Bericht zum Fortgeschrittenenpraktikum

„Objektorientierung und bedingte Anweisungen mit *POV-Ray*“

Ruprecht-Karls-Universität Heidelberg
Fakultät für Mathematik und Informatik
Institut für Informatik

Betreuung: Prof. Dr. Barbara Paech

Durchgeführt
im Wintersemester 2009/2010
von Lisa Tuschner

Inhaltsverzeichnis

1. Einleitung

Diese Arbeit setzt auf ein Softwarepraktikum einer Studentin (Frau Maria Catalina Filler) aus dem Wintersemester 2005/2006 auf. In diesem wurden Lehreinheiten zur Einführung in die objektorientierte Programmierung mit Hilfe von *POV-Ray* entwickelt, in denen die Schüler Einblicke die Grundkonzepte von Objekten, Attributen, Klassen und ansatzweise auch von Methoden erhielten (einzusehen bei Prof. Dr. Barbara Paech).

1.1 Vorbereitung

Für das Praktikum wurde zunächst in einer Literaturrecherche zusammengetragen, welches die für den Informatikunterricht wichtigen Programmierkonzepte sind und wie sie in den Lehrbüchern vermittelt werden. Daraufhin wurden einige Konzepte ausgewählt und Lehreinheiten erstellt, die die zuvor ausgewählten Programmierkonzepte mithilfe von *POV-Ray* vermitteln sollen.
Die hier vorgestellten Unterrichtseinheiten wurden mit Schülern der Unterstufe durchgeführt und die Ergebnisse der Durchführung in Anschluss an die einzelnen Lehreinheiten vermerkt.

1.2 Informatikunterricht heute

Informatik ist in der Unter- und Mittelstufe, für die dieser Kurs ausgelegt ist, nur in den Bundesländern Bayern, Mecklenburg-Vorpommern und Sachsen schon fest in den Stundenplan integriert. Deshalb gibt es in Baden-Württemberg leider noch keine Informatikbücher und Unterrichtskonzepte für jüngere Schüler.
Ein Ziel dieses Praktikums ist es zu zeigen, dass aber auch schon in diesen Klassenstufen ein Informatikunterricht durchaus sinnvoll und lehrreich sein kann und dass sich auch schon viel jüngere Schüler von Informatik begeistern lassen, wenn sie ihnen auf eine spielerische Weise vermittelt wird.

1.3 Lernziele und Bildungsplanbezug

6. Klasse:

In Bayern sollen Schüler der 6. Klasse im Fach „Natur und Technik“ folgendes Grundwissen in Schwerpunkt Informatik erhalten:
(Übernommen vom Staatsinstitut für Schulqualität und Bildungsforschung München:
http://www.isb-gym8-lehrplan.de/contentserv/3.1.neu/g8.de/index.php?StoryID=26172&PHPSESSID=0a78533124199f1ebe7e37f61288dcb2)

Information und ihre Darstellung

Anhand von Beispielen aus ihrer Erfahrungswelt soll den Schülern deutlich werden, dass Information auf ganz unterschiedliche Weise (z. B. Text, Bild, Diagramm, Ton) dargestellt werden kann. Sie sollen feststellen, dass es stark von der gewählten Darstellungsform abhängt, wie gut bzw. genau sich die ursprüngliche Information durch Interpretation wiedergewinnen lässt. Diese grundsätzliche Erkenntnis wird im Lauf des Schuljahres an verschiedenartigen Dokumenten – unterstützt durch die Gestaltungsmöglichkeiten der jeweiligen Software – weiter vertieft.

Informationsdarstellung mit Graphikdokumenten – Graphiksoftware (Schwerpunkt 1 dieses Praktikums)

Den Schülern soll bewusst werden, dass man mittels Graphiken einfach und effektiv Information darstellen kann. Spielerisch und intuitiv soll es ihnen gelingen, reale Situationen zielgerichtet in Graphiken abzubilden. Überlegungen zur Struktur von Graphiken führen zur objektorientierten Sichtweise. Die Schüler erkennen, dass jedes Objekt der Graphik bestimmte Eigenschaften hat und einer Klasse gleichartiger Objekte zugeordnet ist. Bei der praktischen Arbeit mit Graphikprogrammen wird ihnen auch die Notwendigkeit einer einfachen, einheitlichen Beschreibungssprache zur eindeutigen und effektiven Verständigung deutlich.

- Objekte als Informationseinheiten in Graphiken
- Objekte einer Vektorgraphik: Attribut, Attributwert und Methode
- Beschreibung gleichartiger Objekte durch Klassen: Rechteck, Ellipse, Textfeld, Linie

Informationsdarstellung mit Textdokumenten - Textverarbeitungssoftware

Beim Vergleich unterschiedlich gestalteter Texte stellen die Schüler fest, dass sich durch eine geeignete Strukturierung die enthaltene Information leichter gewinnen lässt. Sie analysieren Textstrukturen und entdecken, dass sich die bei Graphiken kennen gelernte objektorientierte Beschreibungsmethode als übertragbar und hilfreich erweist. Bei der praktischen Arbeit mit Textverarbeitungssoftware wird das Verständnis für diese Begriffe vertieft; dabei zeigt sich, dass einzelne Objekte miteinander in Beziehung stehen können. Die Schüler erkennen, dass viele alltägliche Zusammenhänge ebenfalls durch Beziehungen zwischen Objekten beschrieben werden können, diese Begriffe also eine allgemeinere Bedeutung haben.

- Objekte und Klassen in Texten: Zeichen, Absatz, Textdokument
- Verbesserung der Informationsdarstellung durch geeignetes Ändern von Attributwerten
- die Enthält-Beziehung zwischen Objekten; Entwerfen von Objekt- und Klassendiagrammen

Informationsdarstellung mit einfachen Multimediadokumenten - Präsentationssoftware

Durch Kombination verschiedener, schon bekannter Darstellungsarten von Information erstellen die Schüler Multimediadokumente und erkennen deren Nutzen. Dabei bewähren sich erneut die bereits erlernten Begriffe der objektorientierten Betrachtungsweise. Verschiedenartige Animationen, wie sie Präsentationssoftware zur Gestaltung bietet, helfen den Schülern, das Prinzip der Methoden von Objekten besser zu verstehen. In einem gemeinsamen Projekt mit dem Schwerpunkt Biologie sollen sie eine Präsentation zu einem Themenbereich erstellen. Dabei lernen sie auch Kriterien für die Qualität von Präsentationen kennen.

- die Klasse „Folie“ (Zusammenspiel von Text, Graphik, Bild, Ton)
- Verbesserung der Informationsdarstellung durch geeignete Animation der Objekte

Hierarchische Informationsstrukturen – Dateisystem

Beispiele aus ihrem Erfahrungsbereich bzw. der Biologie machen den Schülern deutlich, dass sich häufig hierarchisches Strukturieren zum Ordnen von Information eignet. Die Ordner und Dateien in einem Dateisystem bieten ihnen eine Möglichkeit zur Darstellung solcher Strukturen. Sie erkennen, dass hierarchische Ordnungen durch die Enthält-Beziehung zwischen Objekten der gleichen Klasse ermöglicht werden.

- hierarchische Informationsstrukturen am Beispiel von Ordnerbäumen: die Klassen „Datei“ und „Ordner“
- erweiterte Anwendung der Enthält-Beziehung: Ordner enthalten Ordner
- Konstruieren von Bäumen zur Strukturierung von Information (Verwandtschaft der Wirbeltiere)

7. Klasse:

In der 7. Klasse sind die Informatik Schwerpunkte folgende:

Vernetzte Informationsstrukturen - Internet

Die Schüler erfahren, dass inhaltliche Zusammenhänge zwischen Dokumenten zu vernetzten Strukturen führen können, für die eine hierarchische Darstellung nicht ausreicht. An Beispielen aus dem Internet sammeln sie Erfahrungen mit dem Hypertext-Konzept, das sich besonders gut zur Darstellung solcher Strukturen eignet und die Zusammenhänge einfach verfolgen lässt. In einem gemeinsamen Projekt mit dem Schwerpunkt Physik vertiefen sie die neu gewonnenen Kenntnisse, indem sie selbst Hypertexte beispielsweise zum Thema „optische Geräte“ erstellen. Ihr Wissen über die Informationsstruktur des Internets macht ihnen die Notwendigkeit geeigneter Suchstrategien deutlich, um die erforderlichen Informationen zu beschaffen. In diesem Rahmen werden auch rechtliche Aspekte des Interneteinsatzes angesprochen.

- das Vernetzungsprinzip von Hypertexten, insbesondere im Internet
- die Klassen „Verweis“ und „Verweisziel“, Adressen als Attributwerte von Verweisen
- die Beziehung „verweist auf“ zwischen Objekten
- Analysieren und Erstellen von Hypertextstrukturen; Informationsbeschaffung im Internet

Austausch von Information – E-Mail

Bei der Verwendung elektronischer Postsysteme erkennen die Schüler die vielfältigen Möglichkeiten zur bequemen, schnellen, weltweiten Kommunikation.

Mithilfe des entsprechenden Objektmodells sollen sie die wichtigsten Abläufe verstehen.

- die Klassen „Nachricht“ und „Anhang“, Adressen als Attributwerte von Nachrichten
- Verfassen, Versenden und Empfangen elektronischer Nachrichten
- Transportmechanismen: Zustellen und Abholen; Analogie zur Briefpost; Sicherheit

Beschreibung von Abläufen durch Algorithmen (Schwerpunkt 2 dieses Praktikums)

Die Schüler lernen eines der wichtigsten Grundprinzipien der automatischen Informationsverarbeitung kennen und erhalten einen ersten Einblick in seine Anwendung. Sie lernen, dass sich ganz allgemein mit Algorithmen Abläufe präzise und verständlich beschreiben lassen, und üben an konkreten Sachverhalten, insbesondere naturwissenschaftlichen Experimenten, Vorgänge aus einfachen Bausteinen aufzubauen. Dabei arbeiten sie mit einem Programmiersystem, mit dem sie die Algorithmen intuitiv umsetzen können und bei dem die Einzelschritte des Ablaufs altersgemäß visualisiert werden.

- Formulieren von Verarbeitungsvorschriften und Versuchsabläufen in Alltagssprache
- Bausteine von Algorithmen: Anweisung, Sequenz, Bedingte Anweisung, Wiederholung
- Programmieren eines einfachen Informatiksystems unter Verwendung dieser Bausteine

1.4 Lerninhalte (Bayern) in Schulbüchern (Übersicht)

Die unten beschriebenen Lernziele für die gymnasiale Unterstufe in Informatik wurden anhand von Schulbüchern aus Bayern erarbeitet.

Es wurden vor allem folgende Lehrbücher verwendet:
(genauere Beschreibung: siehe Literaturverzeichnis)

(1) Natur und Technik
(2) Netzwerk Informatik 6/7
(3) Enter 1
(4) Ikarus, Natur und Technik

Thematik	Lerninhalte/-ziele	Lehrbücher
Informationen und ihre Darstellung	• Software - Hardware • Grafische Benutzeroberflächen • Darstellungsformen	(1) (1) (3) (1)(2) (4)
Informationsdarstellung mit Grafikdokumenten	• Pixel – Vektorgrafiken • Objekte, Attribute, Methoden • Klassen	(1) (4) (1)(2) (4) (2)(4)
Informationsdarstellung mit Textdokumenten	• Schreiben am Computer • Textdokumente einrichten • Objekte in einem Textdokument • Bilder einfügen • Absätze, Wörter, Zeichen • Formatieren (Methoden) • Objekte und Klassen im Alltag	(1)(2)(3) (1) (3)(4) (1)(2)(3)(4) (3) (1)(2)(3)(4) (1) (2)
Informationsdarstellung mit Multimediadokumenten	• Aufbau von Präsentationsdokumenten • Objekte, Attribute und Klassen in Präsentationen • Objekte animieren	(2) (4) (1) (1)(2) (4)

Hierarchische Informationsstrukturen	• Arten von Ordnung • Bäume mit der Wurzel oben • Dateien und Ordner • Dateien verschieben, löschen • Attribute und Methoden von Ordnern und Dateien • Dateien und Dokumente	(1) (4) (4) (1)(2) (4) (3) (1) (1)
Vernetzte Informationsstrukturen - Internet	• Das World Wide Web • Informationsbeschaffung im Internet • Suchmaschinen • HTML-Dateien • Hyperlinks erstellen • Computerviren und Datenschutz • An einem Chat teilnehmen	(1)(2)(3)(4) (1)(2)(3)(4) (1)(2)(3) (2) (4) (2) (4) (3) (3)
Austausch von Informationen - E-Mail	• Das E-Mail-Objekt • Nachrichtentransport über das Internet • Verfassen, senden und empfangen von E-Mails • E-Mails mit Anhängen	(2) (4) (2) (4) (1) (3) (1)
Beschreibung von Abläufen durch Algorithmen	• Beschreibung von Abläufen – Schritt für Schritt • Einführung am Beispiel einer Programmiersprache • Wiederholung mit fester Anzahl • Bedingte Wiederholung • Bedingte Anweisung • Algorithmen überall	(1)(2) (4) (1)(2) (1)(2) (4) (1)(2) (4) (1)(2) (4) (4)

1.5 *POV-Ray*

POV-Ray steht für „The Persistence of Vision Raytracer“. *POV-Ray* ist ein 3D-Grafikprogramm, das es ermöglicht, dreidimensional wirkende Bilder (Szenen) am Computer zu erzeugen, die - bei geschicktem Einsatz dieses Werkzeugs - einer fotorealistischen Wiedergabe sehr nahe kommen können. Um dies zu erreichen, benutzt *POV-Ray* das Raytracing-Verfahren.
Raytracing ist ein Verfahren, mit dem möglichst wirklichkeitsnahe, in der Regel dreidimensionale Szenen auf einem Computermonitor dargestellt werden. Dafür werden in der Natur vorkommende Mechanismen imitiert - und zwar aus der Welt der Physik, genauer gesagt der Optik. Die Vorgehensweise wird schon im Begriff Raytracing angedeutet. "Ray" heißt Strahl, „to trace" steht unter anderem für (zurück)verfolgen. Ein Raytracer ist also ein Strahlenverfolger.

1.6 Download von *POV-Ray*

POV-Ray gibt es derzeit für die Betriebssysteme Windows, Linux und Mac OS. Die zurzeit stabile Version ist 3.6. Einige Linux-Distributionen (z.B. Suse) bringen *POV-Ray* mit - je nach Alter der Distribution ist dann aber auch die *POV-Ray*-Version unter Umständen etwas älter.

POV-Ray lässt sich von der offiziellen *POV-Ray*-Hompage herunterladen:
http://www.povray.org/download/
oder mit dem Direktlink:
http://www.povray.org/redirect/www.povray.org/ftp/pub/povray/Official/Windows/povwin362-32bit.msi
Alternativ befindet sich die Version 3.6.2 auch auf der beiliegenden CD.
Unter Windows Vista wird dringend empfohlen obige Version zu verwenden, da sonst bei Animationen Fehler auftreten können.

Außerdem findet sich auf der Homepage von Friedrich A. Lohmüller
http://www.f-lohmueller.de/pov_tut/pov__ger.htm
ein hervorragendes Tutorium.

1.7 Warum *POV-Ray* ?

- *POV-Ray* ist kostenlos erhältlich.
- *POV-Ray* eignet sich meiner Meinung nach sehr gut, um Schülern ausgewählte Programmierkonzepte anschaulich näher zu bringen. Die Beschreibungssprache fördert bei den Schülern das Verständnis der Vorgänge. Außerdem gibt es zu diesem Programm umfassende Dokumentationen der Verfahren, die für den Lehrer sehr hilfreich sind.
- Das Arbeiten mit diesem Programm ist für Kinder schnell verständlich, macht Spaß und wird auch nicht schnell langweilig, da die Schüler durch die grafische Darstellung ihrer Programme direkt sehen, was sie programmiert haben.

Näheres dazu in der Projektbeschreibung.

2. Überblick über das Projekt

Zielgruppe: Schüler der Klassenstufen 5-7
Durchführung: 5 Einheiten à 90 Minuten

2.1 Benötigte Vorkenntnisse

- Im Umgang mit dem Computer: Die Schüler sollten vertraut sein mit dem Speichern und Öffnen von Dateien, Anlegen von Ordnern, Bedienen von grafischen Benutzeroberflächen.
- Konzepte der objektorientierten Programmierung: Die Begriffe Objekt, Attribut, Methode und Klasse sollten bekannt sein. Bei einem Anschluss an das Praktikum von Frau Filler ist dies gegeben.
- *POV-Ray*: Es wird davon ausgegangen, dass die Schüler einfache Objekte mit *POV-Ray* erstellen können, Attribute setzten und Klassen erstellen können. (Im ersten Testdurchlauf dieses Projekts mit Schülern, werde ich den Schülern eine kleine Einführung in *POV-Ray* geben, da das Projekt nicht im Anschluss an die Einheiten von Frau Filler erfolgt.)

2.2 Lernziele und Inhalte dieses Projekts

Für die Inhalte dieses Praktikums war vor allem das Schulbuch „Natur und Technik, Schwerpunkt: Informatik, Bayern 6/7 Gymnasium“ ausschlaggebend.
In den ersten beiden Kapiteln „Informationen und ihre Darstellung“ und „Informationsdarstellung mit Grafikdokumenten“ werden die Schüler, nachdem sie die grafische Benutzeroberfläche kennen gelernt haben, in die Theorie der **Objektorientierung** eingeführt.
Für dieses Thema lässt sich *POV-Ray* sehr gut verwenden, da in einem Grafikprogramm die Grafiken leicht als Objekte mit Attributen und bestimmten Methoden dargestellt werden können.
Im Buch werden den Schülern anhand von Begriffen wie „Tisch“ oder „Baum“ die Begriffe Objekte, Attribute, Methoden und Klassen näher gebracht. Da Frau Filler in ihrem Praktikum diese Begriffe schon behandelt und auch weitgehend ausgeschöpft hat, beinhaltet dieses Praktikum eine Erweiterung der Methoden, indem weitere Methoden von *POV-Ray* eingeführt werden. Des Weiteren sollen die Schüler die Begriffe Vererbung und Vererbungshierarchie kennen lernen. Dies wird zwar in dem oben genannten Buch nicht behandelt, jedoch in vielen anderen Schulbüchern wie zum Beispiel in „Netzwerk Informatik 6/7 (Bayern)“.

In „Natur und Technik“ wird mit Hilfe von Vektorgrafiken in die objektorientierte Programmierung eingeführt. Es wird gezeigt, dass es einfacher ist, Grafiken mit Vektoren zu beschreiben als mit Pixel (Speicherplatz). Diese Art der Einführung wäre zum Beispiel mit dem Programm „Paint“ gut möglich, allerdings nicht mit *POV-Ray*, da *POV-Ray* nicht pixelorientiert arbeitet. Die schülergerechte Einführung, die Frau Filler beschrieben hat, ist hierfür eine sehr gute Alternative.
Auch die im Lehrbuch genannten Attribute wie Position, Größe, Farbe und Füllung und die Methoden „Position zuweisen“, „verschieben“ (*translate*), „drehen“ (*rotate*) und „Größe verändern“ (*scale*) werden in Frau Fillers Arbeit ausführlich behandelt.
In diesem Praktikum werden darauf aufbauend die Methoden *difference, intersection, union,* und *declare* eingeführt, damit die Schüler ein breiteres Spektrum an Methoden erhalten.

Fazit: Die Lerninhalte der ersten beiden Kapitel des Lehrbuches können gut mit *POV-Ray* vermittelt werden. Der Einsatz des Grafikprogramms kann hier sehr hilfreich und motivierend sein.

Das nächste Kapitel „Informationsdarstellung mit Textdokumenten“ ist eine Einführung in Textverarbeitungsprogramme. Diese Unterrichtseinheiten sollten deshalb auch besser von solchen Programmen begleitet werden. Man kann allerdings auch hier wieder auf das mit *POV-Ray* gelernte zurückgreifen. So kann man bei der Textverarbeitung zwischen den Objekten „Dokument“, „Absatz“ und „Zeichen“ unterscheiden und diesen die Attribute Seitenlänge, Spaltenanzahl, Ausrichtung, Abstand, Schriftgröße, Schriftfarbe etc. zuweisen.

Im vierten Kapitel „Informationsdarstellung mit Multimediadokumenten“ ist der Einsatz von *POV-Ray* nicht vorteilhaft, da die Schüler hier mit einem Präsentationsprogramm arbeiten sollten, auch wenn man auch bei Präsentationen die Begriffe Objekt und Attribut wieder aufgreifen kann.

In den nächsten drei Kapiteln „Hierarchische Informationsstrukturen – Dateisystem“, „Vernetzte Informationsstrukturen – Internet“ und „Austausch von Informationen – E-Mail“ ist die Zuhilfenahme von *POV-Ray* nicht sinnvoll.

Im Bereich der Algorithmen jedoch, lässt sich mit *POV-Ray* wieder einiges machen.
Im Lehrbuch wird hier der Begriff Algorithmus, eine endliche Folge von Anweisungen, mit Hilfe der Programmierumgebung „Robot Karol“ eingeführt. Ziel ist es, den Roboter Steine auslegen zu lassen. Dies geschieht mit den Befehlen LinksDrehen, Schritt, Hinlegen etc. Anschließend, um das Programmieren zu vereinfachen, werden die Schüler an *for- if-* und *while-*Schleifen herangeführt.

Für die *while*-Schleife, die *if-then-else* und auch die *switch-case* Anweisung bietet sich *POV-Ray* sehr gut an. Zudem hat es eine „schönere“ grafische Umgebung. In diesem Praktikum werden Unterrichtseinheiten entworfen, die die Schüler mit der Struktur von bedingten Anweisungen vertraut machen. Danach sollen sie das Gelernte an *POV-Ray* anwenden.

Zum Abschluss dieses Praktikums sollen die Schüler eine Animation erstellen, in der sie Objekte mit Hilfe von Methoden und Schleifen bewegen, somit also das Gelernte zusammen anwenden.

