofür dieser Computer imstande war, war mit egal. Doch brachte mich mein Vater irgendwann auf eine Idee: Warum spielen, wenn ich diese auch selbst entwickeln kann? Warum sinnlos Zeit verschwenden, wenn ich mit so etwas später Geld verdienen könnte? Sofort war ich von dieser Idee überzeugt, doch hatte ich keinen Anlaufplan, keine Idee wie anzufangen war.

Mein Vater war es, der programmieren konnte, er also war mein erster Anlaufpunkt. Er hatte aber wenig Zeit um mich darin zu lehren, so gab er mir sein aktuellstes C++ Handbuch und besorgte mit ein Lehrbuch für Kinder, wie man einfach programmieren in C++ lerne. Nun steckte ich auch sofort viel Zeit darein und opferte den Großteil meiner Zeit in dieses Projekt, C++ zu erlernen. Ich las mir Seite um Seite dieses Buches durch und schaute mir auch die aktuellsten englischsprachigen Tutorials auf YouTube an. Es war alles sehr leicht zu verstehen und total logisch, man konnte alles total schnell nachvollziehen und selbst ausprobieren. Aber zu schnell erkannte ich, dass ich so leicht niemals an ein Spiel in den 3 Dimensionalen Räumen kommen könne. Soviel Wissen ich nun auch hatte, so verlor ich langsam auch die Lust daran, weiter zu machen. Meine Motivation, ein Hochqualitative großes Videogame zu konstruieren schien hoffnungslos und somit gab ich trotz der vielen Zeitinvestitionen auf.

Dann merkte ich, dass ich nun eine ganz andere Denkweise besaß.

Ungefähr 5 Jahre vergingen, als ich mich wieder mit meinem ehemaligen besten Freund zusammenfand, mit welchem ich mich nach der Grundschule auseinanderlebte. Er war hobbymäßig Webdesigner und arbeitete um diese Zeit fast ausschließlich mit HTML5 und CSS. Er weckte mein Interesse und gab mir auch einen Crashkurs zu diesem Thema, anscheinend wollte er, dass ich mit ihm zusammen arbeite.

Als Webdesigner kommt man ohne Programmierung nicht weit.

Da wir ausschließlich mit Auszeichnungssprachen arbeiteten merkten wir häufig, dass uns etwas fehlte. So startete ich ein Jahr später ein Selbststudium in MySQL und PHP. Ohne mir aber großartig neues Wissen anzueignen, konnte ich mir das meiste schon erschließen, ohne es zu erlernen. Die logische Denkweise reichte vollkommen aus.

Bis heute arbeite ich sehr gerne an Webprojekten, mein bester Freund hat mittlerweile seine eigene UG, in welcher er mich unbedingt als Informatiker einstellen will, doch strebe ich vorher noch um ein Studium in diesem Bereich.

So wie der Zufall es will, kam ich vor Monaten erst wieder auf die Spieleprogrammierung zurück, ich schaffte mir ein Buch an, in welchem einem die Spieleprogrammierung mithilfe von DirectX und C++ beigebracht wird. Und so kam ich auch auf die Idee als Informatikfacharbeit Unterrichtsspezifisch in Java, ein Spiel zu implementieren.

# 1.1. Vorgehensweise

Natürlich bin ich so vorgegangen wie ich es am besten kann und am besten kann ich es wie ich es gelernt habe.

Die Modellierung mit UML habe ich erst sehr spät kennengelernt, und zwar Im ersten Quartal der 1. Qualifikationsphase des Abiturs. Dort nahmen wir die Unified Modelling Language als Teil der „Grundlagen der OOP“ durch. UML Modellierung hilft einem deutlich in der Objekt Orientierten Programmierung. Beim Implementieren hat man also einen klaren Plan vor Augen und muss praktisch nur noch „abschreiben“.

Schreibt man aber mit BlueJ, so hat man während der Programmierung schon einen groben Plan vor Augen und ist nicht so stark auf UML angewiesen. Ich jedoch habe das Programmieren immer ohne UML Diagramm praktiziert und bin ohne dieses ein wenig schneller, deswegen habe ich dieses Projekt zuerst implementiert und im Nachhinein modelliert. Die Modellierung dient schließlich auch im Nachhinein als schnelle Übersicht und erleichtert die Zurechtfindung in dem Projekt für Andere deutsch.

