**Ausarbeitung Software Engineering Projekt**

Dozent: Prof. Dr. rer. Nat. Wolfgang Pekrun

Simulation des Sonnensystems

Felix Willrich Frederik Rieß

70452988 70453642

Aaron Kurda Michael Keller

70453830 70453404

Jascha Schmidt

70453637

**Hochschule**

Ostfalia Hochschule für angewandte Wissenschaften  
Hochschule Braunschweig/Wolfenbüttel   
Salzdahlumer Straße 46/48  
38302 Wolfenbüttel



Inhaltsverzeichnis

[1. Einleitung 1](#_Toc807360)

[2. Projektmanagement 1](#_Toc807361)

[2.1. Besprechungen 1](#_Toc807362)

[2.2. Merkmale Klassisch & Agil 1](#_Toc807363)

[2.3. Aufgabenverteilung 2](#_Toc807364)

[2.4. Probleme 2](#_Toc807365)

[3. Projektplanung 2](#_Toc807366)

[3.1. Themenabsprache 2](#_Toc807367)

[3.2. Projektziel 2](#_Toc807368)

[3.3. Projektabgrenzung 2](#_Toc807369)

[4. Projektvorbereitung 3](#_Toc807370)

[4.1. Projektumgebung 3](#_Toc807371)

[4.2. Datengrundlage 3](#_Toc807372)

[5. Projektumsetzung 4](#_Toc807373)

[5.1. UI 4](#_Toc807374)

[5.2. Berechnung der Elipsen 6](#_Toc807375)

[5.3. Simulation der Umlaufbahnen 8](#_Toc807376)

[5.3.1. Das Zwei-Körperproblem 8](#_Toc807377)

[5.3.2. Das N-Körperproblem 9](#_Toc807378)

[5.3.3. Das Euler-Verfahren 10](#_Toc807379)

[5.3.4. Das klassische Runge-Kutta-Verfahren 11](#_Toc807380)

[5.3.5. Vergleich der Verfahren 12](#_Toc807381)

[5.4. Fehlerberechnung 13](#_Toc807382)

[5.4.1. Messwerte 13](#_Toc807383)

[5.4.2. Referenzwerte 13](#_Toc807384)

[5.4.3. Durchführung 13](#_Toc807385)

[5.4.4. Berechnete Abweichungen 13](#_Toc807386)

[5.4.5. Analyse 17](#_Toc807387)

[5.4.6. Fehlerquellen 17](#_Toc807388)

[6. Fazit 18](#_Toc807389)

[7. Abbildungsverzeichnis 19](#_Toc807390)

[8. Quellen 20](#_Toc807391)

# Einleitung

Die nachfolgende Projektdokumentation wurde angefertigt, um das Projekt im Rahmen der Lehrveranstaltung „Software Engineering“ Projekt an der Ostfalia Hochschule für angewandte Wissenschaften im Studiengang Informatik zu veranschaulichen.

Die Hochschule besteht aus vier Standorten (Wolfenbüttel, Wolfsburg, Suderburg und Salzgitter). Gegründet wurde sie im Jahr 1971 und zählt heutzutage ca. 13000 Studenten.

# Projektmanagement

Die Projektherangehensweise wurde zum Anfang des Projektes still festgelegt. Die Gruppe hatte schon mehrere Projekte zusammen bearbeitet und aus diesem Grund wusste jedes Teammitglied, wie der Projektzeitraum ablaufen wird.

Unsere Methode bestand aus einer Vermischung von einem agilen und klassischen Ansatz.

## Besprechungen

Die regelmäßigen Treffen fanden circa alle 1-2 Wochen statt. In diesen Treffen wurden jeweils der aktuelle Stand des Programms, die zukünftigen Aufgaben und Probleme besprochen. Weiterhin wurden Ideen besprochen.

Die meisten Treffen fanden nach dem folgenden Schema statt:

1. Aktuelle Lage
2. Probleme mit den vorherigen Aufgaben
3. Neue Ideen
4. Probleme Allgemein
5. Aufgabenverteilung

Während dieser Besprechungen konnte jedes Mitglied seine Ideen und Vorschläge mit einbringen. Es gab keinen direkten Produkt Manager, sondern nur einen Leiter, der versucht hat am Ende alle Meinungen und Ideen zusammenzubringen.