2.3 Konkrete Lernziele im Überblick

Die Schüler sollen in den Unterrichtseinheiten (UE)

Allgemein:

- motiviert werden, sich auch zu Hause mit Informatik zu beschäftigen (alle UE);
- lernen, was *POV-Ray* alles kann, und sich im besten Fall auch außerhalb des Unterrichts mit dem Programm beschäftigen (alle UE);
- Variablen benutzen können um Daten zu speichern und Zähler zu setzten (UE II,III,V);
- mit Steuerdateien, Variablen und ihrer Übergabe umgehen können sowie Datei-Konvertierung kennen lernen (UE V);
- mit dem Datentyp *Integer* umgehen können (UE III);
- erfahren, wie Filme entstehen und selbst welche erzeugen können (UE V);
- ini-Dateien schreiben können (Animationsdatei in POV-Ray) (UE V);
- die Begriffe AVI-Dateien, frames und fps kennen lernen (UE V);

Im Bereich der Objektorientierung:

- Methoden auf Objekte anwenden können (alle UE);
- Routine im Umgang mit Methoden erhalten (UE I);
- das Verhalten von Objekten kontrollieren können (UE I,V);
- das Konzept der Vererbung und Vererbungshierarchie verstehen (UE I);
- Vererbungsstrukturen nutzen, um gleichartige Objekte einfacher zu programmieren (UE I);
- in der Lage sein Dinge abstrakt als Objekte zu begreifen (UE I,V);
- Klassen definieren können (UE I,V);
- Methoden und Attribute in Klassen definieren können (UE I,V);
- die Begriffe Instanzen, Ober-/Unterklasse, Einfach- und Mehrfachvererbung verstehen (UE I);

Im Bereich Algorithmen:

- Begriffe wie „Algorithmus“, „Programm“, „Programmcode“, „Folge von Anweisungen“ verstehen (UE II,III);
- Wiederholungen (Schleifen) gezielt und sinnvoll einsetzen können (UE II,III,V);
- bestimmte Anweisungen nur unter bestimmten Bedingungen auszuführen (bedingte Anweisung (UE III) und Verzweigung (UE IV));
- den Unterschied zwischen einseitigen und zweiseitige Anweisungen kennen lernen (UE III);
- geschachtelte Anweisungen programmieren können (UE III,IV);
- die Begriffe „Syntax“, „Kontrollstrukturen“ und „Verzweigung“ in ihren Sprachgebrauch übernehmen (UE IV);
- den Unterschiede *if-else* und *switch-case* Anweisungen kennen lernen und gezielt die für ein bestimmtes Vorhaben bessere einsetzen können (UE IV);
- in der Lage sein Kontrollstrukturen richtig einzusetzen (UE III,IV,V);
- mit der Syntax von Kontrollstrukturen vertraut werden (UE II,III,IV,V);
- lernen Endlosschleifen zu verhindern (UE IV).

3. Unterrichtseinheiten

Bemerkungen

Die Arbeitsblätter werden im Anhang in Druckversion mitgeliefert.

Die Lösungen der Aufgaben und Beispiele, die im Unterricht vorgeführt werden, befinden sich auf der beiliegenden CD.

Es ist möglich den Schülern zu jedem Arbeitsblatt eine Vorlage zu geben (diese befindet sich auch auf der CD), allerdings sind die Arbeitsblätter so gestaltet, dass die Schüler die komplette Datei selbst entwerfen können. Dies soll dazu beitragen, dass die Schüler sich intensiver mit POV-Ray beschäftigen (vor allem mit der Szenenbeschreibung). Sollte jedoch die Zeit im Unterricht etwas knapp werden, ist es durchaus sinnvoll, den Schülern die Vorlage zu Verfügung zu stellen.

Des Weiten werden für die Aufgaben auf den Arbeitsblättern die Zusatz-Packages „Basic_templates“ und „Textures_and_Materials“ benötigt. Diese befinden sich auch auf der CD, können aber auch unter
http://www.f-lohmueller.de/pov_tut/down/Basic_templates.zip (7.179 kB)
http://www.f-lohmueller.de/pov_tut/down/Textures_and_Materials.zip (8.960 kB)
heruntergeladen werden.

Die Datei „Basic_templates.zip“ muss dann in das Insert-Menu-Verzeichnis im *POV-Ray*-Programm-Verzeichnis entpackt werden.
Normalerweise ist dies bei *POV-Ray* für Windows v3.62:
unter Windows XP: "C:\Programme\POV-Ray for Windows v3.6\Insert Menu\"
unter Windows Vista: "C:\Benutzer\MEIN_NAME\Dokumente\POV-Ray\v3.6\Insert Menu\"

Die Datei "Textures_and_Materials.zip" muss in das Insert-Menu-Unterverzeichnis namens "00 - Basic templates" entpackt werden.

Sollte dies zu aufwändig oder nicht möglich sein (da das Programm nicht über das Netzwerk aufrufbar ist, oder immer wieder von einigen Computern gelöscht wird, wie das in der Durchführung von Frau Filler der Fall war), dann reicht es auch aus, den Schüler die Vorlage-Dateien zu den Unterrichtseinheiten zu geben.

3.1 Einheit I (Methoden, Klassen und Vererbung)

Lernzeile/-inhalte dieser Einheit:

Objektorientierung: Methoden, Einfach-/Mehrfachvererbung
Weitere Begriffe, die eingeführt werden: Ober-/Unterklasse, Vererbungshierarchie, Instanzen
Folgende Begriffe werden als bekannt vorausgesetzt: Objekt, Attribut, Methode, Klasse

Ablauf:

Teil 1: Besprechung ohne Verwendung von *POV-Ray*:

Bisher haben die Schüler gelernt, was Objekte sind, welche Attribute sie haben können, Methoden auf die angewendet und Klassen konstruiert.

In *POV-Ray* haben sie die Objekte Kugel, Zylinder und Quader kennen gelernt, und haben diesen Objekten die Attribute Oberflächenfarbe/-struktur zugewiesen. Die Objekte wurden mit Hilfe der Methoden translate(), scale() und rotate() verändert.

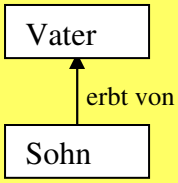
In dieser Einheit werden nun weitere Methoden vorgestellt und darüber hinaus das Konzept der Vererbung vorgestellt.

Vererbung

Die Vererbung dient dazu, aufbauend auf existierenden Klassen neue zu schaffen, wobei die Beziehung zwischen ursprünglicher und neuer Klasse dauerhaft ist. Eine neue Klasse kann dabei eine Erweiterung oder eine Einschränkung der ursprünglichen Klasse sein.
Die Vererbung wird mit der bekannten Vater-Sohn Beziehung eingeführt.

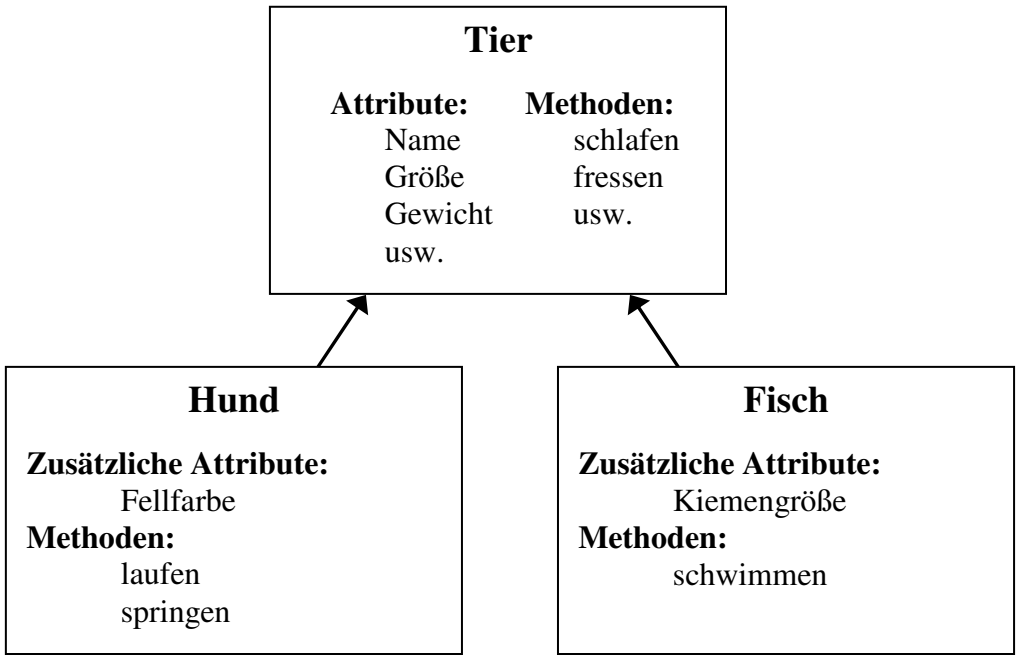
Unter-/Oberklasse

Die vererbende Klasse wird *Oberklasse* (auch *Basis-* oder *Elternklasse*) genannt, die erbende *Unterklasse* (auch *abgeleitete* oder *Kindklasse*).



Die Schüler werden dazu angehalten, sich den Mechanismus der Vererbung am Beispiel von Tieren klar zu machen.
Lehrer: "Wie können wir Tiere klassifizieren?"
Schüler und Lehrer erarbeiten an der Tafel ein Diagramm (hierarchisch) zu Tieren. Die Ergebnisse werden sofort diskutiert. Was genau ist Vererbung?

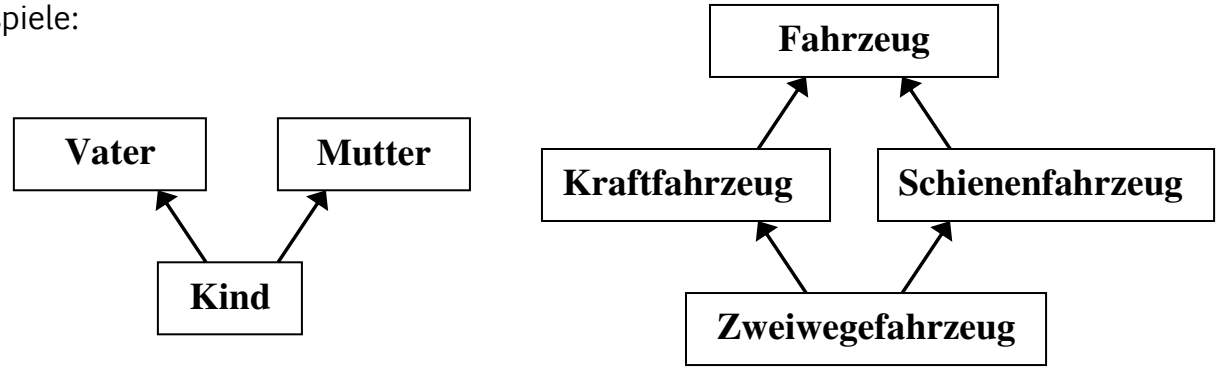
Mögliches Ergebnis (einfaches Beispiel):



Einfach-/Mehrfachvererbung

Bei diesem Bespiel handelt es sich um Einfachvererbung, da jede Unterklasse (Hund, Fisch) höchstens von einer Oberklasse (Tier) Merkmale erbt.
Um Mehrfachvererbung handelt es sich, wenn eine Unterklasse direkt von mehr als einer Oberklasse erbt.

Beispiele:



Vererbungshierarchie

Man sieht, die Vererbung kann mehrstufig eingesetzt werden. Eine Unterklasse kann wiederum Unterklasse weiterer Klassen sein. Auf diese Weise entsteht eine Vererbungshierarchie.

Teil 2: Übergang zu *POV-Ray*:

Methoden:

Es werden nun weitere nützliche Methoden eingeführt:
(Die Methode *union{}* ist bereits bekannt, allerdings nur um Objekte zu einer Klasse zusammenzufassen.)

Weitere Methoden in *POV-Ray*:

union{...}	Vereinigung: Einfaches Zusammenfassen von Objekten
difference{...}	Differenz: Abziehen eines Objektes von einem anderen
intersection{...}	Schnittmenge: Bildung der Schnittmenge von Objekten

Genauere Beschreibung der Methoden:

union{...}:
Diese Anweisung wird zum Verbinden von verschieden Objekten zu neuen Objekten benutzt, wobei die zu verbindenden Objekte nicht notwendig räumlich zusammenhängen müssen. Man kann diese Anweisung verwenden, um die Teile mehrerer Objekte gemeinsam einzufärben und/oder zusammen zu transformieren.

difference{...}:
Mittels der Differenz lassen sich aus vorhandenen Körpern Löcher, Ausbuchtungen, Einkerbungen und dergleichen herausfräsen. Das "Loch" erhält jeweils die Textur der abgezogenen Körpers.

intersection{...}:
Die Schnittmenge zweier Körper enthält diejenigen Körperbereiche, die beiden Körpern gemeinsam sind, also die Bereiche in denen sich alle dabei verwendeten Objekte überlappen.

Die Schüler werden in der Anwendung eine Klasse konstruieren, die aus mehreren Objekten besteht, auf die zuerst die oben genannten Methoden angewandt werden. Dann werden Unterklassen konstruiert und diesen zusätzliche Merkmale hinzugefügt.

Zur Wiederholung:

Klasse in *POV-Ray* konstruieren:
#declare *Klassenname* { ---Objekte innerhalb Klasse mit ihren Attributen---}

Instanzen
Eine Instanz der konstruieren Klasse aufrufen
object {*Klassenname*
translate<-2,0,0> scale ... }
(Bemerkung: Ein Objekt ist immer eine Instanz einer Klasse.)

Es besteht auch die Möglichkeit Klassen mit Makros zu definieren. Mit diesen lässt sich zwar einfacher arbeiten, aber das ist für die Unterstufe, meiner Meinung nach, etwas zu schwierig.

Teil 3: Mit *POV-Ray*:

Die Schüler sollen *POV-Ray* starten und den Anweisungen auf dem Arbeitsblatt folgen. Ihnen wird eine Basisklasse „Haus“ gegeben. Zu dieser sollen sie Subklassen erstellen (Häuser mit anderen Attributen und zusätzlichen Objekten). Diese sollen sie unter anderem mit Hilfe der neu eingeführten Methoden konstruieren.
Dabei soll nochmals deutlich werden, dass Objekte in Klassen zusammengefasst werden und Objekte immer Instanzen einer Klasse sind, Objekte Methoden usw. besitzen und dass Objekte von anderen Objekten erben.
Außerdem sollen die Schüler sich mit der Kameraeinstellung von *POV-Ray* vertraut machen, da sie diese in den nächsten Unterrichtseinheiten ab und zu modifizieren sollen.

Arbeitsblatt I – Methoden, Klassen und Vererbung

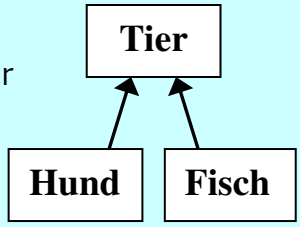
Merke:

Vererbung

Die Vererbung dient dazu, aufbauend auf existierenden Klassen neue zu schaffen, wobei die Beziehung zwischen ursprünglicher und neuer Klasse dauerhaft ist. Eine neue Klasse kann dabei eine Erweiterung oder eine Einschränkung der ursprünglichen Klasse sein.

Ober-/Unterklasse

Die vererbende Klasse wird *Oberklasse* (auch *Basis-* oder *Elternklasse*) genannt, die erbende *Unterklasse* (auch *abgeleitete* oder *Kindklasse*).



Einfach-/Mehrfachvererbung

Bei diesem Bespiel handelt es sich um Einfachvererbung, da jede Unterklasse (Hund, Fisch) höchstens von einer Oberklasse (Tier) Merkmale erbt.
Um Mehrfachvererbung handelt es sich, wenn eine Unterklasse direkt von mehr als einer Oberklasse erbt.

Methoden in *POV-Ray*:

union{...}	Vereinigung: Einfaches Zusammenfassen von Objekten
difference{...}	Differenz: Bildung der Schnittmenge von Objekten
intersection{...}	Schnittmenge: Abziehen eines Objektes von einem anderen
translate<x,y,z>	Verschieben eines Objektes
scale<x,y,z>	Größe des Objekts ändern
rotate<x,y,z>	Drehen eines Objekts

Klassen in *POV-Ray*:

Klasse in *POV-Ray* konstruieren:

#declare *Klassenname* = union{ ---Objekte innerhalb Klasse mit ihren Attributen---}
oder
#declare *Klassenname* = difference{ ... }

Eine Instanz der konstruieren Klasse aufrufen

object {*Klassenname*
translate<-2,0,0> scale ... }

Arbeitsanweisungen:

In dieser Einheit werdet ihr eine kleine Stadt programmieren. Ihr bekommt eine Vorlage, in der schon eine Haus vorhanden ist und sollt aus der Klasse „Haus“ weitere Klassen von Häusern ableiten, also Unterklassen erstellen. Verwendet dazu auch die neue gelernten Methoden.

1. Öffne das Dokument „Stadt_Vorlage.pov“.
2. Schaue dir die Klassendefinition der Klasse „Haus“ an.
3. Erzeuge einige Häuser. Setze sie nebeneinander mit der Methode translate<x,y,z>
4. Schreibe eine Unterklasse, z.B. „HausMitSchornstein“.
5. Erzeuge auch davon einige Instanzen.
6. Schreibe weitere Unterklassen: Häuser mit Fenstern und Türen. Verwende dazu die neu gelernten Methoden.
7. Experimentiere etwas mit *POV-Ray*. Verwende verschiedenen (auch vordefinierte) Objekte und setzte Attribute.

Bei Problemen mit passenden Koordinaten für Schonstein, Fenster etc. frage den Lehrer um Rat.

8. Experimentiere mit der Kameraeinstellung:
```
#declare Camera_0 = camera {/*ultra_wide_angle*/ angle 75
                            location  <1.5 , 2.0 ,-3.0>
                            right    x*image_width/image_height
                            look_at   <0.0 , 1.0 , 0.0>}
```
9. Wechsel die Kamera indem du hier eine andere Nummer einträgst: camera{Camera_0}

Zusatz:

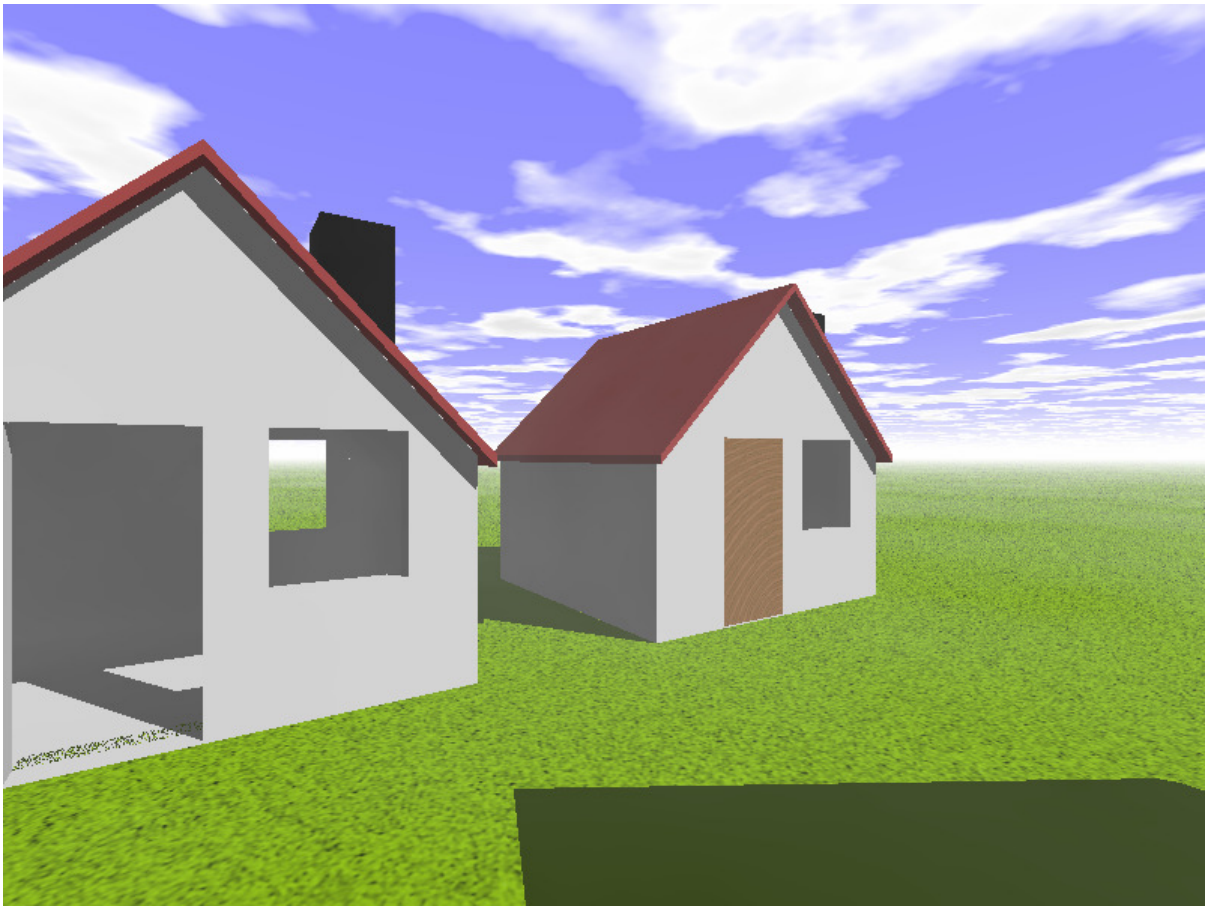
Erzeuge selbst eine neue Kamera: Camera_4 mit beliebigen Einstellungen.

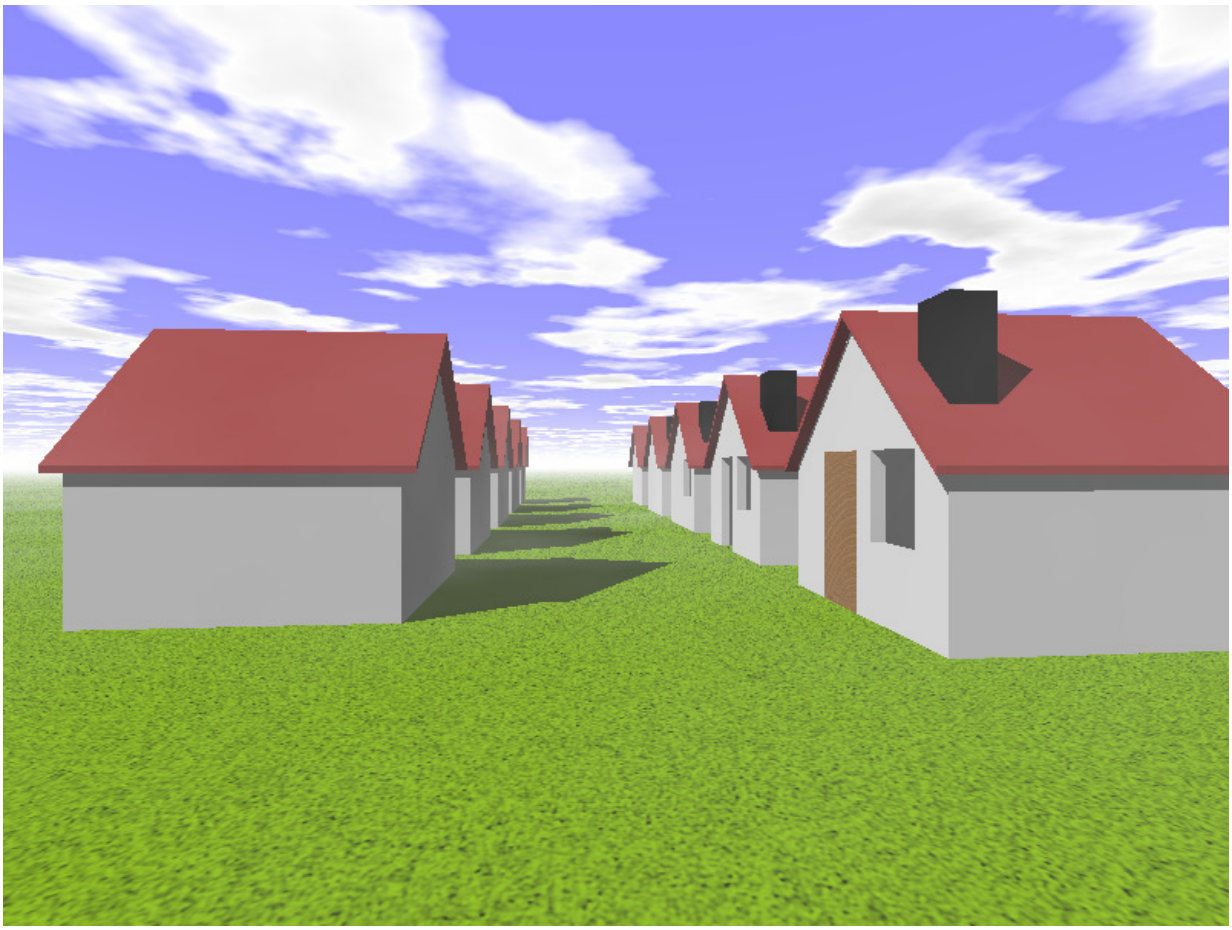
Bei Fragen an den Lehrer wenden.