Logischerweise hat man vorher immer auch einen klaren Plan „vor Augen“, ich bin relativ schnell darauf gekommen, wie ich das Projekt aufzubauen habe. Ich habe das Programm aufgebaut, wie ein Architekt:

> Der Architekt zeichnet einen groben Plan seines Gebäudes, findet er es gut, so legt er ein durchsichtiges Papier auf diesen und zeichnet ein wenig detaillierter. Akzeptiert er auch diesen Entwurf, so nimmt er das nächste Papier und „spezialisiert“ so immer wieder sein Projekt.
>
> Der Programmierer entwickelt eine grobe Klasse, seines Programmes, findet er sie gut, so lässt er eine neue Klasse von der vorherigen erben und spezialisiert diese. Akzeptiert er auch diesen Quellcode, so erstellt er die nächste Klasse und spezialisiert so immer wieder sein Projekt.

Womöglich gibt es trotzdem viele verschiedene andere Methoden, sein Programm zu konstruieren, ich finde meine Methode jedoch am einfachsten und am übersichtlichsten. Was man nicht ausprobiert, das kann man auch nicht beurteilen.

> Der Architekt zeichnet bei dem fertigen und eingerichteten Gebäude, zum Schluss den Gebäudeplan, damit fremde sich darin schneller zurechtfinden.
>
> Ich habe erst bei dem fertigem Programm die Modellierung gemacht, nebenbei auch, damit sich andere darin schneller zurecht finden.

Ich nenne diese Vorgehensweise, der **Architekten-Bauplan**.

# 2. Vorwort

Snake was sich aus dem Englischen zu Schlange übersetzen lässt ist ein Videospielklassiker in welchem der Spieler eine Schlange steuert und das auf dem Feld liegende Futter aufsammeln soll. Anfangs ist die Schlange sehr klein und nimmt mit jedem Aufsammeln von Futter an Länge zu. Die Schwierigkeit des Spiels besteht darin bei der immer länger werdenden Schlange nicht an den Schwanz zu treffen. Dies und auch das antreffen an die Feldwände führen zum Verlieren des Spieles.

In manchen Versionen von Snake werden auch zufällig wände auf dem Feld generiert, was den Schwierigkeitsfaktor wiederrum um einiges erhöht. Zudem gibt es auch Versionen von sich die Schnelligkeit der Schlange mit jedem Aufnehmen von Futter erhöht. Selten gibt es außerdem Versionen, in welchem man mit der Schlange durch ein Labyrinth finden muss

Wer sich schon einmal den Film Tron angeschaut hat merkt, dass auch diese Spielart eine Anlehnung an Snake ist. So macht sich bemerkbar, dass es viele Variationsmöglichkeiten von Snake gibt.

Beliebt wurde Snake auf alten Computer, auf welchen erst sehr wenige Spiele liefen und die Grafik noch lange nicht so ausgereift war wie heute. Vor der Zeit der Smartphones wurde Snake auch auf sehr vielen Tastenhandys installiert und diente als netter Zeitvertreib für die Mehrheit der Nutzer.

Heute aber wird Snake nur noch sehr wenig gespielt, aus dem einfachen Grund, dass die Grafik schon viel weiter ist als früherem und so Spiele mit viel aufwendigere Grafik ermöglicht.

Snake bietet sich auf Grund seiner Einfachheit gut dafür an, dieses nach zu programmieren, da dieses keine komplexe Spiel Engine benötigt, keine Töne und kein hochauflösenden 3 Dimensionalen Raum. Aus diesem Grund ist die Programmierung von Snake eine effiziente Übung für Programmier-Anfänger, Schüler oder Studenten, welche sich zudem in viele verschiedene Variationen entfalten lässt.

## 2.1 Gründe für die Wahl von Snake

Es gibt unendlich viele Möglichkeiten, was man thematisch im Fach Informatik behandeln kann, jedoch war es für mich nicht so leicht ein Thema zu finden, welches an meinen behandelten Unterrichtsthemen anlehnt.