## Merkmale Klassisch & Agil

Wie eingangs erwähnt bestand unsere komplette Projektphase aus einem Mix aus klassischen und agilen Methoden. Die lag daran, dass wir uns für kein direktes Modell entschieden haben, sondern versucht haben für unsere Gruppengröße von fünf Mitgliedern ein geeignetes Modell zu finden.

Unsere agile Arbeitsweise bezieht sich vor allem auf die flexiblen Anforderungen. Unsere Ideen um das Projekt haben sich konstant verändert. Vor allem die genaue Richtung zum Ende hin war am Anfang unklar. Dies konnte sich erst während der Projektphase herausstellen.

Gleichermaßen konnte man den Abstand zwischen den Treffen als einen Sprint sehen. Zu jedem neuen Sprint gab es eine Besprechung und zu jedem abgelaufenen eine Nachbearbeitung.

Die klassische Arbeitsweise bezieht sich vor allem auf das Endprodukt, da der Kunde (Professor Pekrun) nur das Endprodukt erhält. Außerdem wusste jeder ungefähr welche Aufgaben er übernehmen wird während der Entwicklung.

## Aufgabenverteilung

Die Aufgaben wurden vor allem nach Interessen und Wissensstand verteilt. Da in unserem Projekt vor allem die Mathematik und Physik eine große Rolle spielt wurden die jeweiligen Teile auf die entsprechenden Personen aufgeteilt. Weiterhin gab es einen Programmierteil, der für alle fünf relevant war.

Aaron = Physik, Mathematik, Unity Programmierung

Frederik = Physik, Mathematik, Unity Programmierung

Michael = Unity Programmierung, Oberflächenprogrammierung

Jascha = Fehlerabschätzung

Felix = Projektmanagement, Oberflächenprogrammierung

## Probleme

Die Probleme während der Entwicklungsphase wurden untereinander abgesprochen und geklärt. Sollte ein Gruppenmitglied nicht weiterkommen, wurde die Aufgabe in der gesamten Gruppe noch einmal besprochen und versucht zu lösen. Sollte diese Methode nicht funktioniert haben, wurde die Aufgabe weiterverteilt, um einen Ansatz zu versuchen.

# Projektplanung

Da das Projekt im Rahmen einer Lehrveranstaltung erstellt wird, gab es vor ab eine Besprechung, welches Thema ausgewählt wird und auf welcher Basis das Projekt erstellt wird.

Professor Pekrun hatte in der ersten Vorlesung erwähnt, dass das Projekt eine Simulation über ein bestimmtes Themenfeld sein soll. Dies war die einzige Voraussetzung.

## Themenabsprache

Im ersten Treffen zwischen der Gruppe wurden diverse Themenvorschläge ausgetauscht. Am Ende des Meetings gab es die Wahl zwischen einer Ameisenkolonie- und einer Planentenumlaufbahnsimulation. In Rücksprache mit Herr Pekrun hatten wir uns für den zweiten Vorschlag entschieden. Daraus entwickelte sich folgendes Anforderungsprofil für unser Projekt.

## Projektziel

Das Projekt soll dazu dienen die Umlaufbahnen vom Sonnensystem zu simulieren. Um die korrekte Funktionsweise zu überprüfen, sollen die Daten am Ende mit offiziellen Daten verglichen werden. Dieser Vergleich wird in mehreren Testdurchläufen durchgeführt (1 Jahr, 10 Jahre,100 Jahre).

Simuliert werden alle Planeten im Sonnensystem. Gleichzeitig soll es die Möglichkeit geben, verschiedene Objekte wie z.B. ein Asteroid durch die Umlaufbahn zu schicken, um den Einfluss von Fremdkörpern zu simulieren.

Die Daten, die aus der Simulation gewonnen werden, sollen ausgegeben und mit offiziellen Daten verglichen werden.

## Projektabgrenzung

Anfangs war das Projekt dazu gedacht eine Aussage über die Umlaufbahnentwicklung der nächsten 100.000 Jahre zu treffen. Da dies den Projektrahmen aufgrund von Genauigkeit übertreffen würde, entwickeln wir in diesem Schritt ein Prototyp mit einem möglichst kleinen Berechnungsfehler.

# Projektvorbereitung

Wie in der Einleitung erklärt, wurde das Programm für die Veranstaltung „Software Engineering“ entwickelt. Da dies in Gruppenarbeit geschehen ist, wurde ein eigenes Github Repository angelegt, um einen Austausch zu ermöglichen <https://github.com/Voxen4/Planetarium>.