Lösungen für den Lehrer

```
//neue Klasse: Haus mit Schornstein

#declare Haus_mit_Schornstein=

   union{
   object{Haus}

   //Schornstein
   box{<0.4,1.2,-0.8>, <0.7,2,-0.5>  }
   }

//neue Klasse: Haus mit Fenster

#declare Haus_mit_Fenstern=

   difference{
   object{Haus_mit_Schornstein}

   //Fenster
   box{<0.2,0.5,-1.1>, <0.7,1.1,1.1>
        texture { pigment{ color White}}
                  finish { diffuse 0.85 phong 1.0}
                 }  // end of texture
   }

//neue Klasse: Haus mit Türloch

#declare Haus_mit_Tuerloch=

   difference{
   object{Haus_mit_Fenstern}

   //Türloch
   box{<-0.5,0.01,-1.1>, <0.0,1.1,-0.8>
        texture { pigment{ color White}}
                  finish { diffuse 0.85 phong 1.0}
                 } // end of texture
   }

//neue Klasse: Haus mit Tür

#declare Haus_mit_Tuer=

   union{
   object{Haus_mit_Tuerloch}
```

```
   //Tür
   box{<-0.5,0.0,-1.0>, <0.0,1.1,-0.9>
        texture{ T_Wood17
                  normal { wood 0.15 scale 0.05 turbulence 0.1 rotate<0,0,0> }
                  finish { diffuse 0.9 phong 1 }
                 rotate<0,0,0> scale 0.5 translate<0,0,0>
                } // end of texture
        }
   }

//Ausgabe
object{Haus
    translate<-5,0,2>  }

object{Haus_mit_Schornstein
        translate<-2,0,2>  }

object{Haus_mit_Fenstern
        translate<1,0,2>  }

object{Haus_mit_Tuerloch
        translate<4,0,2>  }

object{Haus_mit_Tuer
       translate<7,0,2>  }
```





Erfahrungen aus der Praxis:

21.12.09 (10 Schüler) Informatik-AG, 6. Klasse, Bach-Gymnasium Mannheim

Die Durchführung verlief wie das oben erstellte Konzept, allerdings befassten wir uns zuerst etwas mehr mit Objekten, Attributen und Methoden, die den Schülern aus einem anderen Programm (Gamemaker) bekannt waren, da sie dieses in den Wochen zuvor in der Informatik-AG meines Vorgängers behandelt hatten. Dann übertrugen wir alles auf *POV-Ray.*
Bei den Schülern war schon sehr viel Wissen über Objekte, Attribute etc. bis hin zu Vererbung vorhanden.
Auch bei dem Beispiel Tier-Hund-Fisch brachten sie viele gute Vorschläge. (Sie wollten sogar noch mehr Tiere besprechen.)

Wir überlegten gemeinsam, welche Objekte es in einem Grafikprogramm wie *POV-Ray* geben könnte. Die Schüler nannten von sich aus fast alle (Kugel, Quader, Zylinder, Kegel). Ich stellte ihnen diese dann genauer vor, vor allem Kugel und Quader, die für uns am Anfang wichtig waren. Wir besprachen die Attribute und Methoden, die man den Objekten zuweisen kann.

Ich führte danach das 3D-Koordintensystem an der Tafel ein. Dieses wurde im Allgemeinen gut und auch schnell verstanden. Nur vereinzelte Schüler taten sich am Anfang noch schwer damit. Durch das spätere Ausprobieren im Programm ging es aber auch bei diesen.

Danach folgte die gemeinsame Installation, da man das Programm in jedem Account einzeln installieren musste.

Schwierigkeiten:

Zwei Schüler speicherten das Programm auf dem falschen Laufwerk, was ich dann korrigieren musste, bevor wir anfingen. Dadurch langweilten sich andere Schüler, wieder anderen probierten aber auch schon die Vorlage zu öffnen.

Das Programmieren gestaltete sich etwas schwierig, da die Schüler noch nie mit *POV-Ray* gearbeitet hatten und für den Anfang doch schon recht viel von ihnen verlangt wurde.

Was sich auch als schwierig herausstellte, waren die Klammern { } und < >. Kaum ein Schüler wusste, wie man diese tippen kann.

Einige Schüler hatten Probleme mit der Klammersetzung. Fehlende schließende, öffnende Klammern, was immer wieder zu Fehlermeldungen geführt hat.

Das Programm zeichnete die Häuser mit Schonstein zuerst nicht, da der Objektaufruf über der Klassendeklaration eingefügt wurde, ein Fehler den auch ich nicht sofort bemerkt habe.
Weiter wurden die Häuser mit Fenstern „nicht gezeichnet“, da die Schüler erst ein Haus ohne Fenster weiter oben in ihrem Programm erzeugt hatten und an die gleiche Stelle im Koordinatensystem weiter unten im Programm ein Haus mit Fenstern setzten. Somit wurde das Fenster des zweiten Hauses von den „nicht Fenstern“ des ersten Hauses ausgefüllt und war deshalb nicht zu sehen.

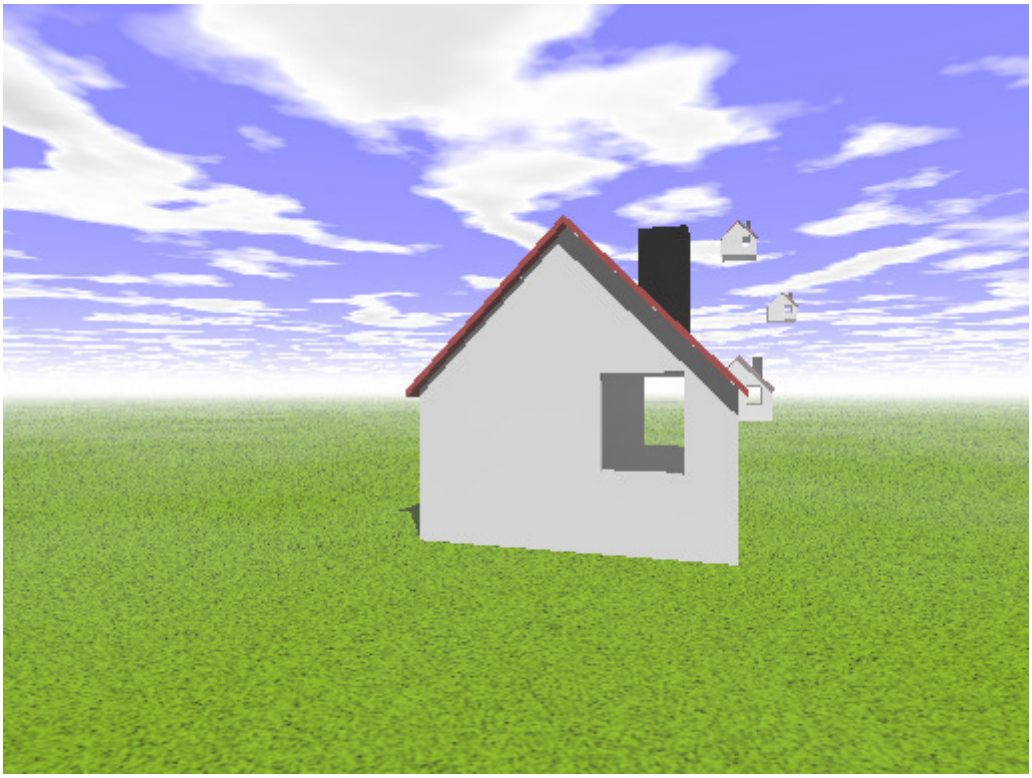
Fazit:

Auch wenn nicht alles auf Anhieb geklappt hat und wir nicht mehr zur Kameraeinstellung gekommen sind, so haben die Schüler doch viel gelernt.

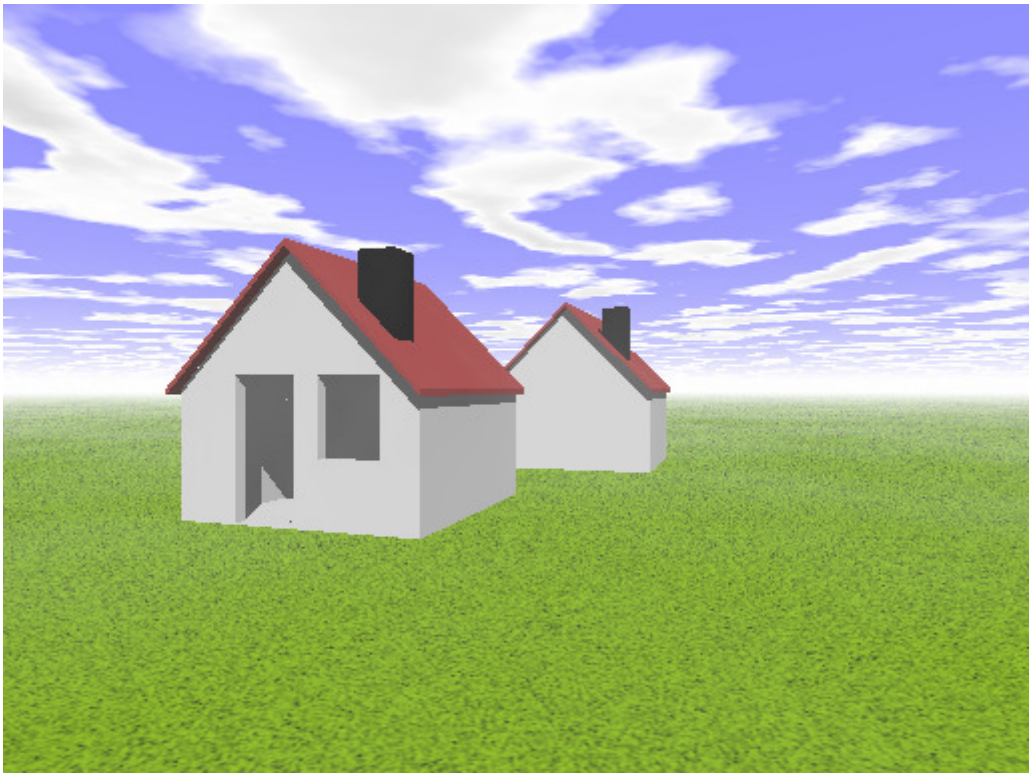
Die Schüler haben sehr gut mitgemacht, auch wenn es schwer war. Sie probierten immer erst selbst etwas zu programmieren und wenn viele nicht weiterkamen, wurde es zusammen gemacht.

Mit Schülern, die schon am Kurs von Frau Filler teilgenommen haben, würde man aber das auch schaffen können.

Schülerergebnisse



Der Schüler hatte Spaß daran, die Häuser absichtlich in der Luft schweben zu lassen.



3.2 Einheit II (Die *while*-Schleife)

Lernzeile/-inhalte dieser Einheit:

Variablen: Variablen verwenden um Zähler zu setzen
Bedingte Anweisungen: *while*-Schleife
Weitere Begriffe, die eingeführt werden: Algorithmus, Programm, Sequenz (Folge), Zählschleife, Bedingung, bedingte Anweisung

Ablauf:

Teil 1: Besprechung ohne Verwendung von *POV-Ray*:

Algorithmus

Definition: Ein **Algorithmus** ist eine Verarbeitungsvorschrift, die aus einer endlichen Folge von eindeutig ausführbaren Anweisungen besteht.
(Quelle: Natur und Technik)

Die Schüler werden dazu angehalten, mit ihren eigenen Worten zu erklären, was sie denken, was nach der obigen Definition ein Algorithmus ist.
Es werden die Begriffe Verarbeitungsvorschrift, endliche Folge und eindeutig ausführbare Anweisung geklärt.

Die Schüler werden aufgefordert, sich Abläufe aus dem Leben oder dem Mathematikunterricht zu überlegen, die der Definition eines Algorithmus entsprechen.
Beispiele hierfür wären:

- Wechseln eines Autoreifens
- Mathematische Lösungsverfahren, z.B.: Konstruieren einer zur Geraden g parallelen Geraden h durch einen Punkt P, der nicht auf g liegt
- Spielregeln
- Gebrauchsanweisungen
- Kochrezepte

Es werden die Schülerbeispiele besprochen. Welche davon sind wirklich Algorithmen und welche nicht? Warum nicht?
Hier soll auf die Begriffe Endlichkeit und Eindeutigkeit Wert gelegt werden.

Der Begriff Programm wird eingeführt:

Programm:
Die Darstellung eines Algorithmus in einer für den Computer verständlichen Form heißt Programm.

Teil 2: Übergang zu *POV-Ray*:

Die Schüler sollen sich überlegen, was sie *POV-Ray* sagen würden, wenn sie erreichen wollen, dass ihr Programm viele Kugeln erzeugt und diese dann entlang einer Linie nebeneinander legt.

Dies soll darauf hinausführen, dass wir eine Folge von Anweisungen (**Sequenz**) brauchen, die immer eine Kugel neben einer anderen platziert.

1) Erstelle Kugel, setze sie an die Position <x,y,z>
2) Erstelle Kugel, setze sie an die Position <x+1,y,z>
3) Erstelle Kugel, setze sie an die Position <x+2,y,z)
...

Die Schüler sollen bemerken, dass diese Art, dem Programm eine Folge von Anweisung dieser Sorte zu übergeben, sehr lästig ist.

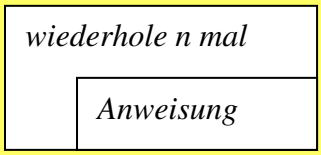
Möglicherweise gibt es Schüler, die gleich die Möglichkeit der Verwendung von Schleifen einbringen. Ansonsten sollen die Schüler dorthin geführt werden.

Nehmen wir an, wir wollen genau vier Kugeln nebeneinander setzen. In diesem Fall müssen wir die Anweisung „Erstelle Kugel, setze sie an Position <?,?,?>“ viermal wiederholen. Der Begriff **Zählschleife** (=Wiederholung mit fester Anzahl) wird eingeführt.

Das soll zu folgendem Tafelanschrieb führen:

Wiederholung mit fester Anzahl (Zählschleife):

Wiederhole *n* mal
    Anweisung
*wiederhole



(Die Anweisungsfolge innerhalb der Wiederholung wird so oft ausgeführt, wie die Zahl n vorgibt. Die Anweisung, die wiederholt werden soll, steht eingerückt zwischen den Schlüsselwörtern „Wiederhole *n* mal“ und „*wiederhole“.)

Variablen

Es wird erklärt, dass es sich bei der Anzahl der Wiederholungen *n* um eine Variable handelt.
Unter einer Variablen versteht man einen Namen, unter welchem ein bestimmter Wert gespeichert werden kann
Man kann *n* also bei jedem Programmdurchlauf einen anderen Wert zuweisen, je nach dem, wie oft man die Anweisung wiederholt haben möchte.

Bemerkung für den Lehrer:
Da es in *POV-Ray* keine *for*-Schleifen gibt, werden die Schüler dahin geführt, die Zählschleife in eine *while*-Schleife umzuwandeln.

Aufgabenstellung: Angenommen wir wollen die Anweisung nicht nur *n* mal wiederholen, sondern so oft, bis die Kugeln über die Bildbegrenzung hinausgehen.

Die Schüler sollen auf den Begriff „Wiederholung mit Eingangsbedingung“ kommen, oder dorthin geführt werden. Es soll nur solange wiederholt werden, wie die Bedingung „Die Kugel liegt im sichtbaren Bereich“ wahr ist.

Methoden und Ausdrücke, die nach ihrer Ausführung den Wahrheitswert „wahr“ oder „falsch“ annehmen heißen **Bedingungen**.

Wiederholung mit Eingangsbedingung: (*while*-Schleife) (*while* = solange, während)

Solange Bedingung tue
    Anweisung
*solange

(Zuerst wird die Bedingung überprüft. Ist sie erfüllt, wird die Anweisung ausgeführt. Solange die Eingangsbedingung wahr ist, wird dieser Vorgang wiederholt. Die Anweisung, die wiederholt werden soll, steht eingerückt zwischen den Schlüsselwörtern „solange … tue“ und „*solange“.)

Umwandlung von Zählschleifen in *while*-Schleifen:
Wenn man eine Zählschleife, die eine Anweisung *n* mal wiederholt, in eine *while*-Schleife umwandeln möchte, braucht man eine Variable x, die von 0 bis n hochgezählt wird.
Die *while*-Schleife sieht dann wie folgt aus:

Definiere $x = 0$
Solange $x < n$ tue
    Anweisung
    Definiere $x = x+1$ (x wird um eins hochgezählt)
*solange

Wenn x soweit hochgezählt wurde, dass $x = n$, bricht die Schleife ab, da die Bedingung $x < n$ nicht mehr erfüllt ist.

Teil 3: Mit *POV-Ray*:

Die Schüler sollen *POV-Ray* starten und den Anweisungen auf dem Arbeitsblatt folgen. Sie lernen einige elementare Anwendungen der *while*-Schleife in *POV-Ray*. Hierbei wird die Schleife zum regelmäßigen Anordnen von Objekten verwendet (durch Translations- und Rotationsbewegungen)

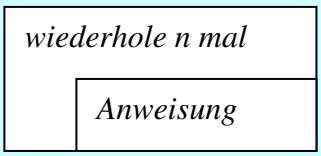
Arbeitsblatt II – Die *while*-Schleife

Merke:

Ein **Algorithmus** ist eine Verarbeitungsvorschrift, die aus einer endlichen Folge von eindeutig ausführbaren Anweisungen besteht.
Die Darstellung eines Algorithmus in einer für den Computer verständlichen Form heißt **Programm.**

Zählschleife:
Wiederholung mit fester Anzahl:

```
Wiederhole n mal
    Anweisung
*wiederhole
```



(Die Anweisungsfolge innerhalb der Wiederholung wird so oft ausgeführt, wie die Zahl n vorgibt. Die Anweisung, die wiederholt werden soll, steht eingerückt zwischen den Schlüsselwörtern „Wiederhole *n* mal“ und „*wiederhole“.)

Methoden und Ausdrücke, die nach ihrer Ausführung den Wahrheitswert „wahr“ oder „falsch“ annehmen heißen **Bedingungen**.

While-Schleife:
Wiederholung mit Eingangsbedingung

```
Solange Bedingung tue
    Anweisung
*solange
```

(Zuerst wird die Bedingung überprüft. Ist sie erfüllt, wird die Anweisung ausgeführt. Solange die Eingangsbedingung wahr ist, wird dieser Vorgang wiederholt. Die Anweisung, die wiederholt werden soll, steht eingerückt zwischen den Schlüsselwörtern „solange … tue“ und „*solange“.)

Um die Schleife n mal durchzulaufen, muss eine Variable definiert werden, die von 0 bis n hochgezählt wird. Variablen werden mit #declare definiert.

Arbeitsanweisungen:

Nun programmiert ihr selbst einige elementare Anwendungen der *while*-Schleife in *POV-Ray*. Verwendet hierbei die Schleife (engl.: loop) zum regelmäßigen Anordnen von Objekten.
Programmiert erst einmal eine einfache Schleife, welche kleine Kugeln längs der x-Achse platziert:

1. Lege ein neues Dokument an.
2. Speichere es in deinem *POV-Ray* Ordner unter dem Namen: „Kugeln.pov“
3. Füge folgende Szene ein: Insert -> Basic templates -> Ready-made scenes -> Basic Scene M4
4. Füge in den Code **unter** der Szenenbeschreibung eine beliebige Kugel ein (Insert -> Basic templates -> Shapes 0 with textures -> sphere with texture)
5. Gibt der Kugel einen Namen, z.B: „Ball“ (mit `#declare Ball =`)
6. Überlege wie oft die Kugel nebeneinander gesetzt auf die x-Achse passt (auch in die negative Richtung)
7. Führe passende Variablen für Zählbeginn und –ende mit `#declare` ein (beachte: Position der Kugeln: <0+*n*,0,0>)
8. Eine *while*-Schleife wird wie folgt programmiert:

```
#while (Bedingung)
        Anweisung
#end
```

9. Überlege, wie man die Schleife zum Abbruch bringt (Variable hochzählen?)

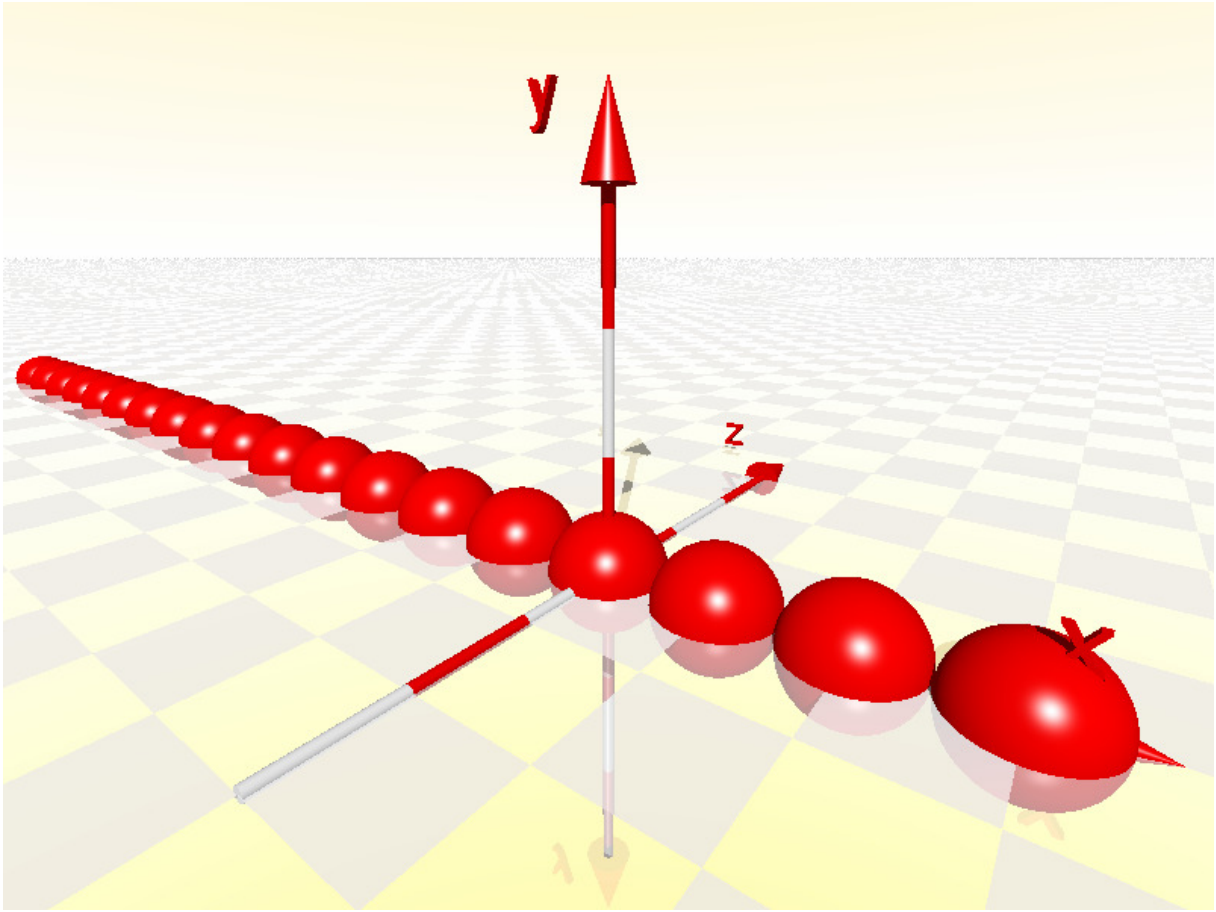
Bisher wurde die *while*-Schleife nur mit Translations-Bewegungen (Parallelverschiebungen) angewandt. Ein interessanter Effekt ergibt sich, wenn man in gleicher Weise Rotationsbewegungen ausführen lässt.