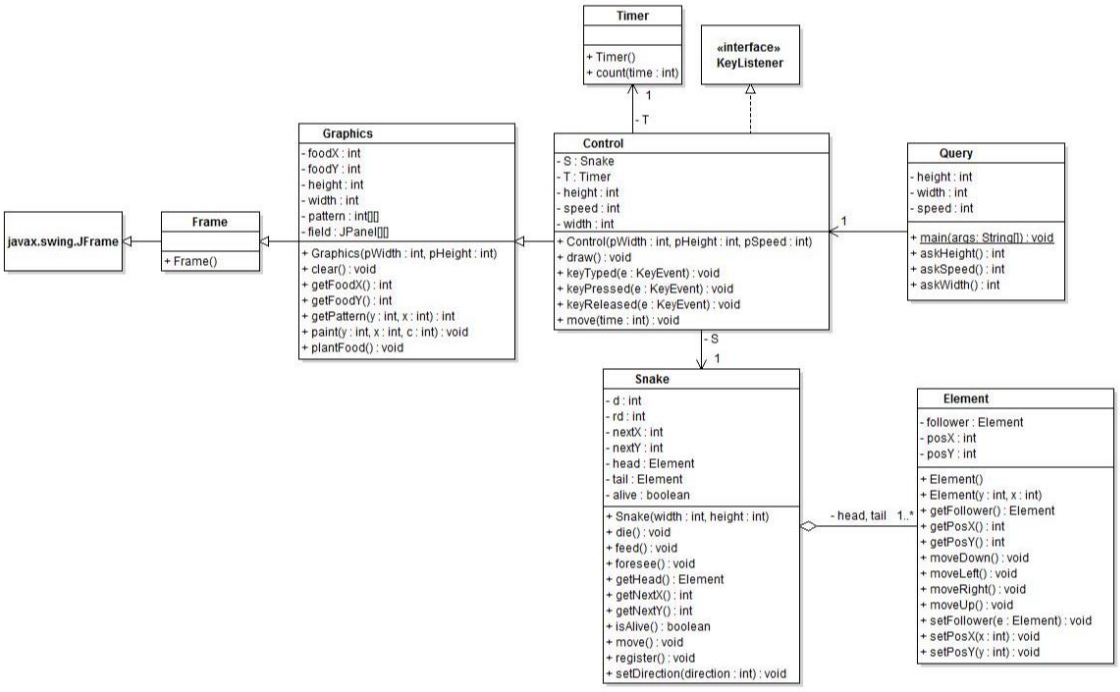
Der Vorschlag von Snake kam letztendlich von meinem Fachlehrer. Snake lehnte wohl am besten an die Unterrichtseinheiten die in meinem Kurs behandelt wurden. Im ersten Halbjahr hatten wir neben der GUI (Graphics User Interface) auch die linearen ADT's (Abstrakter Datentyp) wie Schlangen, Stapel und Listen implementiert, so brauchte ich im Nachhinein eine Combo aus GUI und Schlange zu programmieren und ein wenig mit Zufallsgeneratoren zu arbeiten.

Eine Programmierte Version von Snake auf Java bietet vor allem auch eine gute Grundlage, wenn man ähnliche oder sogar ganz andere 2 Dimensionale Videospiele schreiben möchte, zum Beispiel wollte ich schon immer mal ein ähnliches Spiel entwickeln, welches Diamond Caves ähnlich ist, was wiederrum eine Version vom Klassiker Boulder Dash ist.

Arbeitet man später vielleicht etwas fortschrittlicher im Bereich der Grafik, so lassen sich auch so noch sehr gute Spiele aus der 2D Perspektive programmieren. Ein gutes Beispiel ist das Spiel Terraria von Re-Logic, dieses Spiel läuft ebenfalls auf Java und ist momentan eines der meist gespielten Games auf der Spieleplattform Steam.