## Projektumgebung

Das Projekt wurde mit der Game-Engine Unity 3D erstellt. Unity wurde 2005 von Unity Technologies in Amerika entwickelt. Es unterstützt 2D bzw. 3D Anwendungen, sowie Simulationen und bringt diverse Tools zur Berechnung und Darstellung von Objekten mit. Gleichzeitig läuft Unity auf verschiedensten Plattformen.

Für unsere Simulation haben wir die Programmiersprache C# benutzt.

## Datengrundlage

Für unser System ist es essential wichtig Daten zum vergleichen zu haben. Diese Daten bilden die Grundlage unserer Startpositionen bzw. dienen zum Vergleich der Umlaufbahnen.

Um an diese Daten zu gelangen, haben wir im Zuge des Projekts einige Planetarien (Planetarium Berlin), Universitäten (TU BS) und Sternenwarten (Thüringer Landessternwarte) angeschrieben. Die meisten dieser Ansprechpartner haben uns direkt auf die Seite der NASA verwiesen. Das „Horizon Web-Interface“ bietet für verschiedenste Objekte im All Daten an. Diese Daten reichen von 1600-2500. Für unsere Daten haben wir meist folgende Einstellungen gewählt:

|  |  |
| --- | --- |
| **Ephemeris Type** | VECTORS |
| **Target Body** | Jeweiliger Planet |
| **Coordinate Origin** | Sun (body center) |
| **Time Span** | Variabel (1 day) |

Ein Beispieldatensatz hat folgendes Format:



VX X-component of velocity vector (au/day)

VY Y-component of velocity vector (au/day)

VZ Z-component of velocity vector (au/day)

LT One-way down-leg Newtonian light-time (day)

RG Range; distance from coordinate center (au)

RR Range-rate; radial velocity wrt coord. center (au/day)

Zu der Schnittstelle liegt eine Dokumentation vor, welche die jeweiligen Werte beschreibt. Für unser Projekt waren vor allem die X, Y und Z Koordinaten von Relevanz.

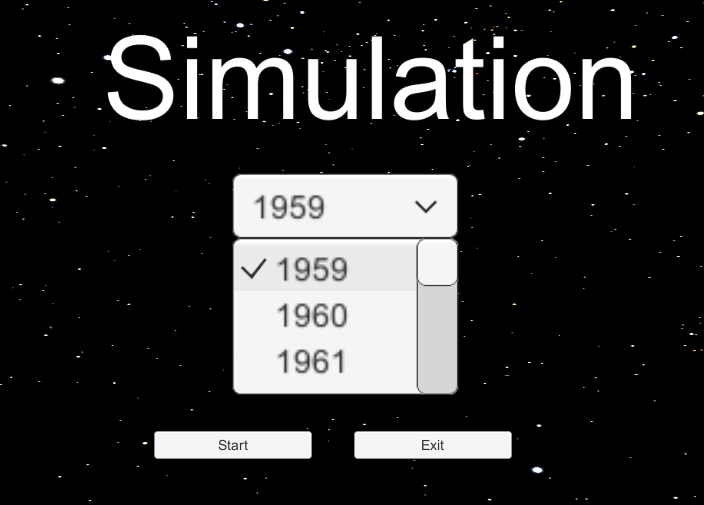
# Projektumsetzung

Die Simulation wurde abwechselnd in zwei Schritten umgesetzt. Auf der einen Seite wurde jeweils die Mathematik/Physik hergeleitet. Die andere Seite beinhaltete die Implementierung der Formeln in Unity und damit Berechnung der jeweiligen Daten zu den Planeten.

Das Ziel war es wie Anfangs erwähnt eine Simulation der Umläufe der Planeten zu erschaffen und anschließend eine Fehlerabschätzung durchzuführen. Dazu sollen Objekte durch den Orbit geschossen werden, um den Einfluss dieser Objekte zu simulieren.