Zusatz: Speichere dein Dokument unter dem Namen „KugelKreis.pov“ ab. Ordne die Kugeln in Kreis um den Ursprung des Koordinatensystems an. Überlege dir hierfür, aus wie vielen Kugel dein Kreis bestehen soll und in welchem Winkel die einzelnen Kugeln gedreht werden müssen (um den Ursprung).
Programmiere nach Belieben noch andere Schleifen mit anderen Objekten.
Schau dir unter Insert -> Basic templates -> While loops, Spline curves an, was man noch alles mit Schleifen programmieren kann.

Bei Fragen an den Lehrer wenden.

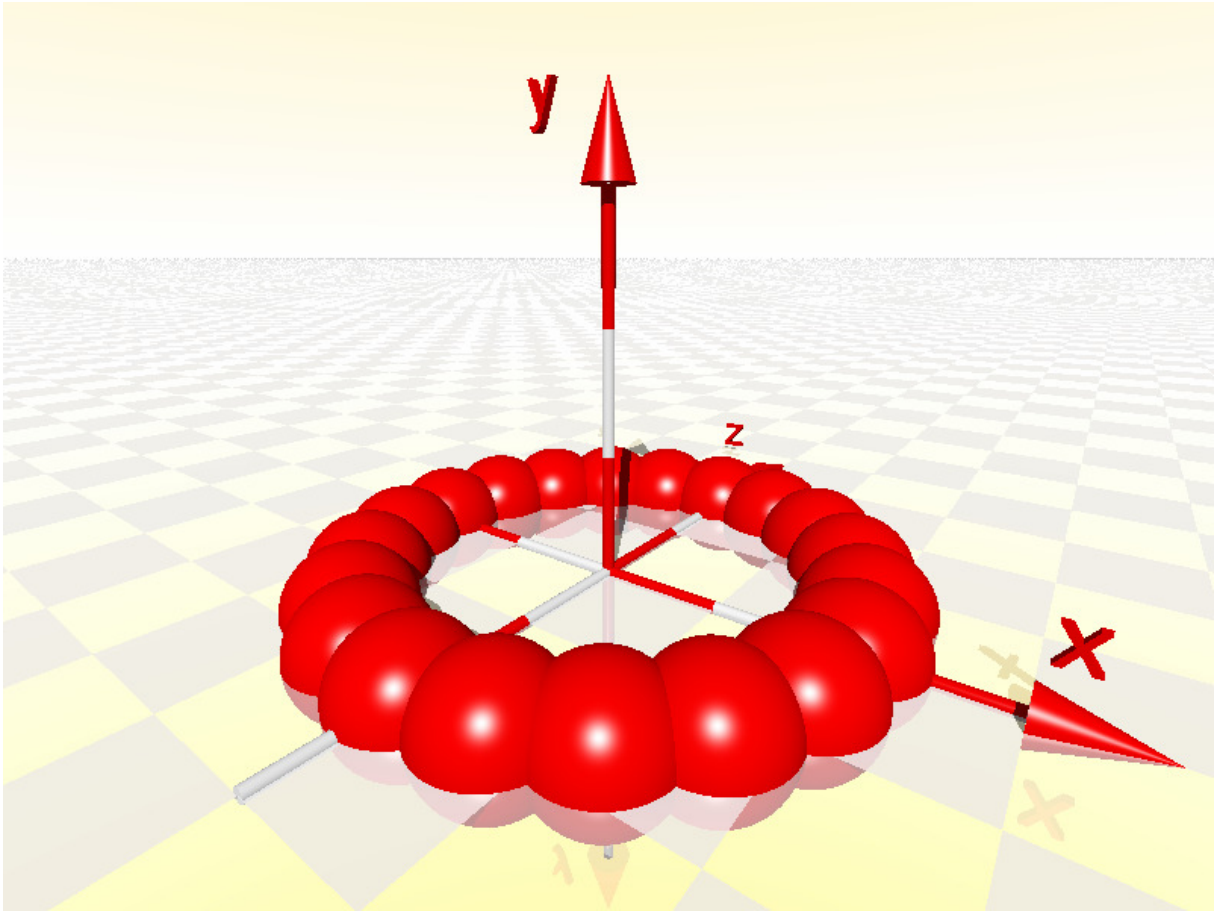
Lösungen für den Lehrer

```
#declare Ball =
 sphere{<0,0,0>,0.5
              texture{ pigment{ color Red}
               finish { phong 1}}
      }//---------------------------------------------

#declare Nr = -15;     // start
#declare EndNr = 3;   // end
#while (Nr< EndNr+1)

   object{Ball translate<Nr,0,0>}

 #declare Nr = Nr + 1;  // next Nr

#end // ---------- end of loop ------------------------------------
```



Zusatzaufgabe:

```
#declare Radius = 2.00;

#declare KreisNr = 0;     // start
#declare KreisEndNr = 20; // end
#while (KreisNr< KreisEndNr)
   object{Ball
       translate<Radius,0,0>
        rotate<0,KreisNr*360/KreisEndNr,0>}
 #declare KreisNr=KreisNr+1;
#end
```



Erfahrungen aus der Praxis:

11.01.10 (6 Schüler)

Beim Thema dieser Stunde handelte es sich um ein neues Thema für alle. Jedoch klärten wir am Anfang erst noch, warum das mit den Häusern bei einigen nicht so funktioniert hatte, wie gewünscht.

Die Schüler, die schon vor der Stunde da waren, haben schon von selbst zum Spaß probiert, in ihrem Häuser-Dokument Kugeln etc. einzufügen.

Der Begriff Algorithmus war allen unbekannt, wurde aber gut verstanden.
Ein Schülerbeispiel war: Ablauf beim Bus fahren (einsteigen, abstempeln ...)
Auf die Frage, was denn ein Programm sei, beschrieben die Schüler Programme wie MS Word etc. Dass das, was wir selbst schreiben, auch ein Programm ist, war ihnen neu.

Wir kamen nun zur „Folge von Anweisungen". Sofort kam von den Schülern, dass man den Text („erstelle Kugel") kopieren und immer wieder einfügen kann. Bei der Frage nach 1000 Kugeln sahen sie ein, dass das nicht die beste Lösung sein kann. Sie wollten dem Computer sagen, dass er das 1000-mal machen soll.
Der Begriff und die Definition der Zählschleife waren ihnen neu. Aber sie verstanden es sofort, auch die Definition einer Variablen.

Sie ließen sich schnell an die *while*-Schleife heranführen (Sie konnten *while* noch nicht übersetzen, also tat ich es, dann war die Schleife kein Problem mehr), und haben diese auch verstanden, was man später bei der Arbeit mit *POV-Ray* merkte.

Das Hochzählen der Variablen machte erstaunlicherweise auch keine Probleme. Sie waren sehr begeistert von den Schleifenkonzepten und wollten sofort programmieren.
So ließ ich sie dann auch recht früh, da die Theorie durch die gute Mitarbeit recht schnell behandelt war.

Sie fingen an sich durch das Arbeitsblatt durchzuarbeiten.

Was keiner auf Anhieb verstand, waren die Anweisungen:
Insert->...
Sie tippten diese ab.
Da es aber nur wenige Schüler waren, konnte ich schnell eingreifen und klarstellen was gemeint ist.

Sie schauten die verschiedenen Szenen an und ich erlaubte ihnen eine ihrer Wahl zu nehmen, da heute nur so wenige da waren, dass ich viel Zeit für die einzelnen Schüler hatte und bei Problemen mit der Kameraeinstellung schnell helfen konnte.

Auch die Kugel durften sie sich aussuchen. So kamen am Ende komplett verschiede Bilder heraus und die Schüler waren auch vom Ergebnis ihres Nachbarn begeistert.

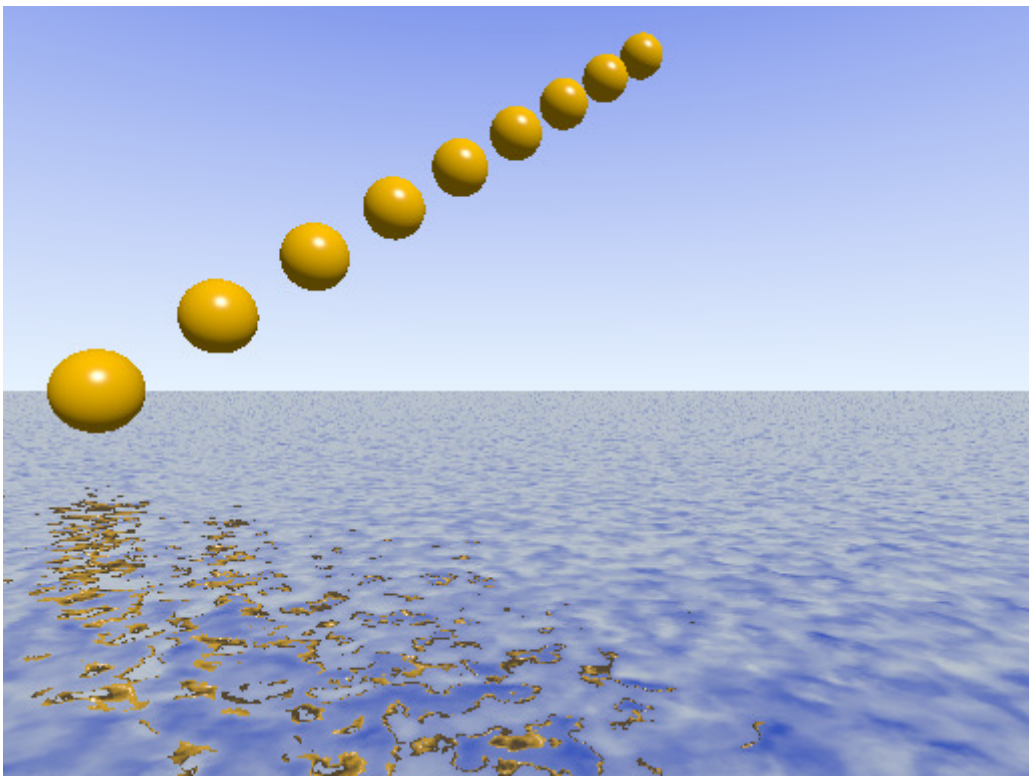
Ich ließ ihnen etwas Zeit die Szenen auszuprobieren und andere Objekte einzufügen, da sie so begeistert waren.
Dann aber sollten sie die erste Schleife schreiben, dies lief sehr gut, es brauchte nur etwas Hilfe bei den Klammern und der Deklaration der Variablen. Die Schüler richteten die Kugel später auch an der y-Achse aus oder stellten sie schräg.

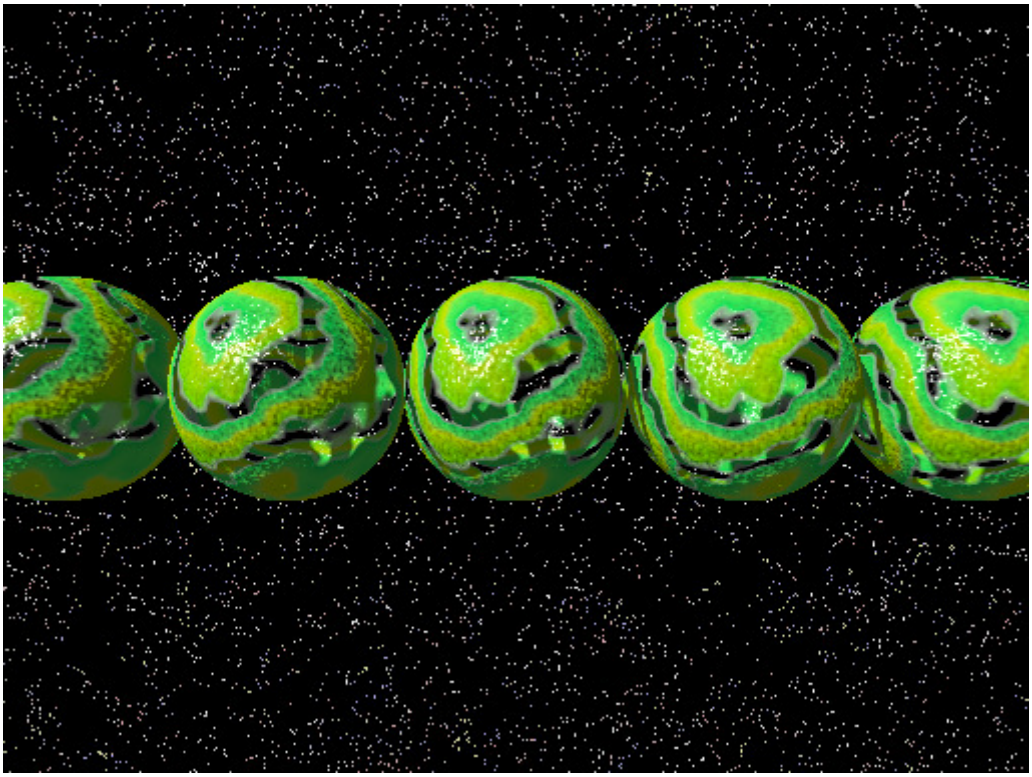
Dann machten alle die Zusatzaufgabe. Das Verstehen des *rotate* dauerte etwas länger, aber am Ende hatten es alle. Es machte ihnen Spaß nicht nur 20 Kugeln im Kreis anzulegen, sondern gleich 1000, die dann nur noch als Ring zu erkennen waren.

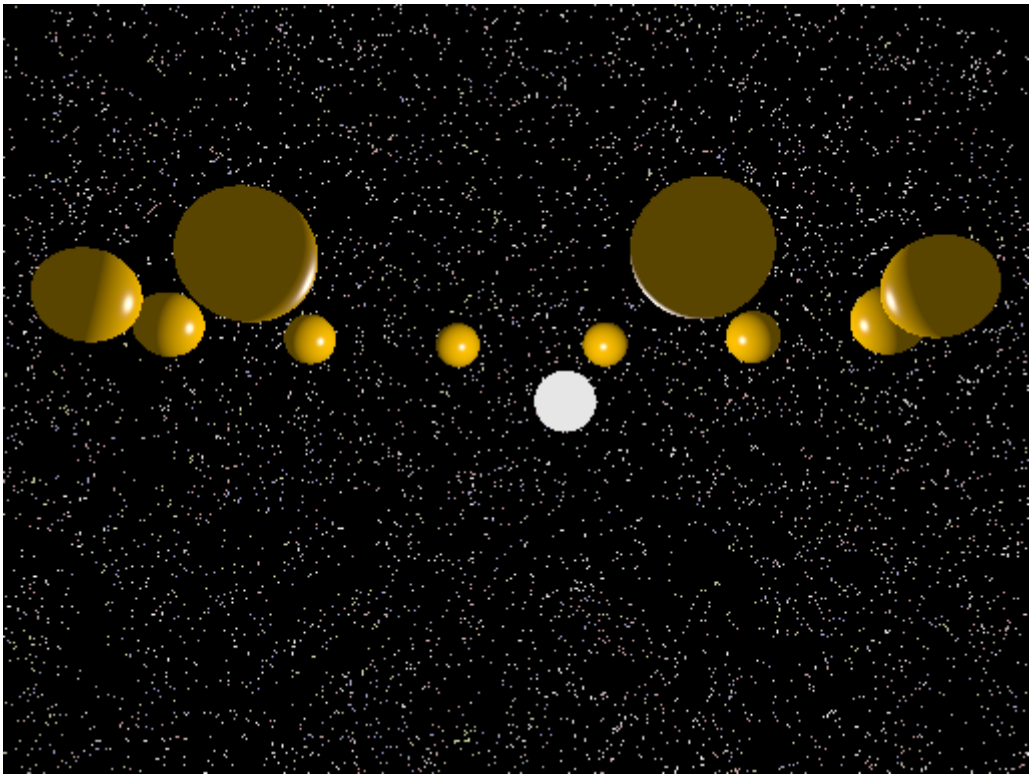
So konnten wir auch das Thema Kameraeinstellung durchnehmen. Die Schüler veränderten diese zuletzt vollkommen selbstständig.

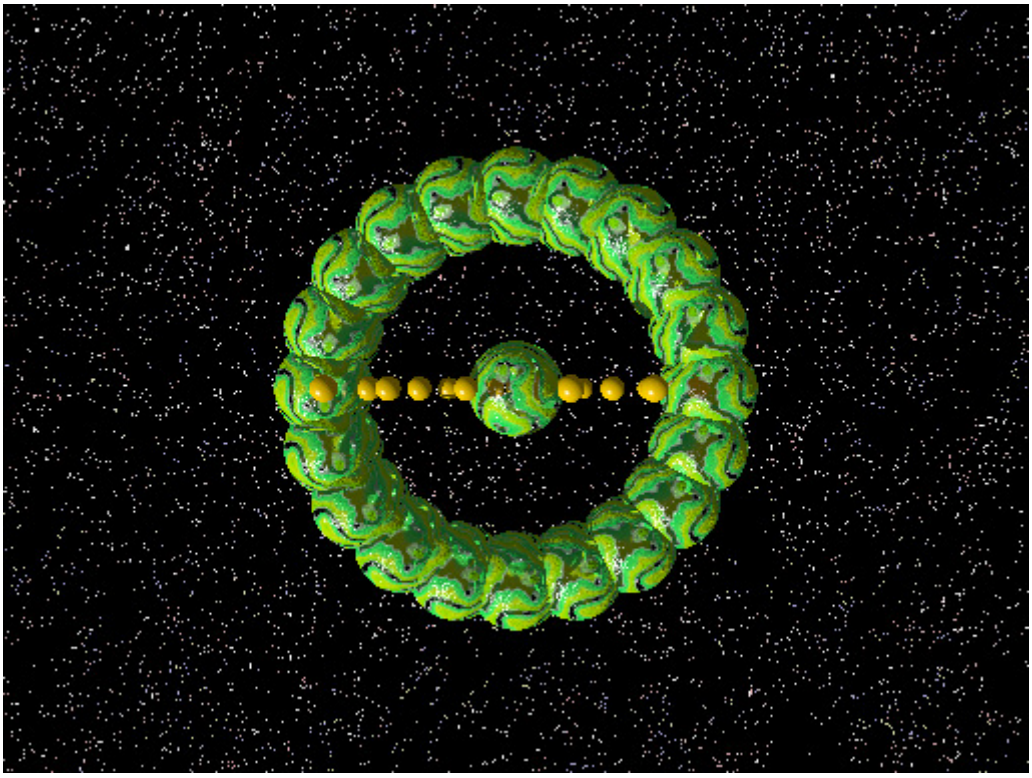
Am Ende waren die Schüler alle mit ihrem Resultat zufrieden. Ich erlaubte ihnen schon 5 min früher einzupacken, aber die meisten blieben noch um noch mehr auszuprobieren.

Schülerergebnisse









3.3 Einheit III (*if-else* Anweisungen)

Lernzeile/-inhalte dieser Einheit:

Bedingte Anweisungen: *if-else* Abfrage, if-else in Schleifen
Weitere Begriffe, die eingeführt werden: einseitige/zweiseitige Anweisungen, Programmcode, geschachtelte Anweisungen, Datentyp Integer

Ablauf:

Teil 1: Besprechung ohne Verwendung von *POV-Ray*:

Entscheidungen und ihre Folgen im Alltag

Ein Beispiel: Wenn es regnet, öffne ich den Schirm

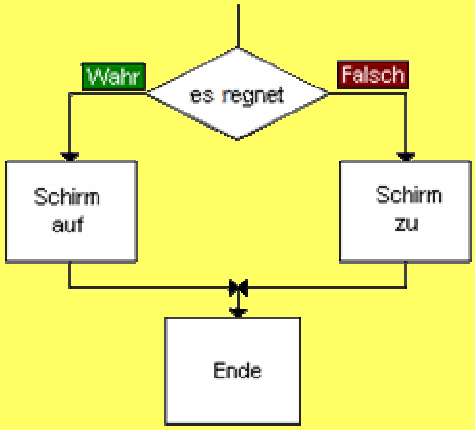
Die Schüler werden aufgefordert weitere ihnen bekannte Entscheidungssituationen mit Folgen aus ihrem Alltag zu nennen.

Bedingung (wenn) Folge (dann)

Wenn es regnet, dann öffne ich den Schirm.
Wenn ich müde bin, dann gehe ich schlafen.
...

weitere Folge (sonst)

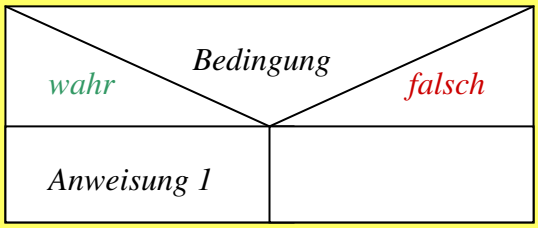
...sonst lasse ich den Schirm zu.
...sonst gehe ich spielen.
...



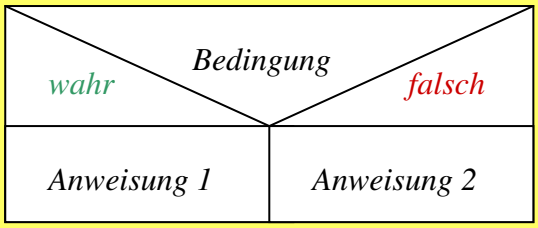
Ein-/zweiseitige Anweisungen

In der Informatik sagt man, es handelt sich um hier um zweiseitige Anweisungen. Lässt man die „sonst“ – Anweisung weg, so erhält man eine einseitige Anweisung. Es wird in diesem Fall nicht vorgegeben was passieren soll, wenn es nicht regnet.