Das wohl berühmteste Spiel auf Java ist vermutlich Minecraft von Mojang AB und auch dieses ist ziemlich primitiv geschrieben. Was ich damit sagen will ist, dass man auch mit wenig Aufwand kreative und vielleicht sogar erfolgreiche Videospiele entwickeln kann. Doch fängt jeder klein an und sollte sich auch den Grundlagen bewusst sein, was man besten lernt, indem man zum Beispiel kleine Videogames implementiert oder andere kleine Programme schreibt. Java bietet dazu geschrieben auf BlueJ eine sehr Anfängerfreundliche und Übersichtliche Entwickler Umgebung, welche man natürlich als fortgeschrittener Programmierer auf NetBeans oder Eclipse ausweiten kann. In meinem Fall benutze ich aber BlueJ, aufgrund des übersichtlichen Quellcodes.

# 3. Die Modellierung




(vgl. Anhang 1)

Bei der Modellierung, stellte ich mir das Spiel vor, als bestand es aus einzelnen Bauteilen. Ich bräuchte also als erstes ein Fenster, darauf ein gitterförmiges Spielfeld und darauf die Schlange.

So kam es zustande, dass ich die Klasse Frame implementierte, welche die Klasse javax.swing.JFrame spezialisierte. Diese Klasse Graphics wiederrum spezialisiert die Klasse Frame mit einem Spielfeld und einem Muster, in welchem der Status eines jeden Feldes steckt. Die wichtigste Klasse ist die Klasse Control, sie spezialisiert die Klasse Graphics mit einer Spielsteuerung und instanziiert außerdem selbst die Schlange. Zudem besitzt sie einen Timer, welcher bis zu jedem neuen Spielzug zählt.

Die Schlange selbst ist ähnlich wie die Queue der linearen ADT's implementiert. Sie besteht aus einem bis unendlich vielen Elementen, welche aneinander hängen. Zu guter Letzt baute ich mir die Klasse Query, welche die Main-Methode des Programms beinhaltet. Diese Klasse fragt vom Benutzer als erster Die Spielfeld Dimensionen ab und die gewünschte Spielgeschwindigkeit, anschließend instanziiert sie die Klasse Control, mit den passenden Werten.

In meinen Augen eine sehr übersichtliche Art der Modellierung, durch die häufige Spezialisierungen erleichtert man sich die Implementationsschritte und bekommt Ordnung in das Projekt, alles wird dort hineingeschrieben, wo es hingehört.

# 4. Die Implementation

Die Implementation scheint wohl als der langwierigste Prozess und somit als Hauptteil der Facharbeit.

Die Klassen- und Variablennamen sind einheitlich alle auf Englisch geschrieben, der einfachheitshalber und um Umlaute zu vermeiden, welche sicher zu Fehlern führen würden.

Angefangen habe ich mit der Klasse Frame, welche direkt alles der von Java bereitgestellten Klasse javax.swing.JFrame erbt. Frame habe ich mit einem Titel, Fenstergröße, Startposition, Größenverstellbarkeit und der Operation für das Schließen des Fensters versehen. Dabei habe ich mich fast komplett an einem Buch orientiert, was setTitle(), setSize() und setDefaultCloseOperation(EXIT_ON_CLOSE) betrifft (vgl. Quelle 7). Für die Zentrale Startposition habe ich setLocationRelativeTo(null) benutzt (vgl. Quelle 1). Die Einstellung, dass man die Größe des JFrame nicht verändern kann, benutzte ich mit setResizable(false) aus (vlg. Quelle 8). Die Klasse Frame importiert alle Klassen und Methoden in javax.swing.* damit es von der Klasse JFrame erben kann.

In der Klasse Graphics befindet sich die Technologie für das Spielfeld von Snake. Die Integer height und width speichern die Dimensionen des Feldes und die Integer foodX und foodY die Koordinaten des Futters für die Schlange. Der Konstruktor konstruiert das Spielfeld, indem er width und height definiert und per for-Schleife das Spielfeld sowie das Muster definiert, welche beide aus zwei-dimensionalen Arrays bestehen. Die Methode void clear() „säubert“ das Spielfeld, färbt also alle Felder Weiß und setzt des Status jeden Feldes auf 0, was leer bedeutet. Die Methoden int getFooxX() und int getFoodY() geben jeweils foodX und foodY zurück, dementsprechend gibt int getPattern(int y, int x) den Status an der Stelle (y/x) des Musters zurück. Die Klasse void paint(int y, int x, int c) setzt den Status des Feldes (y/x) auf c und färbt es anschließend passend ein. Dabei bedeutet der Status 0, dass das Feld leer ist und das entsprechende Feld wird weiß eingefärbt. Der Status 1 bedeutet, dass sich in diesem Feld ein Element der Snake befindet und wird somit schwarz eingefärbt. Der Status 2 bedeutet, dass sich in diesem Feld Futter für die Snake befindet und wird grün eingefärbt. Die Methode void plantFoot() setzt auf ein zufälliges Feld das Futter, ist der Status des Musters des entsprechenden Feldes nicht 0, so wird ein nächster Versuch getätigt. Dabei mache ich Gebrauch der von Java bereitgestellten Methode Math.random() (vgl. Quelle 4). Die Klasse Graphics importiert java.awt.* für das GridLayout() (vgl. Quelle 6) und javax.swing.* für die Klassen JButton, JFrame und JLabel.