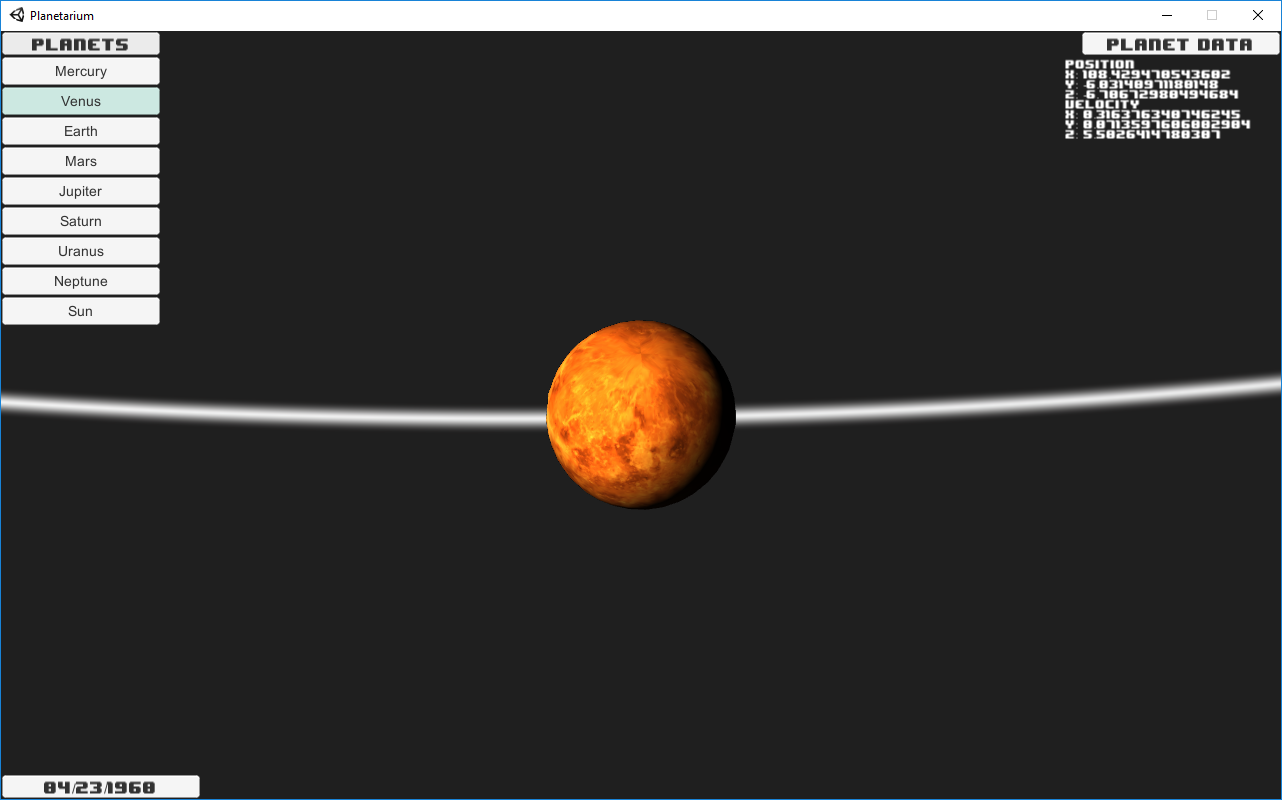
## UI

Die Oberfläche des Projekts besteht aus 2 Szenen, dem Start Menü und der Hauptszene.



In dem Start Menü lässt sich das Startdatum der Simulation über ein Dropdown-Menü auswählen, mögliche Werte reichen von 1959 bis 2019. Das resultierende Datum ist jeweils der 01.01 des ausgewählten Jahres. Mit dem Start-Button oder durch drücken von „ESC“ in einer der Szenen lässt sich die Simulation starten bzw. stoppen und die Szene wechseln. Ein Button zum Schließen des Spiels ist ebenfalls Teil der Menüszene.

Die Hauptszene besteht aus einem Overlay zur Anzeige diverser Informationen, wie etwa das aktuelle Datum der Simulation im englischen Format (MM/DD/YYYY), einem Dropdown zur Auswahl eines Planeten und die Anzeige der Position sowie Geschwindigkeit dieses Planeten.



Das Dropdown zur Auswahl eines Planeten öffnet sich beim Bewegen der Maus über diese. Zur Auswahl eines Planeten muss der entsprechende Name angeklickt werden. Es kann jederzeit nur ein Planet ausgewählt werden. Vor dem Anwählen eines anderen Planeten muss der bisher ausgewählte Planet durch erneutes anklicken des Namens abgewählt werden.

Planeten, die durch das Dropdown angewählt wurden, werden fokussiert und automatisch verfolgt.

Mithilfe des Mausrads und des „W“ bzw. „S“ Buttons lässt sich die Kamera in der Szene vorwärts bzw. rückwärts bewegen.

Die Taste „A“ bewegt die Kamera nach links und die Taste „D“ nach rechts.