Einseitige Anweisungen



Zweiseite Anweisungen



Teil 2: Übergang zu *POV-Ray*:

Programmcode

Programmcode ist in der Informatik der für Menschen lesbare, in einer Programmiersprache geschriebene Text eines Computerprogramms.
Im Englischen heißt die Konstruktion „wenn – dann – sonst“:
„if – then – else“ oder kurz „if – else“.
Deshalb müssen wir, wenn wir *POV-Ray* eine „wenn – dann – sonst“ Anweisung geben wollen, folgende Konstruktion verwenden:

Bedingte Anweisung:

wenn Bedingung dann
    Anweisung 1
sonst
    Anweisung 2
*wenn

Programmcode:

```
#if (Bedingung)
 ... Anweisung 1
#else // (optional)
 ... Anweisung 2
#end
```

Geschachtelte Anweisungen

Betrachten wir wieder unser Beispiel von vorhin.
Es gibt nun ja auch Fälle, bei denen man beim *else* weitere Sachen abfragen will, z.B. könnte ja auch die Sonne scheinen:

Geschachtelte Anweisungen

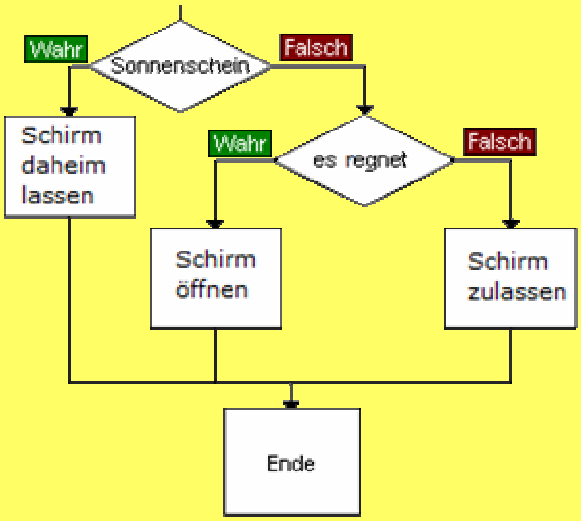
Wenn die Sonne scheint,

dann lass den Schirm daheim,

sonst

    wenn es regnet
    dann öffne den Schirm
    sonst lasse ihn zu
    ende

ende



Datentyp Integer

Mit **Integer** (englisch für ganze Zahl) wird in der Informatik ein Datentyp bezeichnet, der ganzzahlige Werte speichert. Der Wertebereich ist endlich.
Wenn man zum Beispiel prüfen möchte, ob eine Zahl *n* ganzzahlig ist, so kann man das in *POV-Ray* wie folgt machen:

#if(*n* = int(*n*)) ...

Das kann man dann dazu verwenden, um zu testen ob eine Zahl *n* gerade oder ungerade ist:

#if(n/2 = int(*n*/2)) ...

Nach diesem Teil der Unterrichtseinheit sollte klar sein, was *if-else* Anweisungen sind. Kommen wir nun zur Anwendung.

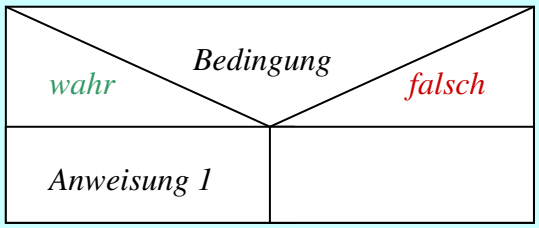
Teil 3: Mit *POV-Ray*:

Die Schüler sollen *POV-Ray* starten und den Anweisungen auf dem Arbeitsblatt folgen. Sie lernen dabei einige Anwendungsmöglichkeiten von *if-else* Bedingungen kennen.
Des Weiteren sollen bisher erlangte Kenntnisse im Umgang mit *while*-Schleifen zur Wiederholung wieder eingesetzt werden, indem die *if-else* Anweisungen zunächst in *while*-Schleifen gepackt werden. Später werden die *if-else* Anweisungen auch direkt in die Objektbeschreibung geschrieben.

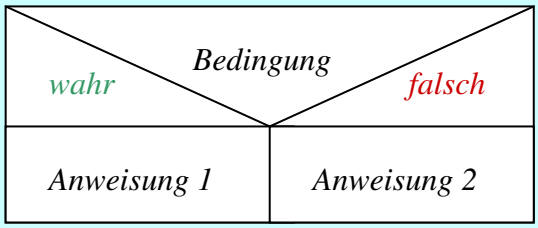
Arbeitsblatt III – *if-else* Anweisungen

Merke:

Einseitige Anweisungen



Zweiseite Anweisungen



Eine *if-else* Anweisung besteht aus einer Bedingung und einer oder mehreren Anweisungen. Wenn im Programm die *if-else* Anweisung erreicht wird, dann wird erst die Bedingung ausgewertet, und wenn sie zutrifft (und nur dann) wird anschließend die Anweisung 1 ausgeführt. Trifft die Bedingung nicht zu, so wird die Anweisung 2 ausgeführt, wenn sie existiert.
Danach wird das Programm nach der *if-else* Anweisung fortgesetzt.

Bedingte Anweisung:

```
wenn Bedingung dann
        Anweisung 1
sonst
        Anweisung 2
*wenn
```

Programmcode:

```
#if (Bedingung)
 ... Anweisung 1
#else // (optional)
 ... Anweisung 2
#end
```

Geschachtelte Anweisungen
(Beispiel:)

```
wenn Bedingung A dann
        Anweisung A1
sonst
        wenn Bedingung B dann
                Anweisung B1
        sonst
                Anweisung B2
        *wenn
*wenn
```

Arbeitsanweisungen:

Nun programmiert ihr selbst einige elementare Anwendungen der *if-else* Anweisung in *POV-Ray*. Verwendet die Anweisung hierbei, um Objekten unterschiedliche Farbe, Position oder Form zu geben.
Programmiert einen Zaun aus Holzpfählen mit unterschiedlichen Holztönen. Verschachtelt dazu die *if-else* Anweisung in einer *while*-Schleife.

1. Lege ein neues Dokument an.
2. Speichere es in deinem *POV-Ray* Ordner unter dem Namen: „Zaun.pov“
3. Füge folgende Szene ein: Insert -> Basic templates -> Ready-made scenes -> Basic Scene 04
4. Verändere in der Kameraeinstellung:
   location <0.0 , 4.0 ,-8.0>
   look_at <0.0 , 0.0 , 6.0>
5. Füge in den Code **unter** der Szenenbeschreibung einen hohen dünnen Zylinder ein, der wie ein Zaunpfeiler aussieht. Verwende hierzu einen Holzton aus Insert -> Basic templates -> Textures and Materials -> Woods (Höhe: 3, Radius: 0.4)
6. Füge noch einen Pfahl mit einem anderen Holzton ein.
7. Gib den Pfählen Namen, z.B: „Pfahl_Holzton_1“ (mit #declare)
8. Überlege wie lang der Zaun sein soll. (Errichte ihn entlang der x-Achse).
9. Führe passende Variablen für Zählbeginn und –ende der *while*-Schleife ein.
10. Füge innerhalb der Schleife eine *if-else* Anweisung ein, die dafür sorgt, dass jeder zweite Pfahl von einer anderen Holzart ist.

Zusatz: Speichere dein Programm unter „ZaunKomplett.pov“
ab und schreibe es so um, dass du einen Zaun erhältst, der eine quadratische Fläche einzäunt.

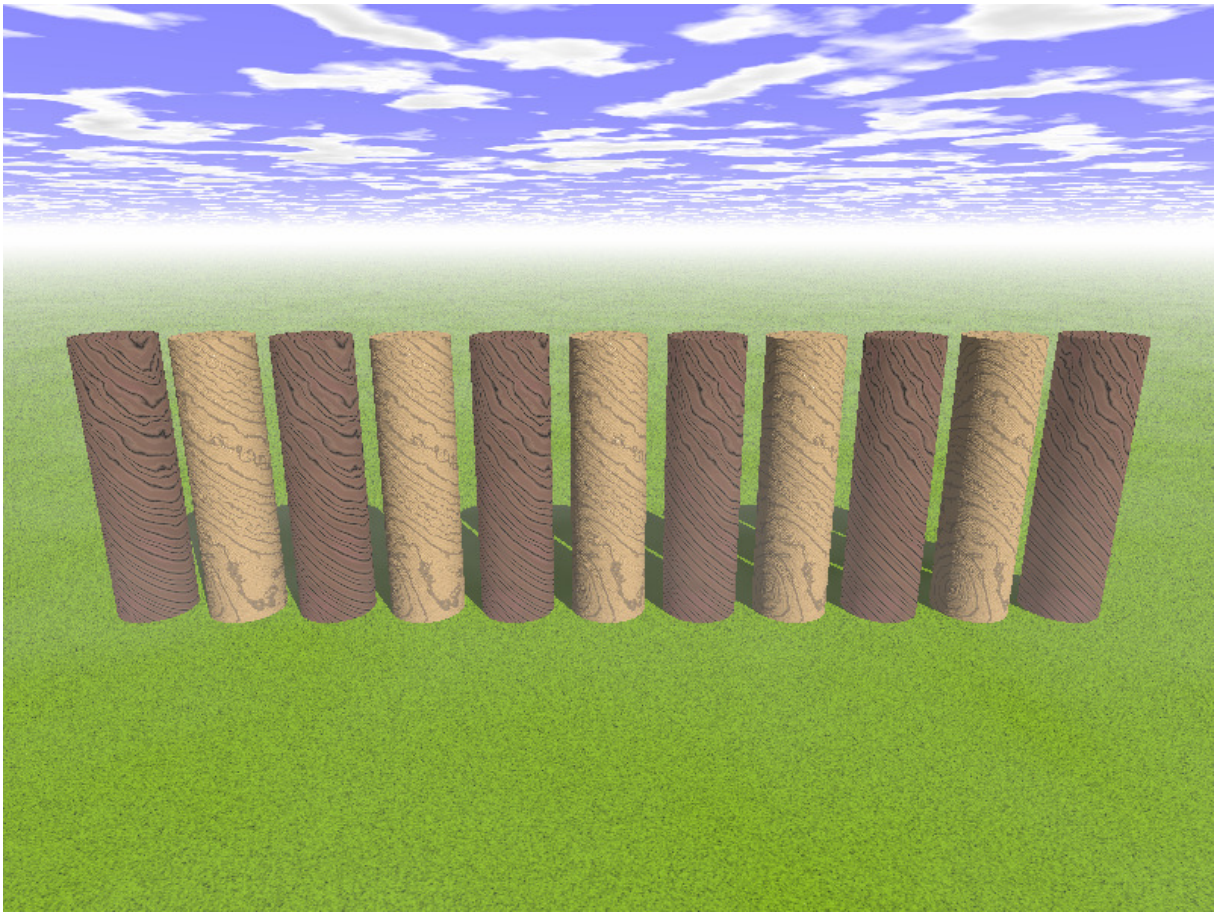
Die Ecken der quadratischen Fläche könnten folgende Koordinaten haben:
<-5,0,0>, <5,0,0>, <5,0,10>, <-5,0,10>

Versuche dich nach Belieben an weiteren geschachtelten Anweisungen mit verschiedenen Objekten.

Bei Fragen an den Lehrer wenden.

Lösungen für den Lehrer:

```
#declare Pfahl_Holzton_1 =
 cylinder { <0,0,0>,<0,3,0>,0.4

        texture{ Tan_Wood
            normal { wood 0.5 scale 0.3 turbulence 0.1}
            finish { diffuse 0.9 phong 1 }
            rotate<60,0,45> scale 0.1
          } // end of texture

}//--------------end of cylinder------------------


#declare Pfahl_Holzton_2 =
 cylinder { <0,0,0>,<0,3,0>,0.4

             texture{ Rosewood
            //normal { wood 0.5 scale 0.3 rotate<0,90,0> turbulence 0.1}
            finish { diffuse 0.85 phong 1}
            rotate<60,0,45> scale 0.5
          } // end of texture

}//--------------end of cylinder------------------


#declare Nr = -5;     // start
#declare EndNr = 5;   // end

#while (Nr< EndNr+1)

    #if(  Nr/2 = int(Nr/2) ) // only with even numbers Nr
        object{Pfahl_Holzton_1 translate<Nr,0,0>}

    #else // with odd numbers Nr
        object{Pfahl_Holzton_2 translate<Nr,0,0>}

    #end // ---end of "#if( Nr/2 = int (Nr/2))"----


#declare Nr = Nr + 1;  // next Nr

#end // ---------------- end of loop ------------------------------------
```

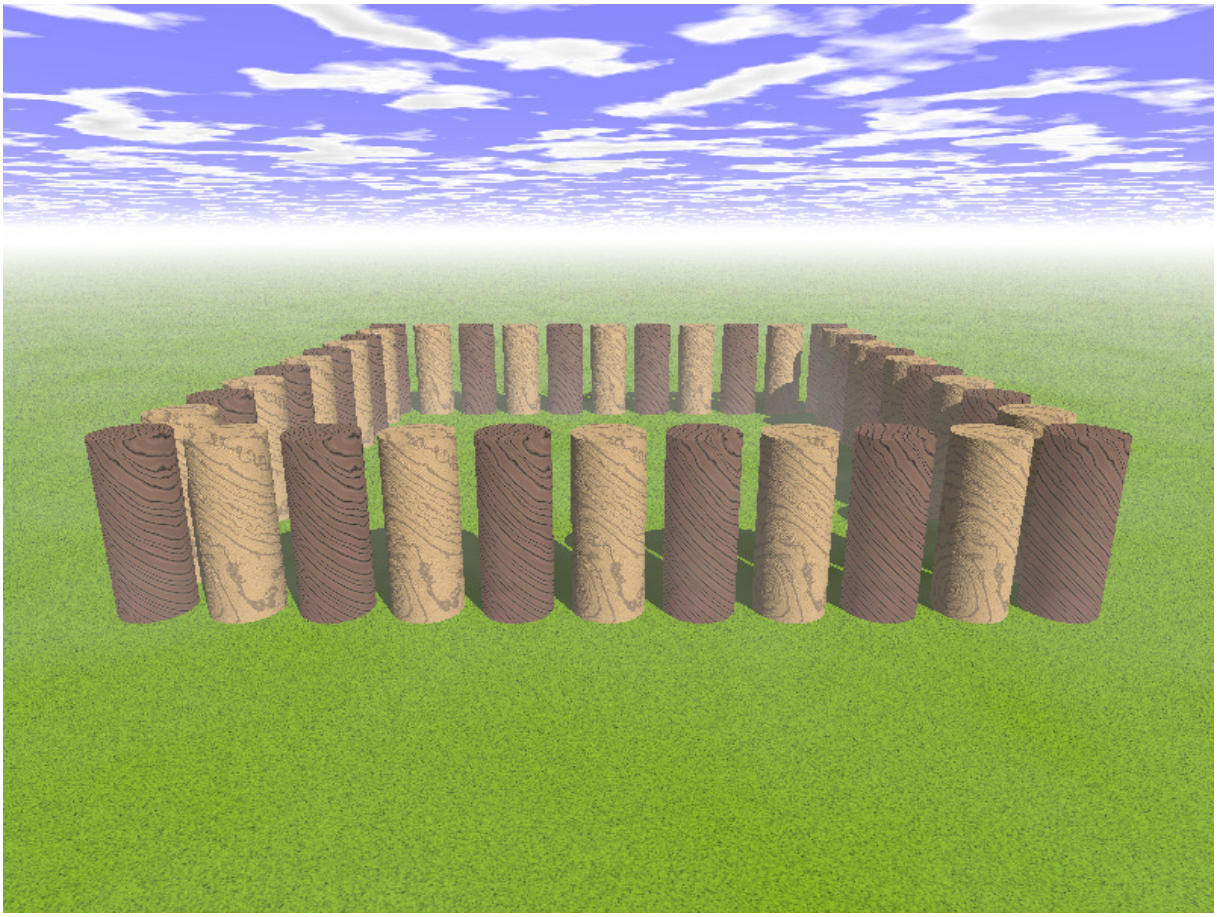


```
#declare Camera_0 = camera {/*ultra_wide_angle*/ angle 75      // front view
                    location  <0.0 , 4.0 ,-8.0>
                    right     x*image_width/image_height
                    look_at   <0.0 , 0.0 , 6.0>}
```

Zusatzaufgabe:

Kompletter Zaun:

```
#declare Nr = -5;     // start
#declare EndNr = 5;   // end
#while (Nr< EndNr+1)

    #if(  Nr/2 = int(Nr/2) )
      // only with even numbers Nr
      object{Pfahl_Holzton_1 translate<Nr,0,0>}
      object{Pfahl_Holzton_1 translate<5,0,Nr+5>}
      object{Pfahl_Holzton_1 translate<-5,0,Nr+5>}
      object{Pfahl_Holzton_1 translate<Nr,0,10>}

    #else // with odd numbers Nr
      object{Pfahl_Holzton_2 translate<Nr,0,0>}
      object{Pfahl_Holzton_2 translate<5,0,Nr+5>}
      object{Pfahl_Holzton_2 translate<-5,0,Nr+5>}
      object{Pfahl_Holzton_2 translate<Nr,0,10>}

    #end // end of "#if( Nr/2 = int (Nr/2))"

#declare Nr = Nr + 1;  // next Nr

#end // ---------- end of loop ----------------------------------
```



Erfahrungen aus der Praxis:

18.01.10 (7 Schüler)

Die Schüler verstanden sofort, was die „*wenn-dann*-Sätze“, die sie bilden sollten, gemeinsam hatten. Es war auch kein Problem zwischen ein- und zweiseitigen Anweisungen zu unterscheiden und zu erklären, wie sie funktionieren.

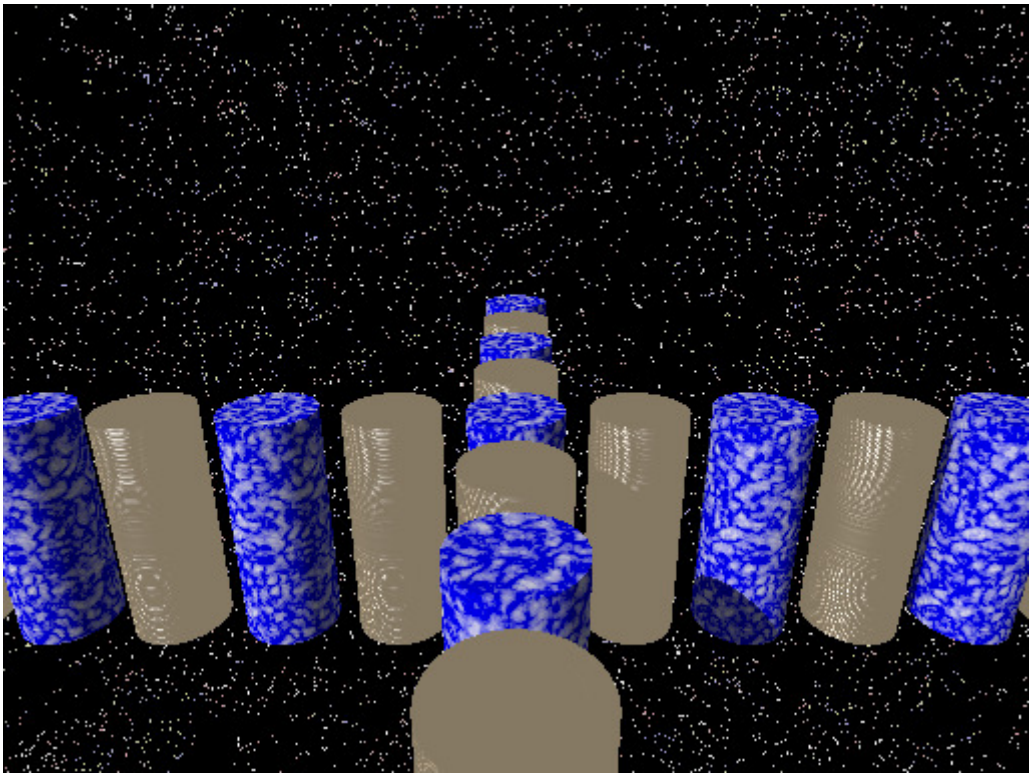
Auf die Frage, wie man denn in der Anweisung weitere Bedingungen abfragen könne, kamen die Schüler sofort darauf, dass es vielleicht möglich wäre diese Anweisungen zu verschachteln. Ich erwähnte, dass man als Bedingung beispielsweise nehmen könnte, „wenn Zahl gerade“, worauf direkt gefragt wurde, wie man dies denn dem Programm sagen könnte.
Ein Schüler kam sogar auf die Idee zu testen, ob die Zahl durch 2 teilbar ist. So kamen wir auf Integer und dann auch recht schnell auf die mögliche Eingabe in *POV-Ray* zu sprechen.

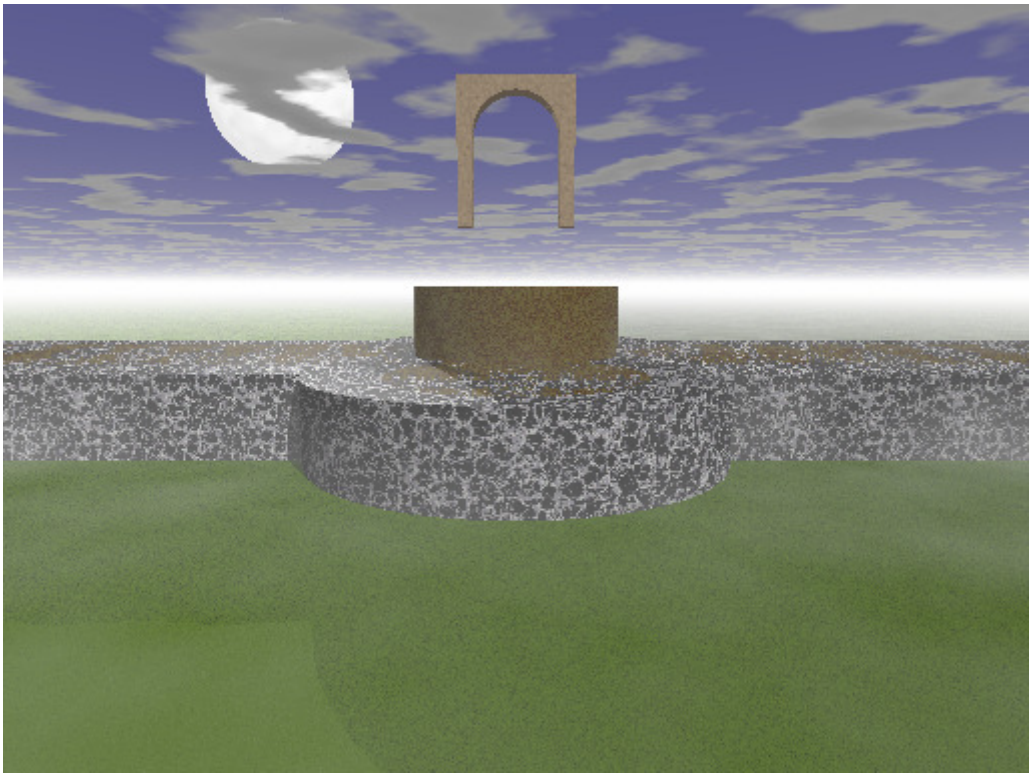
Das Bearbeiten der Programmieraufgabe lief recht gut. Ich stellte es den Schülern frei, sich einen beliebigen Hintergrund auszuwählen, da dies letzte Stunde sehr gut geklappt hatte und es den Schülern viel Spaß machte.
Wir überlegten, dass unser Zaun aus mehreren Pfählen bestehen könnte, die alle die Form eines Zylinders haben. Die Kameraeinstellungen zu ändern stellte auch kein Problem mehr da, nachdem wir dies letzte Stunde mitbehandelt hatten.
Auch stellte ich den Schülern frei, sich für die Pfähle ein beliebiges Material auszusuchen, um die Aufgabe etwas freier zu gestalten. Den meisten Schülern bereitete es keine Probleme die „texture“ an der richtigen Stelle im Code einzusetzen.
Viele Schüler erstellten über die zwei benötigten Klassen hinaus noch weitere Pfahl-Klassen, mit Pfählen von jeweils unterschiedlichem Material.