Die Klasse Control importiert java.awt.event.KeyEvent, welches nötig ist, um eine Aktion zu definieren, welche durch einen Tastendruck ausgelöst wird, java.awt.event.KeyListener, um die gedrückten Tasten zu erkennen und um den KeyListener in die Klasse Control zu implementieren und javax.swing.JOptionPane, um Gebrauch der Klasse JOptionPane machen zu können (vgl. Quelle 3). Zunächst erbt die Klasse Control alle Methoden und Attribute der Klasse Graphics und besitzt drei weitere Attribute, zum einen Snake S, einen Timer T und drei Integer height, width und speed. Der Konstruktor instanziiert Graphics, erstellt eine neue

Schlange und bringt den Spielverlauf in Bewegung. Verliert man in dem Spiel, so kann es jederzeit neugestartet werden mithilfe des Konstruktors. Die Methode void draw() zeichnet das Feld neu, säubert es also, setzt alle Elemente der Schlange ein und zudem das Futter. Die Methoden keyTyped(KeyEvent e), keyPressed(KeyEvent e) und keyReleased(KeyEvent e) müssen zwangsläufig überschrieben werden, da das Interface KeyListener implementiert wird (vgl. Quelle 2). Wichtig für uns ist die Klasse keyPressed(KeyEvent e), sie bestimmt nämlich, was passiert, wenn eine der vier Pfeiltasten gedrückt wird. Die Methode void move(int time) bewegt nicht nur die Schlange, sondern insziniert einen kompletten Spielzug. Die Methode setzt das erste Futter und aktualisiert das Spielfeld. Solange die Schlange noch lebt, wird der Timer heruntergezählt, der Nächste Zug der Snake registriert und geprüft, die Schlange bewegt und zuletzt das Feld neu gezeichnet. Die Methode boolean revive() bestimmt wenn das Spiel vorbei ist, ob eine neue Runde beginnen soll, oder das Spiel beendet werden soll.

Die Klasse Snake beinhaltet vier Integer. Zum einen d für die aktuelle Richtung und rd für die registrierte Richtung, welche nach Ablauf des Timers auf jeden Fall betreten wird, anschließend nextX und nextY, welche die Koordinaten des nächsten Feldes bestimmen, welches die Schlange mit sicherheit betreten wird. Zudem besitzt die Klasse zwei Zeiger, welche auf das erste und das letzte Element der Schlange zeigen, also den Kopf und den Schwanz. Zudem hat die Klasse noch einen Wahrheitswert alive, welcher bestimmt, ob die Schlange tot oder lebendig ist. Der Konstruktor der Klasse Snake setzt die Schlange mittig in das Spielfeld. Die Methode void die() „tötet“ die Schlange. Das Spiel wird somit automatisch beendet. Die Methode void feed() „füttert“ die Schlange, weswegen sie um ein Element länger wird. Die Methode foresee (deutsch: Vorraussehen) bestimmt die Koordinaten des Feldes welches mit Sicherheit als nächstes von der Schlange betreten wird. Die Methoden Element getHead(), int getNextX() und int getNextY() geben jeweils head, nextX und nextY zurück. Die Methode boolean isAlive() gibt den Wahrheitswert alive zurück. Die Methode void move() bewegt die Schlange mitsamt all ihren Elementen um ein Feld in die registrierte Richtung weiter. Die Methode void register() registriert die Richtung und ruft die methode foresee() auf und die Methode void setDirection(int direction) setzt die aktuelle Richtung.