Die Kamera lässt sich außerdem 3D Drehen, indem man Rechtsklick gedrückt hält und diese durch Bewegen der Maus dreht.

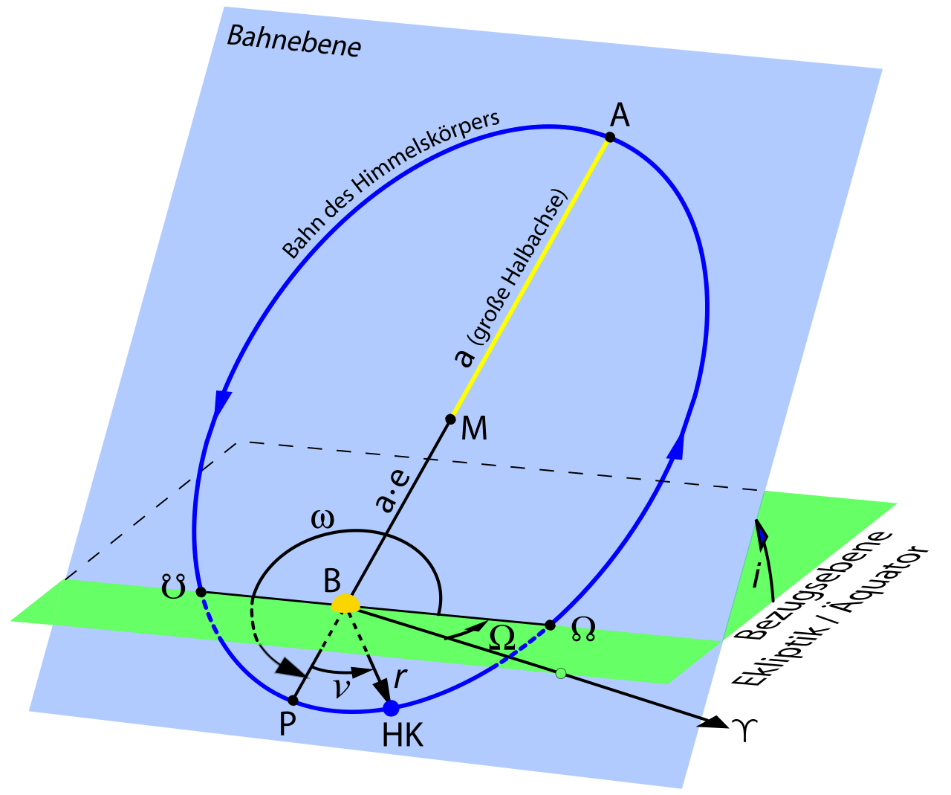
Beim Bewegen der Maus über einen Planeten wird der Name des Planeten angezeigt.

Um zu veranschaulichen dass es sich bei unserer Simulation wirklich um eine solche handelt gibt es die Möglichkeit mit einem Druck auf die Leertaste einen Asteroiden zu „spawnen“, dieser hat eine Masse von genau 1e+25.Die Richtung in die der Asteroid sich bewegt lässt sich dabei mit der Maus Position beim Drücken der Leertaste bestimmen. Je weiter der Ort zu dem der Mauszeiger zeigt entfernt ist desto schneller ist der Asteroid.

Die errechneten Daten der Simulation werden im Hauptverzeichnis der Anwendung gespeichert, jeder Planet hat eine eigene Text Datei. Die Log Datei ist mit Tabs getrennt und in der ersten Zeile mit den Spaltennamen versehen.

## Berechnung der Elipsen

Für die physikalische Simulation von Planeten ist ein Parameter unabdingbar: die Richtung und Stärke der initialen Geschwindigkeit „v“ (siehe Abbildung) zu Beginn der Simulation. Um die Richtung der initialen Geschwindigkeit bestimmen zu können, müssen wir zuerst die Umlaufbahn des Planeten näher betrachten.