An die *while*-Schleife aus letzter Stunde konnten sich nicht mehr alle so gut erinnern, aber mit es was Mühe haben wir auch diese gemeinsam hinbekommen.
Die *if-else* Anweisung stellte kein Problem dar.

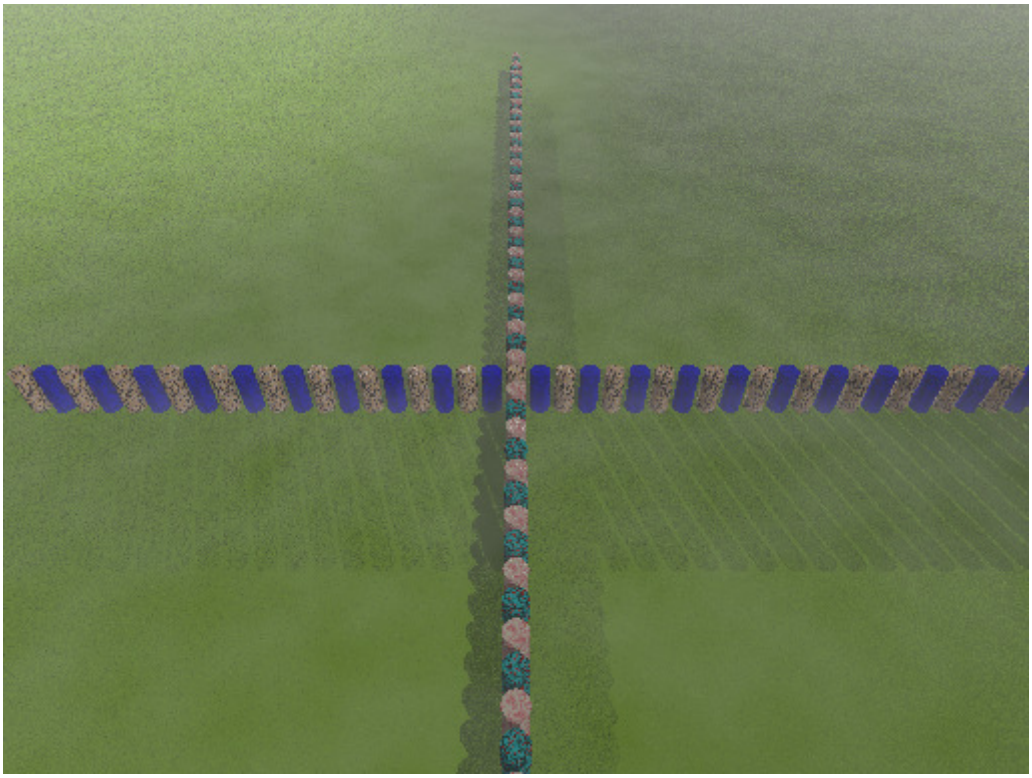
Den kompletten Zaun (mit vier Seiten) schafften wir in der Stunde nicht. Jedoch erstellten die Schüler, teilweise schon alleine, eine Zaunreihe orthogonal zur schon vorhandenen.
Und weil es ihnen so Spaß machte, beließ ich es dabei. So konnte sie noch beobachten was geschieht, wenn sie den ein oder anderen Wert in ihrem Programm veränderten.

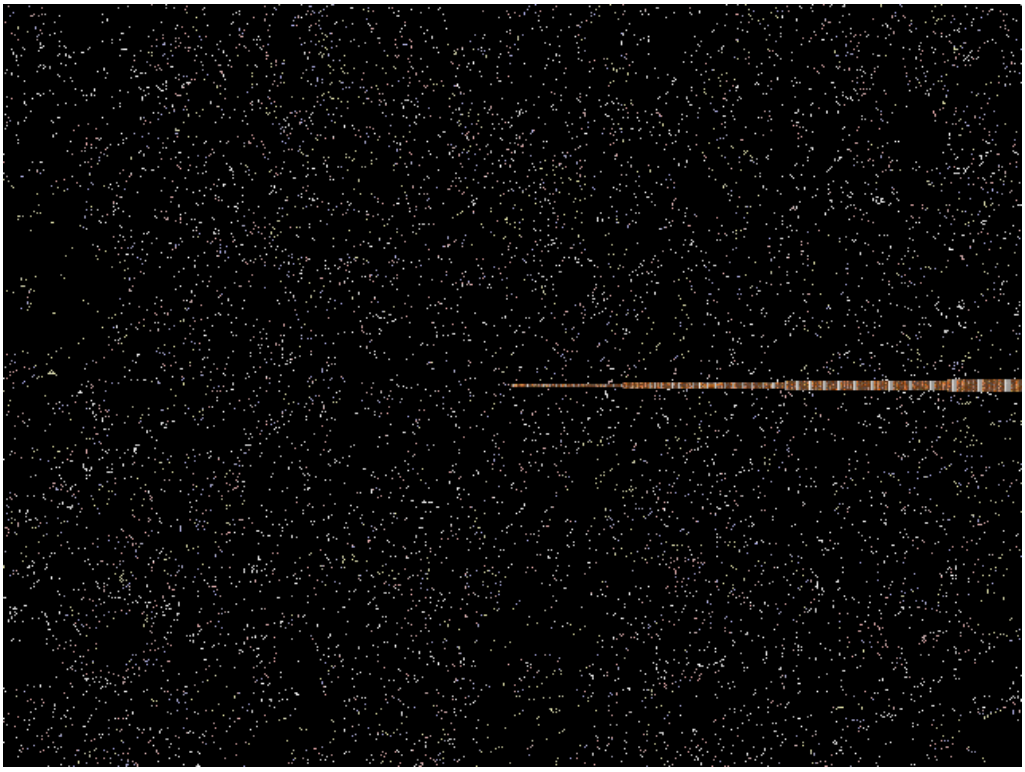
Schülerergebnisse





Der Schüler machte die Pfähle absichtlich so dick, dass sie zu einer Mauer zusammenwuchsen. „Das ist ja sicherer als ein Zaun.“





Hier wurde die Kameraeinstellung absichtlich so gewählt.

3.4 Einheit IV (*switch-case* Anweisungen)

Lernziele/-inhalte dieser Einheit:

Bedingte Anweisungen: *switch-case Anweisungen*
Weitere Begriffe, die eingeführt werden: Verzweigung, mehrfache Verzweigungen, Syntax, Kontrollstruktur, Endlosschleife

Ablauf:

Teil 1: Besprechung ohne Verwendung von *POV-Ray*:

Verzweigungen

Anhand von zwei Beispielen werden Situationen durchgespielt, die je nach Ausgangsituation andere Folgen nach sich ziehen sollen.

Beispiel 1 (Mensa):

Die Schüler kommen nach der Schule zum Mittagessen in die Mensa. Sie haben die Wahl zwischen drei verschiedenen Gerichten.

Gericht:
Spaghetti Bolognese
Kartoffelsuppe mit Brot
Frühlingsrolle mit Reis

Nun müssen sich die Schüler entscheiden, welches Essen sie nehmen wollen, und sich dann an der passenden Schlange anstellen.

Wähle (Gericht):

Fall (Spaghetti Bolognese)
    -> Gehe zur Ausgabe 1.
Fall (Kartoffelsuppe mit Brot)
    -> Gehe zur Ausgabe 2.
Fall (Frühlingsrolle mit Reis)
    -> Gehe zur Ausgabe 3.

Je nach Entscheidung muss der Schüler also anders handeln.

Für diejenigen, die keinen Hunger haben, muss es allerdings auch noch eine Verzweigung geben:

sonst
    -> Gehe nach Hause.

Beispiel 1 (Zeugnis):

Dieses Beispiel trägt dazu bei, dass die Schüler die Verzweigungen ähnlich der Syntax einer Programmiersprache formulieren.

Der Lehrer erstellt die Jahresabschlusszeugnisse. Dazu muss er für jedes Fach eine Note in den Computer eintragen. Trägt er eine 1 ein, so soll im Zeugnis „sehr gut“ stehen. Trägt er eine 2 ein, so soll das Programm „gut“ ausgeben, usw.

Wähle (Note)
Fall (Note=1): Ausgabe = „sehr gut“
Fall (Note=2): Ausgabe = „gut“
Fall (Note=3): Ausgabe = „befriedigend“
Fall (Note=4): Ausgabe = „ausreichend“
Fall (Note=5): Ausgabe = „mangelhaft“
Fall (Note=6): Ausgabe = „ungenügend“

Verzweigungen haben wir schon letztes Mal kennen gelernt (*if-else*)

```
wenn Bedingung dann
        Anweisung 1
sonst
        Anweisung 2
```

Die Verzweigung besteht also aus einer Bedingung und zwei Anweisungen. Es wird erst die Bedingung ausgewertet, und wenn sie zutrifft, wird anschließend die erste Anweisung ausgeführt, anderenfalls wird die zweite Anweisung ausgeführt.

Teil 2: Übergang zu *POV-Ray*:

Mehrfache Verzweigungen

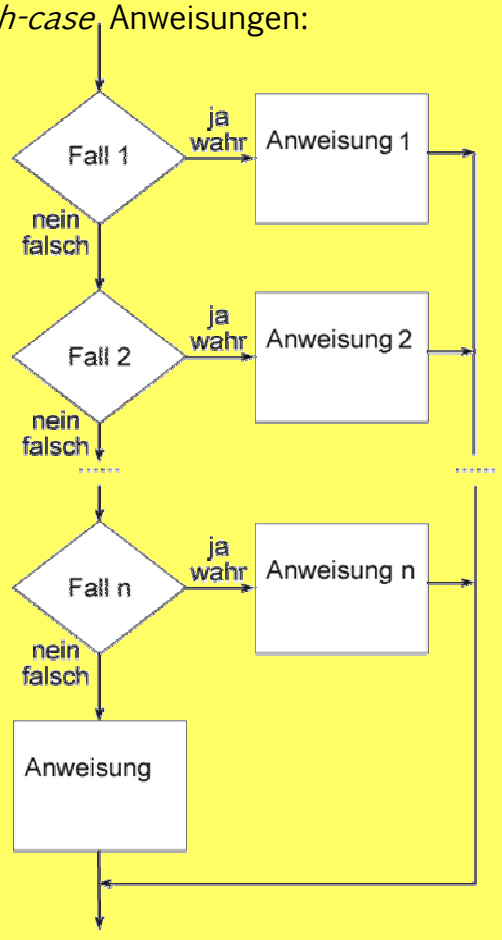
Oft braucht man, wie oben, auch mehrfache Verzweigungen. Diese nennt man auch Fallunterscheidungen genannt. Um solche in einem Computerprogramm zu verwenden schreibt man *switch-case* Anweisungen.
Es können so viele Fälle angegeben werden wie gewünscht, allerdings keiner doppelt. *switch/case* kann man eigentlich auch mit *if/else* darstellen, allerdings ist es nicht vorteilhaft, da es erstens unleserlicher ist und zweitens die Ausführung länger dauert.

Syntax

Wenn man tatsächlich ein Programm schreiben möchte, das wie oben Zeugnisse erstellt, so muss man die Syntax beachten.
Unter der Syntax bei Programmiersprachen versteht man ein System von Regeln, das vom Programm verstanden wird.

Für *POV-Ray* lautet die Syntax für *switch-case* Anweisungen:

```
#switch (Auswahlvariable)

#case (Wert 1 der Variable)
 ... Anweisung 1
#break

#case (Wert 2 der Variable)
 ... Anweisung 2
#break

....

#case (Wert n der Variable)
 ... Anweisung n
#break

#else
 ... Anweisung
#break

#end
```



Wichtig ist hier auch das „break“ (=brich ab). Man muss dem Programm mitteilen, dass es, wenn ein Fall zutrifft, nicht mehr die anderen Fälle durchlaufen muss.

Die Schüler werden nun dazu angehalten, das Zeugnis Beispiel in *POV-Ray* Syntax zu formulieren.

Programm zur Zeugniserstellung:

```
#switch (Note)

 #case (1)
   Ausgabe = „sehr gut“
 #break
 #case (2)
   Ausgabe = „gut“
 #break
 #case (3)
   Ausgabe = „befriedigend“
 #break
 #case (4)
   Ausgabe = „ausreichend“
 #break
 #case (5)
   Ausgabe = „mangelhaft“
 #break
 #case (6)
   Ausgabe = „ungenügend“
 #break
 #else
   Ausgabe = „EINGABEFEHLER“
 #break

#end
```

Kontrollstrukturen

Kontrollstrukturen werden in Programmiersprachen verwendet, um den Ablauf eines Computerprogramms zu steuern. Eine Kontrollstruktur gehört entweder zur Gruppe der Verzweigungen oder der Schleifen.

Die Schüler haben nun die wichtigsten Kontrollstrukturen kennen gelernt.
Sie werden nun aufgefordert diese zu wiederholen und entweder den Schleifen oder den Verzweigungen zuzuordnen.
(Zählschleifen, while-Schleifen, if-else Anweisungen (einseitig/zweiseitig), switch-case Anweisungen (mit/ohne default-Anweisung))

Endlosschleifen

Es ist wichtig den Schüler noch mitzugeben, dass sie aufpassen müssen, dass ihr Programm nicht in eine Endlosschleife gerät.

Wie entstehen Endlosschleifen?
Endlosschleifen können bei der Programmierung durch Fehler entstehen, wenn die Abbruchbedingung nicht definiert ist oder nicht eintreten kann.

Beispiel:

```
#declare Nr = -5;
#declare EndNr = 5;
#while (Nr< 10)
   .....
#end
```

Da „Nr“ immer kleiner ist als 10, wird die *while*-Schleife unendlich lange durchlaufen, da die Bedingung (Nr < 10) immer wahr ist.

Teil 3: Mit *POV-Ray*:

Die Schüler sollen *POV-Ray* starten und den Anweisungen auf dem Arbeitsblatt folgen. Sie sollen dabei *switch-case* Anweisung mit recht vielen Verzweigungen schreiben. Diese soll in eine *while*-Schleife eingebettet werden.

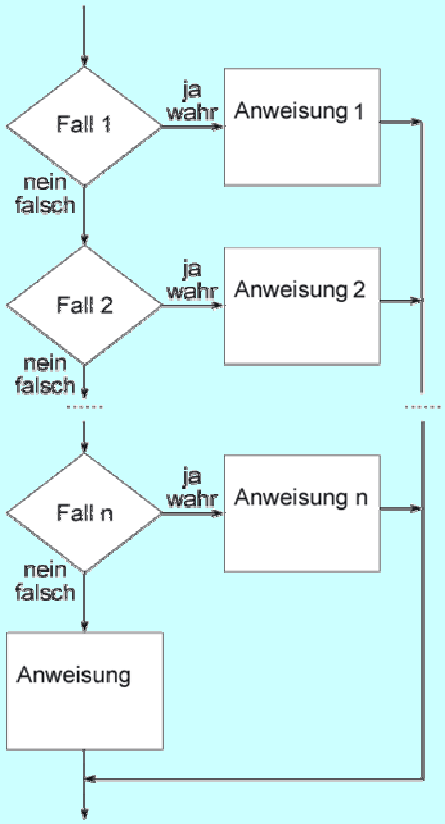
Arbeitsblatt IV – Switch-case Anweisungen

Merke:

Kontrollstrukturen werden in Programmiersprachen verwendet, um den Ablauf eines Computerprogramms zu steuern. Eine Kontrollstruktur gehört entweder zur Gruppe der Verzweigungen oder der Schleifen.

Für *POV-Ray* lautet die Syntax für *switch-case* Anweisungen:

```
#switch (Auswahlvariable)
#case (Wert 1 der Variable)
 ... Anweisung 1
#break
#case (Wert 2 der Variable)
 ... Anweisung 2
#break
....
#case (Wert n der Variable)
 ... Anweisung n
#break
#else
 ... Anweisung
#break
#end
```



Es können so viele Fälle angegeben werden wie gewünscht, allerdings keiner doppelt. *switch/case* kann man eigentlich auch mit *if/else* darstellen, allerdings ist es nicht vorteilhaft, da es erstens unleserlicher ist und zweitens die Ausführung länger dauert.

Achtung! Hüte dich vor **Endlosschleifen**.

Beispiel:

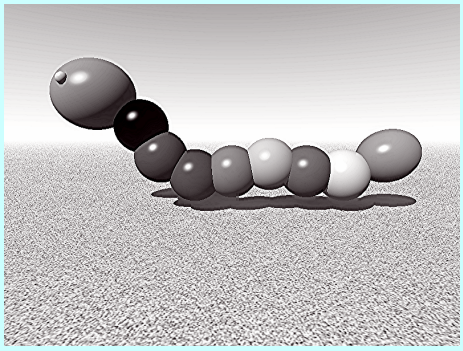
```
#declare Nr = -5;
#declare EndNr = 5;
#while (Nr< 10)
   .....
#end
```

Da „Nr“ immer kleiner ist als 10, wird die *while*-Schleife unendlich lange durchlaufen, da die Bedingung (Nr < 10) immer wahr ist.

Arbeitsanweisungen:

Nun programmiert ihr selbst einen Wurm mit Hilfe der *switch-case* Anweisung. Programmiert dazu eine Schleife die z.B. 10 Kugeln erstellt. Verwendet *switch/case* um die Kugel hintereinander zu platzieren, ihre Farbe und ihre Form zu verändern:

1. Lege ein neues Dokument an.
2. Speichere es in deinem *POV-Ray* Ordner unter dem Namen: „Wurm.pov“
3. Füge folgende Szene ein: Insert -> Basic templates -> Ready-made scenes -> Basic Scene 03
4. Schreibe die oben beschriebene Schleife.
5. Füge in die Schleife eine beliebige Kugel ein (Insert -> Basic templates -> Shapes 0 with textures -> sphere with texture)
6. Entscheide mit einer *switch-case* Anweisung, welche Textur die einzelnen Kugel haben sollen, die mit Hilfe der Schleife erzeugt werden. (Mit finish{phong 1} bekommt die Kugel einen schönen Glanz.)
7. Programmniere die *switch-case* Anweisung **in** den Code der Kugel ein.
8. Zur Orientierung: Eine der mittleren Kugeln des Wurms solle bei <0,1,0> liegen.
9. Kannst du dem Wurm noch Augen geben?



Zusatz: Fasse alle Teile/Kugeln des Wurms mit Hilfe von union{} zusammen. Verschiebe dann den kompletten Wurm mit translate und rotate oder ändere seine Größe mit scale.

Bei Fragen an den Lehrer wenden.

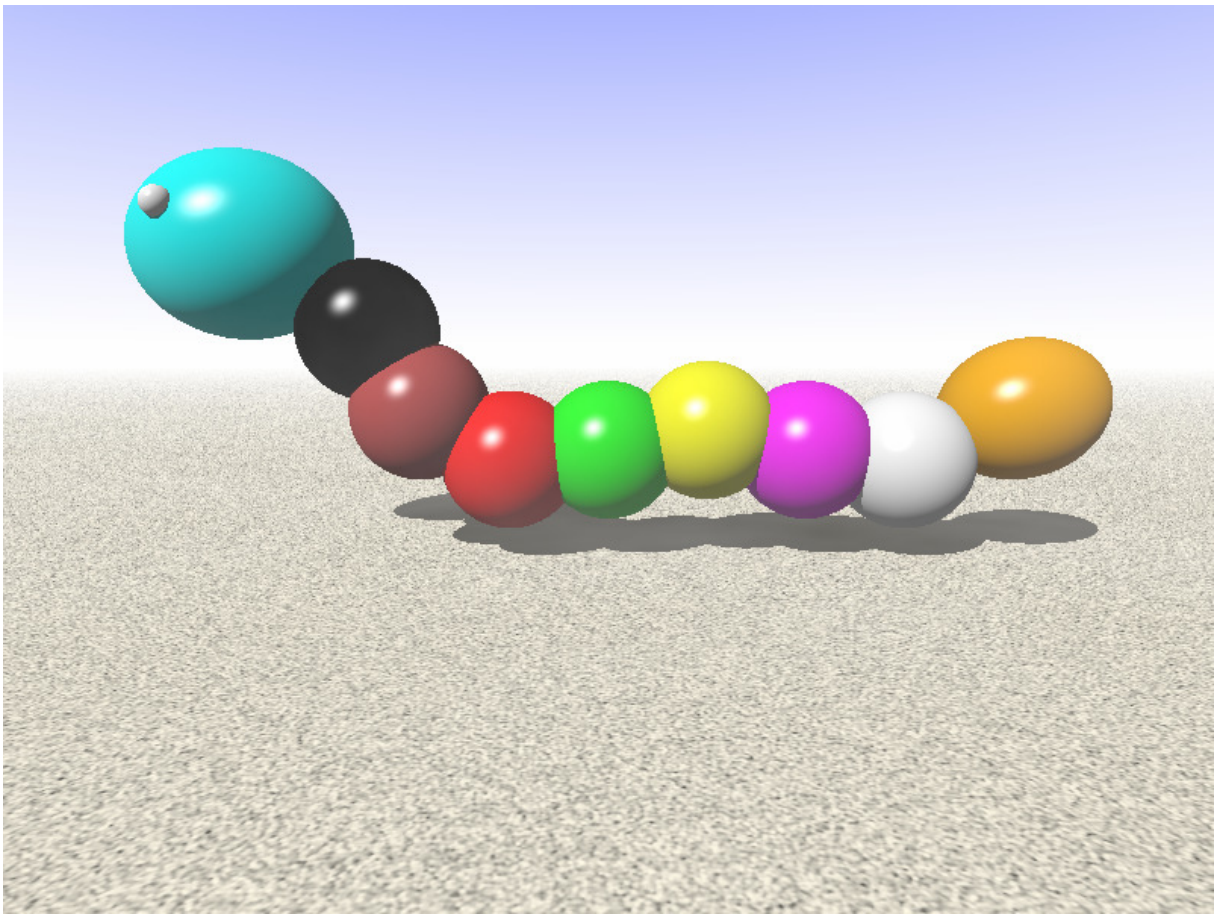
Lösungen für den Lehrer

Aufgabe und Zusatzaufgabe:

```
union{
 #local Nr = 0;     // start
 #local EndNr = 11; // end
 #while (Nr< EndNr)
  sphere{ <0,0,0>,0.7

   #switch (Nr)

   #case (0)
      texture{ pigment{ color Cyan}
               finish { phong 1}
              }
      translate<-2.4,2.4,0>
      scale<1.5,1.3,1.3>
   #break

   #case (1)
      texture{ pigment{ color Black}
               finish { phong 1}
              }
      translate<-2.4,2.3,0>
      scale<1,1,1>
    #break

   #case (2)
      texture{ pigment{ color Scarlet}
               finish { phong 1}
              }
      translate<-1.9,1.5,0>
      scale<1,1,1>
   #break

   #case (3)
      texture{ pigment{ color Red}
               finish { phong 1}
              }
      translate<-1,1,0>
      scale<1,1,1>
   #break

   #case (4)
      texture{ pigment{ color Green}
               finish { phong 1}
```

```
        }
  translate<0,1.1,0>
  scale<1,1,1>
#break

#case (5)
  texture{ pigment{ color Yellow}
         finish { phong 1}
        }
  translate<1,1.3,0>
  scale<1,1,1>

#break

#case (6)
  texture{ pigment{ color Magenta}
         finish { phong 1}
        }
  translate<2,1.1,0>
  scale<1,1,1>
#break

#case (7)
  texture{ pigment{ color White}
         finish { phong 1}
        }
  translate<3,1,0>
  scale<1,1,1>
#break

#case (8)
  texture{ pigment{ color Orange}
         finish { phong 1}
        }
  translate<3.7,0,0>
  scale<1.2,1,1>
  rotate<0,0,20>
#break

#case (9)   //Auge
  texture{ pigment{ color White}
         finish { phong 1}
        }
  translate<-20,17,-4>
  scale<0.2,0.2,0.2>

#break
```

```
    #case (10)   //Auge
      texture{ pigment{ color White}
               finish { phong 1}
              }
      translate<-20,17,4>
      scale<0.2,0.2,0.2>

    #break


    #end // end of #switch

    }// --- end of sphere ----


 #local Nr = Nr + 1;    // next Nr
 #end // ---------------  end of loop

rotate<0,0,0>
translate<0,0,0>
} // end of union --------------------
```



Erfahrungen aus der Praxis:

25.01.10 (7 Schüler)

Da sich die Schüler schon letzte Stunde dafür interessiert hatten, wie man denn bei *if-else* Anweisungen noch mehr Anweisungen mit einbringen könnte, so war ihnen das Thema dieser Stunde sehr willkommen.
Leider waren die Schüler heute sehr aufgedreht und laut. Ob es am Neuschnee lag oder am Feueralarm, der zuvor stattgefunden hatte, kann ich nicht beurteilen.