Die Klasse Element besitzt einen Zeiger follower, welcher auf das nachfolgende Element zeigt und zwei Integer posX und posY, welche die Position des Elements bestimmt. Der Konstruktor Element() setzt den Nachfolger auf null und lässt die Koordinaten des Elements unbestimmt. Der Konstruktor Element(int y, int x) dagegen setzt ebenso den Nachfolger auf null, setzt aber dafür die Koordinaten des Elements. Die Methoden Element getFollower(), int getPosX() und int getPosY() geben jeweils den Nachfolger, X-Koordinate oder die Y-Koordinate zurück. Die Methoden void moveDown(), void moveLeft(), void moveRight() und void moveUp() bewegen das Element in die entsprechende Richtung. Die Methoden void setFollower(Element e), int setPosX(int x) und int setPosY(int y) setzen jeweils Nachfolger, X-Koordinate oder Y-Koordinate fest.

Die Klasse Timer besitzt einen Konstruktor und eine Methode void count(int time). Bei dieser Methode und allen Methoden, welche diese instanziieren muss zwangsläuft throws InterruptedException hinter stehen (vgl. Quelle 5). Diese Methode zählt per Thread.sleep() eine bestimmte Zeit in Millisekunden herunter und „stoppt“ somit das Programm, während dieser Zeit kann der Spieler seine Wunschrichtung der Snake wählen.

Die Klasse Query importiert javax.swing.JOptionPane für die Verwendung der Klasse JOptionPane. Die Klasse besitzt drei Integer, welche erst nach Verwendung der JOptionPane’s definiert werden und mit diesen Werten die Klasse Control instanziiert wird. Der Konstruktor der Klasse Query instanziiert die Klasse Control, nachdem Sie die Werte für height, width und speed vom Benutzer übergeben bekommen hat. Die Main-Methode static void main(String[] args) wird bei dem Programmstart aufgerufen und instanziiert die Klasse Query. Die Methode

int askHeight(),int askSpeed() und int askWidth() fragen vom Benutzer jeweilst die gewünschte Höhe oder Breite des Spielfeldes ab und die gewünschte Spielgeschwindigkeit.

Die Klassen, wie sie hier aufgelistet sind, wurden auch so in dieser Reihenfolge implementiert, die darin enthaltenen Methoden und Attribute haben leider keine mir bekannte Reihenfolge mehr. Für die bessere Übersicht würde ich aber empfehlen die Funktionen der Methoden im Quelltext der Klassen zu entnehmen, so ist es deutlich einfacher, sich in das Projekt hineinzuversetzen(vgl. Anhang 9).

# 5. Literaturverzeichnis


1. Andrew Swan, http://stackoverflow.com/questions/144892/how-to-center-a-window-in-java, 28. September 2008, 0:49 (vgl. Anhang 2)
2. „COD3BOY“ (Nickname), http://stackoverflow.com/questions/10876491/how-to-use-keylistener, 04. Juni 2012, 5:28 (vgl. Anhang 3)
3. http://de.wikibooks.org/wiki/Java_Standard:_Grafische_Oberfl%C3%A4chen_mit_Swing:_Top_Level_Container:_javax_swing_JOptionPane, 26. Januar 2014, 12:20 (vgl. Anhang 4)
4. https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Math/random (vgl. Anhang 5)
5. https://docs.oracle.com/javase/tutorial/essential/concurrency/sleep.html (vgl. Anhang 6)
6. https://docs.oracle.com/javase/tutorial/uiswing/layout/grid.html (vgl. Anhang 7)
7. Siebler, Florian, Einführung in Java mit BlueJ. Objektorientierte Programmierung für Einsteiger, Galileo Press, Bonn 2011, 1. Auflage 2011
8. „The_S“ (Nickname), http://www.developers-guide.net/forums/1946,jframe-mittig-und-unveraenderbar-in-der-groesse, 30. Juni 2005, 14:47 (vgl. Anhang 8)

# 6. Anhang

# 7. Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als angegeben verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Ort:_____

Datum:_____Unterschrift:_____