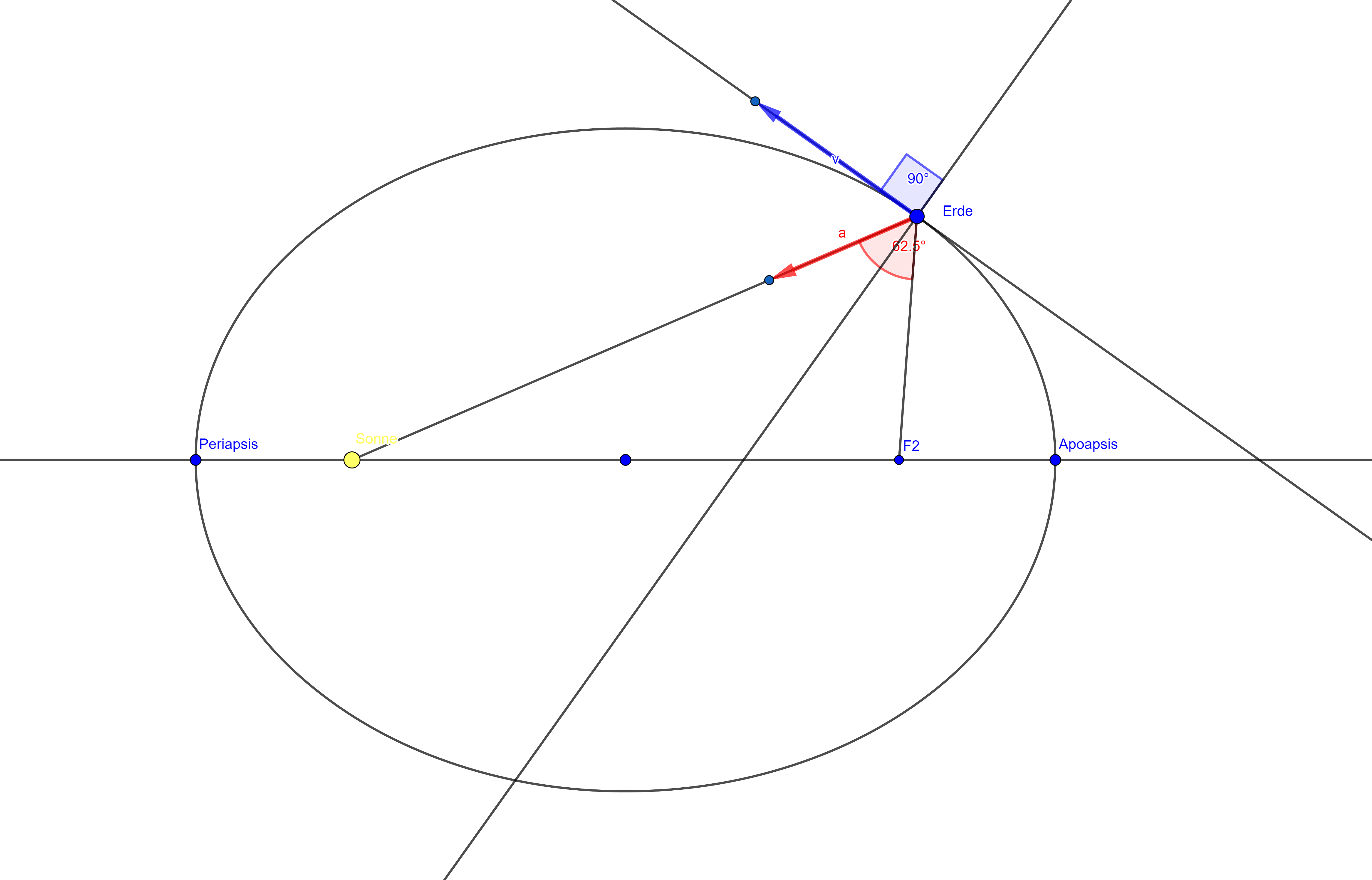
Um überhaupt den Orbit eines Planeten beschreiben zu können müssen wir zuerst Bezugspunkte definieren, zu denen dann die Ellipse relativ ausgerichtet wird. Bei der Bezugsebene handelt es sich bei uns einfach um die Äquatorialebene der Sonne (XZ-Ebene) und unsere Referenzrichtung ist die X-Achse.

Die Bahnebene des Planeten ist durch zwei Winkel definiert:

* Ω / Länge des Aufsteigenden Knotens ist ein Winkel von der vorher definierten Referenzrichtung hin zu einem Punkt, an dem der Planet die Referenzebene von unten durchläuft. Eine Grade vom Stern zu diesem Punkt beschreibt daher eine Schnittgrade der Bahnebene und Bezugsebene.
* Ί / Inklination beschreibt den Winkel zwischen Bahnebene und Referenzebene.