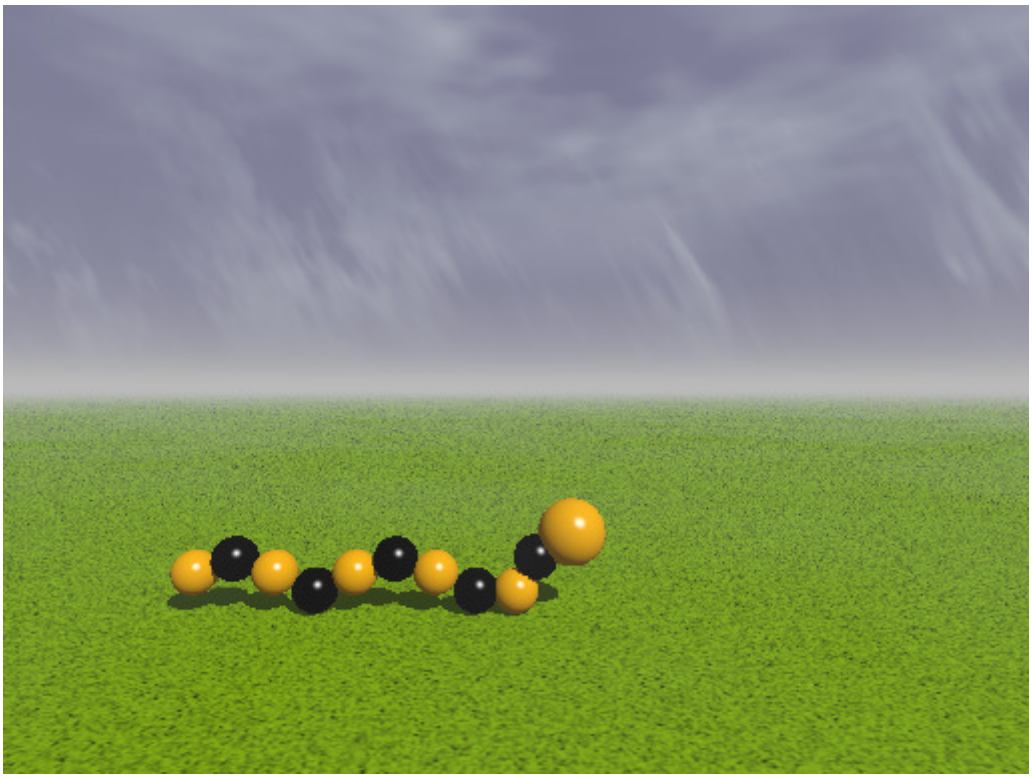
Die Theorie arbeiteten wir, wenn auch mit etwas mehr Unruhe als sonst, problemlos durch. Auch auf meine Frage, was wir denn bisher alles gelernt hätten (Kontrollstrukturen), kamen alle erwünschten Antworten.

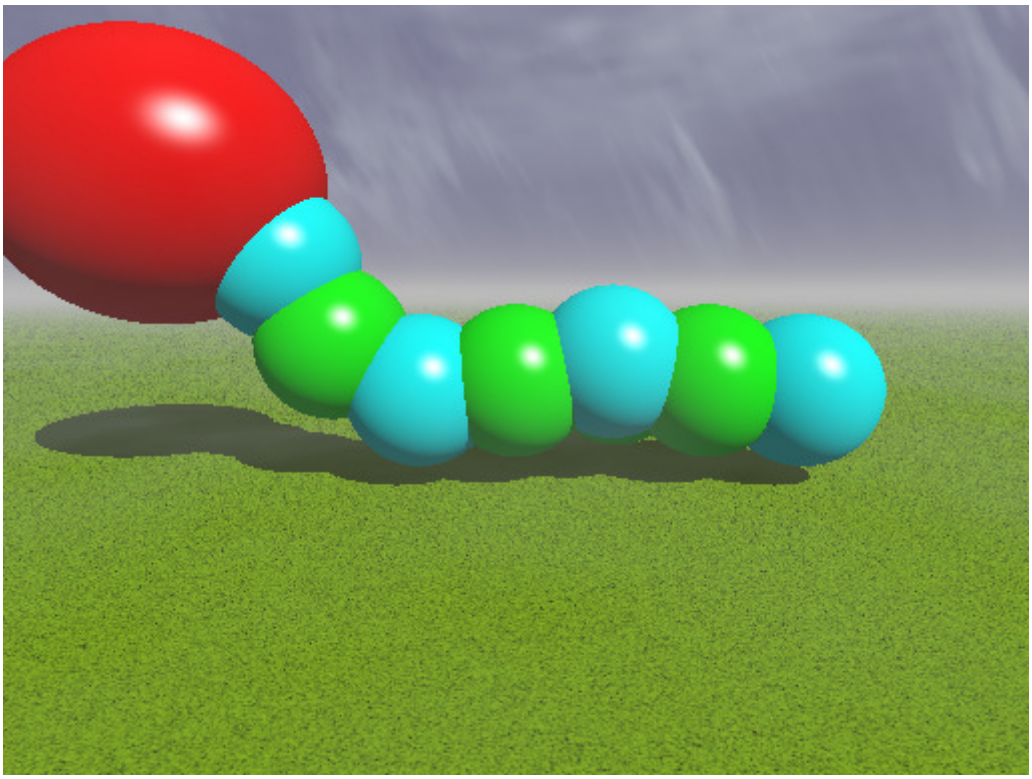
Bei der Arbeit mit *POV-Ray* stellte ich es den Schülern wieder frei sich einen beliebigen Hintergrund auszuwählen.
Die benötigte *while*-Schleife stellte kein allzu großes Problem mehr da, die *switch-case* Anweisung genauso wenig.

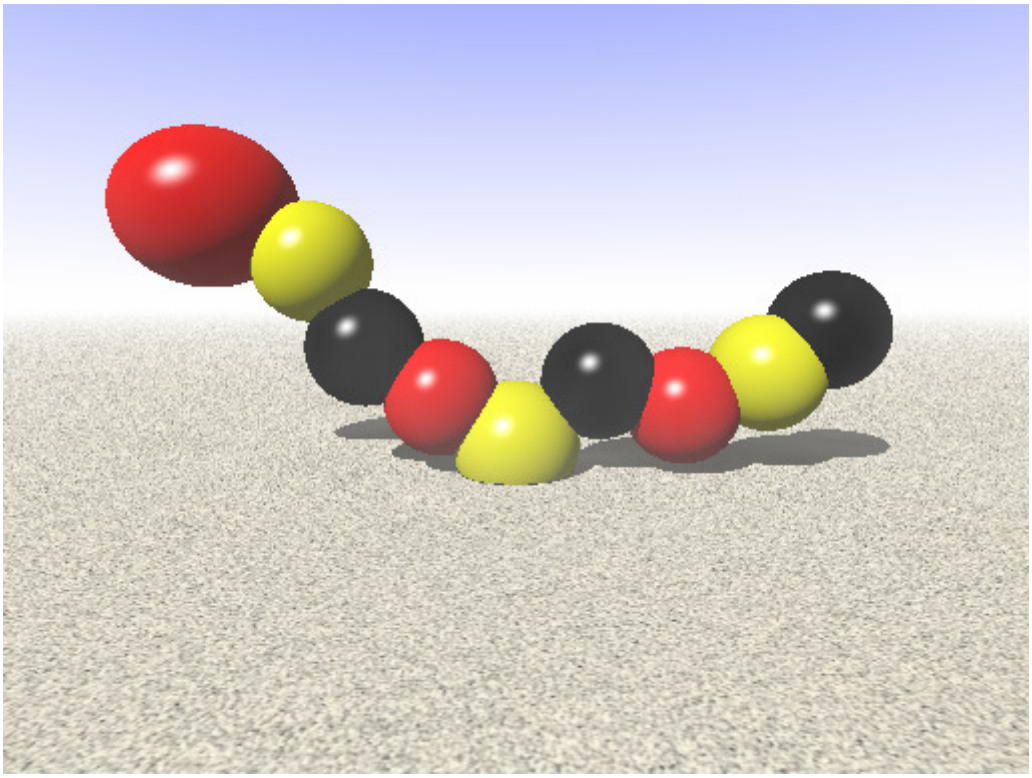
Nachdem alle die erste Kugel erstellt hatten, arbeiteten die Schüler komplett selbstständig. Einige hatten etwas Probleme mit der Koordinatenberechnung der nächsten Kugeln, andere jedoch schafften dies ohne Probleme.

Alles in Allem würde ich beurteilen, dass den Schüler diese Stunde leichter gefallen ist als die letzten.

Schülerergebnisse







3.5 Einheit V (Animation)

Lernziele/-inhalte dieser Einheit:

Wiederholung und Anwendung aller gelernten Strukturen in *POV-Ray*
Objektorientierung: Klassen definieren, Objekten Attribute zuweisen, Methoden anwenden
Bedingte Anweisungen: *while-Schleife*, *if-else* Anweisungen, *switch-case Anweisungen*
Allgemein: Wie entstehen Filme?, Umgang mit ini-Dateien, Bilder zu einem Film zusammenfügen
Weitere Begriffe, die eingeführt werden: AVI-Dateien, frames, fps

Ablauf:

Teil 1: Besprechung ohne Verwendung von *POV-Ray*:

Motivation

In dieser Unterrichtseinheit sollen die Schüler das in den vorherigen Einheiten Gelernte anwenden. Sie soll zu Festigung des Wissens dienen und die Schüler dazu bringen das Vergessene nochmals nachzuschlagen.
Außerdem soll sie die Schüler motivieren auch nach dem Themenblock „*POV-Ray*“ sich weiter mit dem Programm zu beschäftigen.
Des Weiteren soll den Schülern etwas klarer werden, wofür man die vorher gelernten Programmierkonzepte braucht, und dass sie durchaus für interessante Anwendungen wichtig sind.

Frames

frame = das Einzelbild eines Filmes, Videos, einer Animation
Jeder Film ist eine schnelle Abfolge von Einzelbildern ("frames"), die sich nur wenig verändern. Durch die Trägheit unserer Lichtwahrnehmung im Auge wird diese Bildfolge von unserem Gehirn als kontinuierliche Bewegung interpretiert.

Frames per second

fps = "frames per second" = die Frame-Rate = Anzahl der Einzelbilder pro Sekunde
Das menschliche Auge nimmt eine sich langsam ändernde Folge von Einzelbildern dann als kontinuierliche Bewegung wahr, wenn es ca. 25 Bilder pro Sekunde sieht. Eine langsamere Bildfolge führt zu einem unangenehmen Ruckeln bzw. Flackern.
Bei Bildrate von 25.0 fps, beträgt de Bilddauer 40.0 ms.

Teil 2: Übergang zu *POV-Ray*:

Technische Details

Achtung!
Unter Windows Vista kann es zu Problemen beim starten von Animationen kommen. Diese kann man beheben, indem man seine Version auf die Version „*POV-Ray* 3.6 2“ updatet. Ist die Installation von der beiliegenden CD gestartet worden, so sind Sie bereits im Besitz der neusten Version.

Für eine Animation mit *POV-Ray* benötigen wir nur 2 verschiedenartige Dateien:
1. die *POV-Ray*-Szenendatei (Endung: .pov), welche den "clock"-Wert benutzt und
2. die *POV-Ray*-Amimations-Inizialisierungsdatei (Endung: .ini), welche den "clock"-Wert definiert und mit jeweils verändertem "clock"-Wert die Szenendatei wiederholt aufruft.

Teil 3: Mit *POV-Ray*:

Beispiel Objektbewegung

Vorführung: rotierende Kugel (Lehrer)

(Die Dateien Kugel.pov und Kugel.ini sollten im selben Verzeichnis liegen.)
Animation starten, Bilder erzeugen, Bilder in eine avi-Datei packen.

Erklärung der Dateien:

Kugel.pov

Die Kugel wird durch den Befehl **translate <1.0, 0, 0>** zur Seite verschoben. Dann wird sie in dieser Animation um die y-Achse durch **rotate <0, 360*clock, 0>** gedreht. Der **"clock"-Wert** wächst während der Berechnung der Einzelbilder der Animation von 0 auf 1 an. (Seine Grundeinstellung ist "**clock = 0**".) Das Hochzählen wird durch die Animations-ini-Datei erledigt.

Ausschnitt aus Kugel.pov:

```
// Die rotierende Kugel:
sphere{ <0,0,0>, 0.25
  texture { pigment{ rgb<1,0,0>}
            finish { diffuse 0.9
                     phong 1}
          } // end of texture
  translate< 1.0, 0, 0>
  rotate < 0, 360*clock, 0>
       } // end of sphere ----------
```

Kugel.ini

Die Zeilen "**Initial_Frame=1**" und "**Final_Frame=30**" legen fest, dass 30 Bilder gerechnet werden sollen.
Die Zeilen "**Initial_Clock=0**" "**Final_Clock=1**" geben den Start- und Endwert für die clock-Variable an. Er sollte möglichst immer nur von 0 bis 1 gezählt werden.
"**Cyclic_Animation=on**" sorgt dafür dass bei einer zyklischen Animation das letzte Bild (= das erste Bild) nicht mehr berechnet wird.

```
; POV-Ray animation ini file

Input_File_Name="Kugel.pov"

Initial_Frame=1
Final_Frame=30
Initial_Clock=0
Final_Clock=1

Cyclic_Animation=on
```

In der INI-Datei:

Anzahl der Bilder, die berechnet werden:
"**Initial_Frame=1**"
"**Final_Frame=30**"

Start und Endwert für die clock-Variable:
"**Initial_Clock=0**"
"**Final_Clock=1** (möglichst immer nur von 0 bis 1)

In der POV-Datei:

Ein Objekt um die y-Achse drehen: **rotate <0, 360*clock, 0>**
Der **"clock"-Wert** wächst während der Berechnung der Einzelbilder der Animation von 0 auf 1 an.

Starten der Berechnung der Bilder der Animation:

Für Testzwecke wäre es gut eine kleine Bildauflösung einzustellen (z.B. "160 x 120")
Gestartet wird die Animation indem man die ini-Datei startet (nicht die pov-Datei).
POV-Ray beginnt die Berechnung der Szenendatei "sphere1.pov" mit dem "clock"-Wert von 0 und speichert das berechnete Bild als frame Nr.1 mit dem Dateinamen "sphere101.bmp" ab.
Dann berechnet das Programm den nächsten "clock"-Wert (neuer clock-Wert = letzter clock-Wert + 1/Final_Frame, hier: 0 + 1/30) und startet die Berechnung unserer Szene "sphere1.pov" mit diesem Wert erneut. Das fertige Teilbild wird als "sphere102.bmp" gespeichert.
Dies geht so weiter bis "clock" = 1 erreicht ist mit der Bildbezeichnung "sphere130.bmp".

Zusammenfügen zu einer Animation:

POV-Ray kann eine gerenderte Animation nur als einfache durchnummerierte Einzelbilder ("frames") abspeichern.
Wenn wir unsere Animationen als animierte Gif-Datei, avi-, mov- oder mpeg-Datei speichern wollen, müssen wir dazu ein anderes Programm verwenden.

AVI-Videoformat

Dieses Videoformat kann mit dem "Windows Mediaplayer" abgespielt werden, sofern der bei der Komprimierung verwendete Videocodec auf dem Computer installiert wurde.
Es erlaubt eine Darstellung in true-color. Die Komprimierung ist wesentlich besser als mit animierten Gif-Dateien bei wesentlich höherer Bildqualität.

Ich empfehle das kostenlose Programm „Bmp_zu_Avi“. Dieses ist ein sehr einfaches Programm, das die Schüler nicht mit zu vielen Funktionen möglicherweise verwirrt. Die Version „Bmp_zu_Avi_6_1“ befindet sich auf der beigelegten CD.
Für eine sich schnell drehende Kugel empfiehlt sich eine Abspielzeit von 100ms pro Bild. Unter „Vorschau“ kann man auch die Option „Wiederholen“ auswählen, sodass die Drehung der Kugel immer wieder von vorne beginnt.

Für den Unterricht muss das Programm je nach Möglichkeiten im Computerraum der Schule von den Schülern selbst installiert werden. Dies sollte aber kein Problem darstellen. Nach starten der Setup-Datei wird einfach nur den Anweisungen der Installation gefolgt.

Beispiel Kamerabewegung

Vorführung Kirche (Lehrer)

Nicht nur Objekte sondern auch der Kamerastandpunkt sowie der Blickpunkt und Blickwinkel lassen sich animieren.

Dabei eignen sich *if-else* Anweisungen sehr gut, um die Kamera von unten nach oben zu bewegen.

Der Lehrer startet die Kirche.ini Datei und erklärt wie und wo die Höhe der Kamera festgelegt wird.

```
// camera --------------------------------------
#declare Jump_Start  = 0.5;
#if (clock < Jump_Start )
 #declare Camera_Y = 1.00+(clock*10);
#else
 #declare Camera_Y = 11.00-(clock*10);
#end

camera {
 angle 38
 location <3,Camera_Y,-20>
 right x*image_width/image_height
 look_at <-3,3,5>
 rotate<0,-360*(clock+0.1),0>
} //------------------- end of camera
```

Arbeitsblatt V – Animation

Merke:

Grundsätzliche Animationsbefehle:

In der ini-Datei:

Input_File_Name="Schneemann.pov" (die Datei , die die Bilder liefert.)

Die Zeilen "**Initial_Frame=1**" und "**Final_Frame=30**" legen fest, dass 30 Bilder gerechnet werden sollen.
Die Zeilen "**Initial_Clock=0**" "**Final_Clock=1**" geben den Start- und Endwert für die clock-Variable an. Er sollte möglichst immer nur von 0 bis 1 gezählt werden.

Cyclic_Animation=On (Dies ist sehr nützlich bei zyklischen Animationen wie z.B. dem Drehen eines Objekts um 360 Grad: Es bewirkt, dass der clock-Unterschied 1 bei einer Frame-Anzahl von 10 in z.B. 10 Abschritte unterteilt wird.)

In der pov-Datei:

Zum Beispiel:
rotate < 0, 360*clock, 0>
(ein Objekt um die y-Achse drehen)

In der pov-Datei: Kamerabewegung (drehen und Höhe ändern):

```
// camera -------------------------------------
#declare Jump_Start  = 0.5;
#if (clock < Jump_Start )
 #declare Camera_Y = 1.00+(clock*10);
#else
 #declare Camera_Y = 11.00-(clock*10);
#end

camera {
 angle 38
 location <3,Camera_Y,-20>
 right x*image_width/image_height
 look_at <-3,3,5>
 rotate<0,-360*(clock+0.1),0>
} //------------------- end of camera
```

Arbeitsanweisungen:

Nun programmiert ihr selbst ein paar Schneemänner, die ihr dann animiert. Danach lasst kommt zur Bewegung der Schneemänner noch eine Kamerabewegung dazu.

1. Öffnet das Dokument „Vorlage_Schneemann.pov“
2. Speichere es in deinem *POV-Ray* Ordner unter dem Namen: „Schneemann.pov“
3. Schaue dir die vorgegebene Klasse Schneemann an. Erzeuge einen Schneemann.
4. Erstelle das Dokument „Schneemann.ini“.
5. Lasse einen Schneemann um sich selbst rotieren. Hole dazu die ini-Datei in den Fordergrund und starte die Bildererzeugung mit „Run“.
6. Öffne das Programm Bmp_zu_Avi. Lade die erzeugten Bilder in das Programm. Die Bilder befinden sich im gleichen Ordner wie die Datei „Schneemann.pov“.
7. Wähle 100ms pro Bild aus und starte die Vorschau.
8. Kannst du den Schneemann auch im Kreis laufen lassen?
9. Erstelle weitere Schneemänner im Bild (while-Schleife) und erstelle eine Animation. (Bewege alle Schneemänner.)
10. Bewege zusätzlich zum animierten Schneemann noch die Kamera. (if-else Anweisung)
11. Ändere die Klasse die Schneemänner, sodass die Schneemänner auch Augen, Nase etc. haben.

Bei Fragen an den Lehrer wenden.

Lösungen für den Lehrer

Klasse Schneemann

```
#declare schneemann=union{
union{
//threshold 0.08
sphere { <0, 2, 0>,2
pigment{ White}

}
sphere { <0,3.6,0>, 1.3
pigment { White }
}
sphere { <0,5,0>, 0.9
pigment { White }
}
sphere { <0,5,-1>, 0.1
pigment { Orange }
}
sphere { <0.2,5.2,-0.9>, 0.1
pigment { Black }
}
sphere { <-0.2,5.2,-0.9>, 0.1
pigment { Black }
}
sphere { <1.2,3.6,0>, 0.5
pigment { White }
}
sphere { <-1.2,3.6,0>, 0.5
pigment { White }
}
sphere { <0,3.6,-1.4>, 0.1
pigment { Blue }
}
}
cylinder {<0,5.5,0>,<0,5.6,0>, 1
pigment { Blue }}
cylinder {<0,5.5,0>,<0,6.4,0>, 0.7
pigment { Blue }}
}
```

Schneemann, der um sich selbst rotiert (1), bzw. im Kreis läuft (2):

```
//(1)
object{schneemann
rotate<0, 360*clock, 0>}

//(2)
object{schneemann
translate<1,0,0>
rotate<0, 360*clock, 0>}
```

Erfahrungen aus der Praxis:

Leider konnte diese Stunde am 01.02.10 nicht wie geplant durchgeführt werden.
Als ich in der Schule ankam, wurde mir eröffnet, dass heute wegen eines Elternsprechtages kein Unterricht stattfindet.
Leider konnte diese auch in den folgenden Wochen nicht nachgeholt werden (zeitliche Gründe meinerseits und Schulferien).