Mit Vektoren und Drehungen wird diese Ebene zu konstruiert. Zuerst wird der Vektor der Referenzrichtung (bei uns die X-Achse) um den Winkel Ω auf der Bezugsebene (bzw. um den Normalenvektor der Bezugsebene) gedreht. Den daraus resultierenden Vektor nennen wir e1. Nun drehen wir e1 um weitere 90° auf der Bezugsebene, sodass dieser orthogonal zu unserem Vektor e1 ist. Danach wird Vektor um den Wert Ί um den Vektor e1 gedreht. Dieser wird e2 genannt. e1 und e2 bilden die Vektoren der Bahnebene.

Um die Ausrichtung der Umlaufbahn auf der Bahnebene zu beschreiben wird ein weiterer Winkel ω / Argument der Periapsis benötigt. Dies ist der Winkel vom aufsteigenden Knoten (e1) bis zur Periapsis des Orbits. Durch erneutes Drehen des Vektors e1 um den Normalenvektor der Bahnebene erhalten wir die genaue Orientierung der Periapsis. Um die Richtung der initialen Geschwindigkeit berechnen zu können, wird gleichzeitig die Position der beiden Brennpunkte der Ellipse benötigt.



Dafür wird der Einheitsvektor m, der die Richtung von Periapsis zur Sonne beschreibt, mit berechnet. Da sich die Periapsis auf der großen Halbachse der Ellipse befindet, lässt sich die Position des Zentrums mit berechnen, wobei a die Länge der großen Halbachse ist. Ausgehen von diesem Punkt kann die Position des zweiten Brennpunkts berechnet werden mit , wobei e die Exzentrizität der Ellipse ist.

Mit den beiden Brennpunkten der Ellipse beschrieben, konstruieren wir zwei weitere Vektoren und . Die Richtung der Geschwindigkeit ist immer orthogonal zur Winkelhalbierenden dieser beiden Vektoren. Wir bestimmen also den Winkel α zwischen den beiden Vektoren d1 und d2 und drehen einen davon um in die Richtung des Anderen. Anschließend wird orthogonaler Vektor zur Winkelhalbierenden konstruiert, indem dieser um 90° gedreht wird.

Um die Länge dieses Vektors zu bestimmen wird die Vis-Viva-Gleichung verwendet. Durch einsetzten der bekannten Parameter in wird die Länge des Vektors und somit die initiale Geschwindigkeit v berechnet. (r := Abstand Erde-Sonne,a:=große Halbachse,M:= Masse der Sonne,m:=Masse des Planeten) Die Implementierung ist kommentiert im Code zu sehen (Assets\Scripts\Planets\PlanetData.cs).

## Simulation der Umlaufbahnen

Für uns stellte sich die Frage, wie wir ohne die Physik-Engine von Unity die Umlaufbahnen der Planeten darstellen können. Von interessanter Bedeutung war auch, ob man in Zukunft die Positionen der Planeten unseres Sonnensystems vorhersagen kann. Ob und wie dies möglich ist, wird im Folgenden mit Bezug auf das sogenannte N-Körperproblem erklärt. Für unsere Simulation musste schließlich eine geeignete Methode sondiert werden, damit alle Planeten miteinander interagieren und so auf ihren möglichst exakten Umlaufbahnen laufen.

### Das Zwei-Körperproblem

Durch das Zweikörperproblem wird die Bewegung zweier Körper beschrieben, die in Wechselwirkung stehen. Schon zu Beginn des 17. Jahrhunderts erkannte Johannes Kepler durch Beobachtungen, wie sich die Himmelskörper bewegen. Dadurch wurden die drei sogenannten keplerschen Gesetze entwickelt.

1. Keplersche Gesetz:

Die Planeten bewegen sich auf elliptischen Bahnen mit der Sonne als einen ihrer Brennpunkte.

1. Keplersche Gesetz:

Ein von der Sonne zum Planeten gezogener Fahrstrahl überstreicht in gleichen Zeiten gleich große Flächen.