Dies ist sehr schade, da sich die Schüler sehr auf die letzte Stunde gefreut hatten, sie wussten ja schon, was wir machen würden.
Außerdem konnte ich mir die restlichen Schülerergebnisse aus den letzten Stunden von den Schülern nicht mehr geben lassen.
Die geplante Abschlussbefragung (, ob es den Schüler Spaß gemacht hat, ob die Lernziele erreicht wurden,) konnte somit leider auch nicht durchgeführt werden.

4. Zusammenfassung

4.1 Abschlussbefragung

Fand leider nicht statt.

4.2 Bewertung des Praktikums aus eigener Sicht

Meiner Ansicht nach verlief das Praktikum erstaunlich gut. Mir war klar, dass in all den Unterrichtseinheiten sehr viel von den Schülern gefordert würde, und ich war positiv überrascht, wie gut die Schüler mit den neuen Inhalten und dem Programmieren zurechtkamen. Sie hatte vorher noch keine Programmiererfahrung gesammelt.

4.3 Ausblick

Für eine Informatik-AG sind *POV-Ray* und die hier behandelten Inhalte durchaus sehr gut verwendbar. Allerdings sollte man darauf achten, dass nicht zu viele Schüler an der AG teilnehmen, da es doch immer wieder nötig war den Schüler einzeln zu helfen.
Ich könnte mir des Weiteren gut vorstellen, dass *POV-Ray* und die obigen Unterrichtseinheiten auch in höheren Klassen viel Interesse hervorbringen könnten. Dann könnte man das Projekt auch mit mehr Schülern durchführen.

Da es in Baden-Württemberg zur Zeit noch keinen Informatikunterricht in Unter- und Mittelstufe gibt, fällt die Option weg, das Programm auch dort einzusetzen. Sollte sich das aber in Zukunft ändern, so wäre dies auch einen Gedanken wert. Allerdings muss der Lehrer hier erst einmal seine Schüler richtig einschätzen. Diese Einheiten mit 20 oder mehr Schülern durchzuführen ist sicherlich nicht einfach. Wenn die Klasse fit ist, kann man dies aber durchaus versuchen.

5. Literatur- und Quellenverzeichnis

Bildungspläne Gymnasium Klasse 6 und 7, Bayern, Staatsinstitut für Schulqualität und Bildungsforschung München

Bildungsplan 2004, Allgemein bildendes Gymnasium, Baden-Württemberg, Ministerium für Kultus, Jugend und Sport

Enter 1, Informationstechnische Grundbildung, Gymnasium; Schroedel-Verlag (3)

Ikarus, Natur und Technik, Schwerpunkt: Informatik; Odenburg-Verlag (4)

Natur und Technik, Schwerpunkt: Informatik, Bayern 6/7 Gymnasium; DUDEN PAETEC Schulbuchverlag (1)

Netzwerk Informatik 6/7, Bayern; Schroedel-Verlag (2)

http://www.f-lohmueller.de/pov_tut/pov__ger.htm (Beschreibungen und Beispiele zum Raytracer *POV-Ray* von Friedrich A. Lohmüller)

http://www.povray.org/ (offizielle Homepage von *POV-Ray*)

6. Anhang

Arbeitsblätter in Druckformat

Übersicht über die wichtigsten geometrischen Objekte in *POV-Ray*

Kugel:

```
sphere{< 0, 1, 0>, 0.5
                texture{...}}
```

Mittelpunkt der Kugel $<x_M.y_M,z_M>$, Radius ...

Zylinder:

```
cylinder{< 0, 0, 0>,< 0, 1, 0>, 0.1
                    pigment {...}}
```

Mittelpunkt auf der einen Seite $<x_{M1},y_{M1},z_{M1}>$,
Mittelpunkt auf der anderen$<x_{M2},y_{M2},z_{M2}>$, Radius.

Quader:

```
box{<-1,-1,-1>,< 1, 3, 2>
            pigment {...}}
```

Die zwei entgegen gesetzten Ecken: $<x_1.y_1,z_1>$ $<x_2,y_2,z_2>$

Kegel:

```
cone{< 0, 0, 0>, 1.25, < 0, 2.5, 0>, 0.5
                pigment {...}}
```

Mittelpunkt auf der einen Seite $<x_{M1},y_{M1},z_{M1}>$, Radius auf der einen Seite,
Mittelpunkt auf der anderen Seite$<x_{M2},y_{M2},z_{M2}>$, Radius auf der anderen Seite

Arbeitsblatt I – Methoden, Klassen und Vererbung

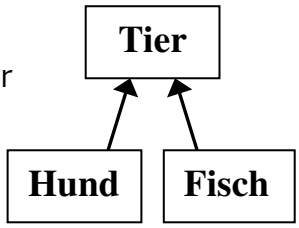
Merke:

Vererbung

Die Vererbung dient dazu, aufbauend auf existierenden Klassen neue zu schaffen, wobei die Beziehung zwischen ursprünglicher und neuer Klasse dauerhaft ist. Eine neue Klasse kann dabei eine Erweiterung oder eine Einschränkung der ursprünglichen Klasse sein.

Ober-/Unterklasse

Die vererbende Klasse wird *Oberklasse* (auch *Basis-* oder *Elternklasse*) genannt, die erbende *Unterklasse* (auch *abgeleitete* oder *Kindklasse*).



Einfach-/Mehrfachvererbung

Bei diesem Bespiel handelt es sich um Einfachvererbung, da jede Unterklasse (Hund, Fisch) höchstens von einer Oberklasse (Tier) Merkmale erbt.
Um Mehrfachvererbung handelt es sich, wenn eine Unterklasse direkt von mehr als einer Oberklasse erbt.

Methoden in *POV-Ray*:

union{...}	Vereinigung: Einfaches Zusammenfassen von Objekten
difference{...}	Differenz: Bildung der Schnittmenge von Objekten
intersection{...}	Schnittmenge: Abziehen eines Objektes von einem anderen
translate<x,y,z>	Verschieben eines Objektes
scale<x,y,z>	Größe des Objekts ändern
rotate<x,y,z>	Drehen eines Objekts

Klassen in *POV-Ray*:

Klasse in *POV-Ray* konstruieren:

#declare *Klassenname* = union{ ---Objekte innerhalb Klasse mit ihren Attributen---}
oder
#declare *Klassenname* = difference{ ... }

Eine Instanz der konstruieren Klasse aufrufen

object {*Klassenname*
translate<-2,0,0> scale ... }

Arbeitsanweisungen:

In dieser Einheit werdet ihr eine kleine Stadt programmieren. Ihr bekommt eine Vorlage, in der schon eine Haus vorhanden ist und sollt aus der Klasse „Haus“ weitere Klassen von Häusern ableiten, also Unterklassen erstellen. Verwendet dazu auch die neue gelernten Methoden.

1. Öffne das Dokument „Stadt_Vorlage.pov“.
2. Schaue dir die Klassendefinition der Klasse „Haus“ an.
3. Erzeuge einige Häuser. Setze sie nebeneinander mit der Methode translate<x,y,z>
4. Schreibe eine Unterklasse, z.B. „HausMitSchornstein“.
5. Erzeuge auch davon einige Instanzen.
6. Schreibe weitere Unterklassen: Häuser mit Fenstern und Türen. Verwende dazu die neu gelernten Methoden.
7. Experimentiere etwas mit *POV-Ray*. Verwende verschiedenen (auch vordefinierte) Objekte und setzte Attribute.

Bei Problemen mit passenden Koordinaten für Schonstein, Fenster etc. frage den Lehrer um Rat.

8. Experimentiere mit der Kameraeinstellung:

```
#declare Camera_0 = camera {/*ultra_wide_angle*/ angle 75
                            location  <1.5 , 2.0 ,-3.0>
                            right     x*image_width/image_height
                            look_at   <0.0 , 1.0 , 0.0>}
```

9. Wechsel die Kamera indem du hier eine andere Nummer einträgst:
10. camera{Camera_0}

Zusatz:

Erzeuge selbst eine neue Kamera: Camera_4 mit beliebigen Einstellungen-

Bei Fragen an den Lehrer wenden.

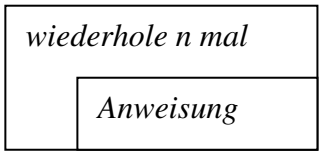
Arbeitsblatt II – Die *while*-Schleife

Merke:

Ein **Algorithmus** ist eine Verarbeitungsvorschrift, die aus einer endlichen Folge von eindeutig ausführbaren Anweisungen besteht.
Die Darstellung eines Algorithmus in einer für den Computer verständlichen Form heißt **Programm.**

Zählschleife:
Wiederholung mit fester Anzahl:

Wiederhole *n* mal
    Anweisung
*wiederhole



(Die Anweisungsfolge innerhalb der Wiederholung wird so oft ausgeführt, wie die Zahl n vorgibt. Die Anweisung, die wiederholt werden soll, steht eingerückt zwischen den Schlüsselwörtern „Wiederhole *n* mal“ und „*wiederhole“.)

Methoden und Ausdrücke, die nach ihrer Ausführung den Wahrheitswert „wahr“ oder „falsch“ annehmen heißen **Bedingungen**.

While-Schleife:
Wiederholung mit Eingangsbedingung

Solange Bedingung tue
    Anweisung
*solange

(Zuerst wird die Bedingung überprüft. Ist sie erfüllt, wird die Anweisung ausgeführt. Solange die Eingangsbedingung wahr ist, wird dieser Vorgang wiederholt. Die Anweisung, die wiederholt werden soll, steht eingerückt zwischen den Schlüsselwörtern „solange … tue“ und „*solange“.)

Um die Schleife n mal durchzulaufen, muss eine Variable definiert werden, die von 0 bis n hochgezählt wird. Variablen werden mit #declare definiert.

Arbeitsanweisungen:

Nun programmiert ihr selbst einige elementare Anwendungen der *while*-Schleife in *POV-Ray*. Verwendet hierbei wird die Schleife (engl.: loop) zum regelmäßigen Anordnen von Objekten.
Programmiert erst einmal eine einfache Schleife, welche kleine Kugeln längs der x-Achse platziert:

1. Lege ein neues Dokument an.
2. Speichere es in deinem *POV-Ray* Ordner unter dem Namen: „Kugeln.pov“
3. Füge folgende Szene ein: Insert -> Basic templates -> Ready-made scenes -> Basic Scene M4
4. Füge in den Code **unter** der Szenenbeschreibung eine beliebige Kugel ein (Insert -> Basic templates -> Shapes 0 with textures -> sphere with texture)
5. Gibt der Kugel einen Namen, z.B: „Ball“ (mit `#declare Ball =`)
6. Überlege wie oft die Kugel nebeneinander gesetzt auf die x-Achse passt (auch in die negative Richtung)
7. Führe passende Variablen für Zählbeginn und –ende mit `#declare` ein (beachte: Position der Kugeln: <0+*n*,0,0>)
8. Eine *while*-Schleife wird wie folgt programmiert:

```
#while (Bedingung)
        Anweisung
#end
```

9. Überlege, wie man die Schleife zum Abbruch bringt (Variable hochzählen?)

Bisher wurde die *while*-Schleife nur mit Translations-Bewegungen (Parallelverschiebungen) angewandt. Ein interessanter Effekt ergibt sich, wenn man in gleicher Weise Rotationsbewegungen ausführen lässt.

Zusatz: Speichere dein Dokument unter dem Namen „KugelKreis.pov“ ab. Ordne die Kugeln in Kreis um den Ursprung des Koordinatensystems an. Überlege dir hierfür, aus wie vielen Kugel dein Kreis bestehen soll und in welchem Winkel die einzelnen Kugeln gedreht werden müssen (um den Ursprung).

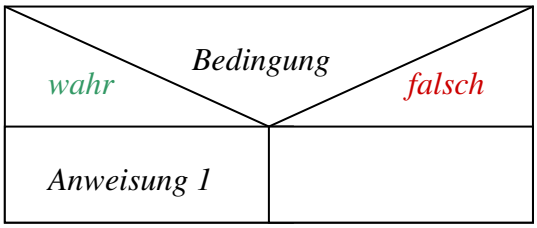
Programmiere nach Belieben noch andere Schleifen mit anderen Objekten.
Schau dir unter Insert -> Basic templates -> While loops, Spline curves an, was man noch alles mit Schleifen programmieren kann.

Bei Fragen an den Lehrer wenden.

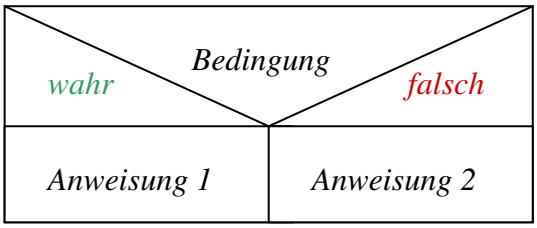
Arbeitsblatt III – *if-else* Anweisungen

Merke:

Einseitige Anweisungen



Zweiseite Anweisungen



Eine *if-else* Anweisung besteht aus einer Bedingung und einer oder mehreren Anweisungen. Wenn im Programm die *if-else* Anweisung erreicht wird, dann wird erst die Bedingung ausgewertet, und wenn sie zutrifft (und nur dann) wird anschließend die Anweisung 1 ausgeführt. Trifft die Bedingung nicht zu, so wird die Anweisung 2 ausgeführt, wenn sie existiert.
Danach wird das Programm nach der *if-else* Anweisung fortgesetzt.

Bedingte Anweisung:

```
wenn Bedingung dann
        Anweisung 1
sonst
        Anweisung 2
*wenn
```

Programmcode:

```
#if (Bedingung)
 ... Anweisung 1
#else // (optional)
 ... Anweisung 2
#end
```

Geschachtelte Anweisungen
(Beispiel:)

```
wenn Bedingung A dann
        Anweisung A1
sonst
        wenn Bedingung B dann
                Anweisung B1
        sonst
                Anweisung B2
        *wenn
*wenn
```

Arbeitsanweisungen:

Nun programmiert ihr selbst einige elementare Anwendungen der *if-else* Anweisung in *POV-Ray*. Verwendet die Anweisung hierbei, um Objekten unterschiedliche Farbe, Position oder Form zu geben.
Programmiert einen Zaun aus Holzpfählen mit unterschiedlichen Holztönen. Verschachtelt dazu die *if-else* Anweisung in einer *while*-Schleife.

1. Lege ein neues Dokument an.
2. Speichere es in deinem *POV-Ray* Ordner unter dem Namen: „Zaun.pov“
3. Füge folgende Szene ein: Insert -> Basic templates -> Ready-made scenes -> Basic Scene 04
4. Verändere in der Kameraeinstellung:
   location <0.0 , 4.0 ,-8.0>
   look_at <0.0 , 0.0 , 6.0>
5. Füge in den Code **unter** der Szenenbeschreibung einen hohen dünnen Zylinder ein, der wie ein Zaunpfeiler aussieht. Verwende hierzu einen Holzton aus Insert -> Basic templates -> Textures and Materials -> Woods (Höhe: 3, Radius: 0.4)
6. Füge noch einen Pfahl mit einem anderen Holzton ein.
7. Gib den Pfählen Namen, z.B: „Pfahl_Holzton_1“ (mit #declare)
8. Überlege wie lang der Zaun sein soll. (Errichte ihn entlang der x-Achse).
9. Führe passende Variablen für Zählbeginn und –ende der *while*-Schleife ein.
10. Füge innerhalb der Schleife eine *if-else* Anweisung ein, die dafür sorgt, dass jeder zweite Pfahl von einer anderen Holzart ist.

Zusatz: Speichere dein Programm unter „ZaunKomplett.pov“
ab und schreibe es so um, dass du einen Zaun erhältst, der eine quadratische Fläche einzäunt.

Die Ecken der quadratischen Fläche könnten folgende Koordinaten haben:
<-5,0,0>, <5,0,0>, <5,0,10>, <-5,0,10>

Versuche dich nach Belieben an weiteren geschachtelten Anweisungen mit verschiedenen Objekten.

Bei Fragen an den Lehrer wenden.

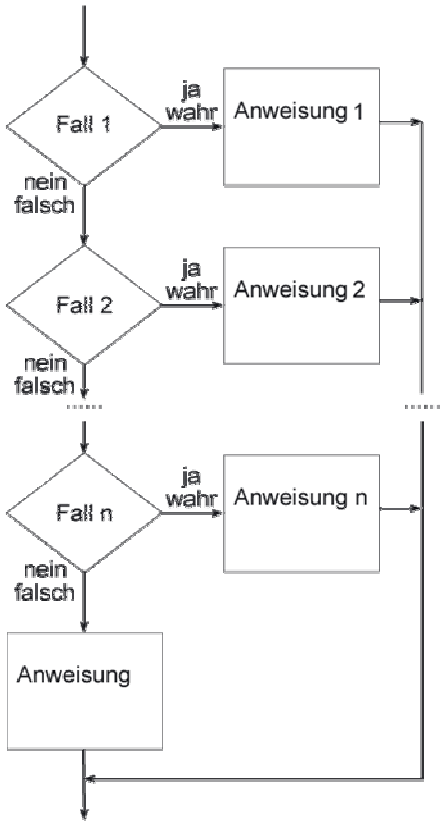
Arbeitsblatt IV – Switch-case Anweisungen

Merke:

Kontrollstrukturen werden in Programmiersprachen verwendet, um den Ablauf eines Computerprogramms zu steuern. Eine Kontrollstruktur gehört entweder zur Gruppe der Verzweigungen oder der Schleifen.

Für *POV-Ray* lautet die Syntax für *switch-case* Anweisungen:

```
#switch (Auswahlvariable)
#case (Wert 1 der Variable)
 ... Anweisung 1
#break
#case (Wert 2 der Variable)
 ... Anweisung 2
#break
....
#case (Wert n der Variable)
 ... Anweisung n
#break
#else
 ... Anweisung
#break
#end
```



Es können so viele Fälle angegeben werden wie gewünscht, allerdings keiner doppelt. *switch/case* kann man eigentlich auch mit *if/else* darstellen, allerdings ist es nicht vorteilhaft, da es erstens unleserlicher ist und zweitens die Ausführung länger dauert.

Achtung! Hüte dich vor **Endlosschleifen**.

Beispiel:

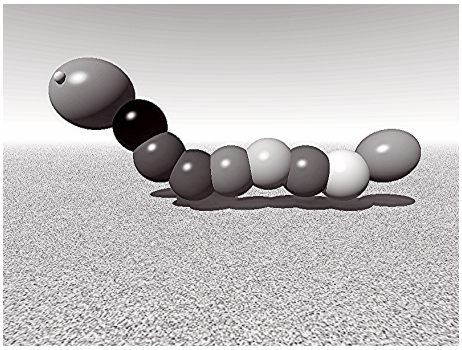
```
#declare Nr = -5;
#declare EndNr = 5;
#while (Nr< 10)
   .....
#end
```

Da „Nr“ immer kleiner ist als 10, wird die *while*-Schleife unendlich lange durchlaufen, da die Bedingung (Nr < 10) immer wahr ist.

Arbeitsanweisungen:

Nun programmiert ihr selbst einen Wurm mit Hilfe der *switch-case* Anweisung. Programmiert dazu eine Schleife die z.B. 10 Kugeln erstellt. Verwendet *switch/case* um die Kugel hintereinander zu platzieren, ihre Farbe und ihre Form zu verändern:

1. Lege ein neues Dokument an.
2. Speichere es in deinem *POV-Ray* Ordner unter dem Namen: „Wurm.pov“
3. Füge folgende Szene ein: Insert -> Basic templates -> Ready-made scenes -> Basic Scene 03
4. Schreibe die oben beschriebene Schleife.
5. Füge in die Schleife eine beliebige Kugel ein (Insert -> Basic templates -> Shapes 0 with textures -> sphere with texture)
6. Entscheide mit einer *switch-case* Anweisung, welche Textur die einzelnen Kugel haben sollen, die mit Hilfe der Schleife erzeugt werden. (Mit finish{phong 1} bekommt die Kugel einen schönen Glanz.)
7. Programmniere die *switch-case* Anweisung **in** den Code der Kugel ein.
8. Zur Orientierung: Eine der mittleren Kugeln des Wurms solle bei <0,1,0> liegen.
9. Kannst du dem Wurm noch Augen geben?



Zusatz: Fasse alle Teile/Kugeln des Wurms mit Hilfe von union{} zusammen. Verschiebe dann den kompletten Wurm mit translate und rotate oder ändere seine Größe mit scale.

Bei Fragen an den Lehrer wenden.

Arbeitsblatt V – Animation

Merke:

Grundsätzliche Animationsbefehle:

In der ini-Datei:

Input_File_Name="Schneemann.pov" (die Datei , die die Bilder liefert.)

Die Zeilen "**Initial_Frame=1**" und "**Final_Frame=30**" legen fest, dass 30 Bilder gerechnet werden sollen.
Die Zeilen "**Initial_Clock=0**" "**Final_Clock=1**" geben den Start- und Endwert für die clock-Variable an. Er sollte möglichst immer nur von 0 bis 1 gezählt werden.

Cyclic_Animation=On (es ist sehr nützlich bei zyklischen Animationen wie z.B. dem Drehen eines Objekts um 360 Grad: Es bewirkt, dass der clock-Unterschied 1 bei einer Frame-Anzahl von 10 in z.B. 10 Abschritte unterteilt wird.)

In der pov-Datei:

Zum Beispiel:
rotate < 0, 360*clock, 0>
(ein Objekt um die y-Achse drehen)

In der pov-Datei: Kamerabewegung (drehen und Höhe ändern):

```
// camera ------------------------------------
#declare Jump_Start  = 0.5;
#if (clock < Jump_Start )
 #declare Camera_Y = 1.00+(clock*10);
#else
 #declare Camera_Y = 11.00-(clock*10);
#end

camera {
 angle 38
 location <3,Camera_Y,-20>
 right x*image_width/image_height
 look_at <-3,3,5>
 rotate<0,-360*(clock+0.1),0>
} //------------------- end of camera
```

Arbeitsanweisungen:

Nun programmiert ihr selbst ein paar Schneemänner, die ihr dann animiert. Danach lasst kommt zur Bewegung der Schneemänner noch eine Kamerabewegung dazu.

1. Öffnet das Dokument „VorlageV.pov“
2. Speichere es in deinem *POV-Ray* Ordner unter dem Namen: „Schneemann.pov“
3. Schaue dir die vorgegebene Klasse Schneemann an. Erzeuge einen Schneemann.
4. Erstelle das Dokument „Schneemann.ini“.
5. Lasse den Schneemann um sich selbst rotieren. Hole die ini-Datei in den Fordergrund und starte die Bildererzeugung mit „Run“.
6. Öffne das Programm Bmp_zu_Avi. Lade die erzeugten Bilder in das Programm. Die Bilder befinden sich im gleichen Ordner wie die Datei „Schneemann.pov“.
7. Wähle 100ms pro Bild aus und starte die Vorschau.
8. Kannst du den Schneemann auch im Kreis laufen lassen?
9. Erstelle weitere Schneemänner im Bild (while-Schleife) und erstelle eine Animation. (Bewege alle Schneemänner.)
10. Bewege zusätzlich zum animierten Schneemann noch die Kamera. (if-else Anweisung)
11. Ändere die Klasse die Schneemänner, sodass die Schneemänner auch Augen, Nase etc. haben.

Bei Fragen an den Lehrer wenden.