1. Keplersche Gesetz:

Die Quadrate (zweite Potenzen) der Umlaufzeiten zweier Planeten um das gleiche Zentralgestirn verhalten sich wie die Kuben (dritte Potenzen) der großen Bahnhalbachsen

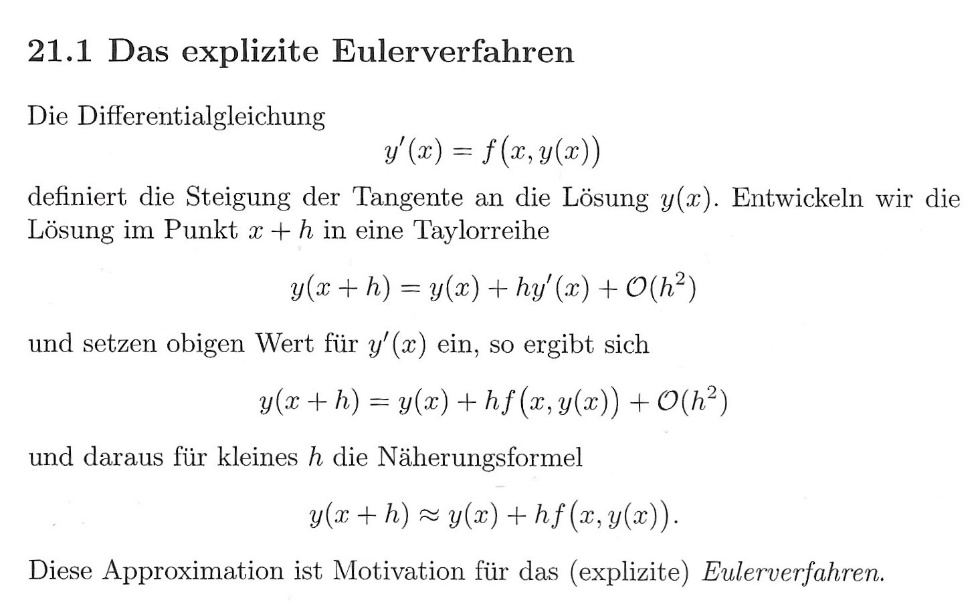
Später wurden diese Gesetze durch Newtons Werk „Philosophiae Naturalis Principia Mathematica“ mathematisch begründet und somit die physikalischen Wechselwirkungen zwischen zwei Körpern erklärt.

### Das N-Körperproblem

Da unser Sonnensystem aber mehr als zwei Körper besitzt, stellt sich die Frage, wie dieses Problem gelöst werden kann. Daraus ergibt sich das sogenannte N-Körperproblem oder die Frage: Wie kann die Wechselwirkung zwischen allen Planeten unseres Sonnensystems berechnet werden?

In den letzten Jahrhunderten haben sich viele Mathematiker an diesem Problem versucht, aber scheiterten letztendlich. 1889 bewies Henri Poincaré schließlich, dass sich dieses Problem nie lösen lassen wird, wenn mehr als zwei Körper in dem System beteiligt sind. Das bedeutete zugleich auch, dass die genauen Positionen der Planeten in der Zukunft nicht vorhersagbar sind. Nun entdeckten viele Wissenschaftler eingeschränkte Lösungen für dieses Problem, bis Karl Frithiof Sundman eine unendliche mathematische Reihe entwickelte, die eine exakte Lösung darstellt. Die Reihe wird zu einem gewissen Zeitpunkt konvergieren, aber dieser Zeitpunkt ist ungewiss. Zudem wird bei der Reihe davon ausgegangen, dass sich zwei Körper nie berühren dürfen, sodass die Distanz der beiden Körper 0 ist. Daher ist bis heute keine richtige Lösung gefunden worden.

Wie lassen sich aber sonst die Bahnen von mehr als zwei Himmelskörpern beschreiben? In der Mathematik gibt es verschiedene numerische Verfahren, mit denen bestimmte nichtlineare Probleme approximiert werden können. Anhand des sogenannten Euler-Verfahrens soll gezeigt werden, wie die Approximation funktioniert.



Für die Herleitung unserer Annäherung an die elliptischen Bahnen der Planeten sind zwei wichtige Grundsätze zu beachten.

1. Die Beschleunigung ist die Änderung der Geschwindigkeit über eine gewisse Zeit
2. Die Geschwindigkeit ist die Änderung der Position über eine gewisse Zeit

Da es viele verschiedene Verfahren für die Approximation gibt, sind hier das bekannteste (aber auch einfachste) und ein genaueres Verfahren aufgeführt.

### Das Euler-Verfahren

Durch die gerade genannten Gleichungen kann, falls es einen festen Startpunkt und eine feste Startgeschwindigkeit gibt, eine Vorhersage getroffen werden, wo sich die nächste Position eines Punktes befindet.

Gegeben sei eine aktuelle Position und eine aktuelle Geschwindigkeit . Zudem muss ein Zeitschritt h (in diesem Fall ) vorgegeben sein. Die nächste Position berechnet sich dann wie folgt:

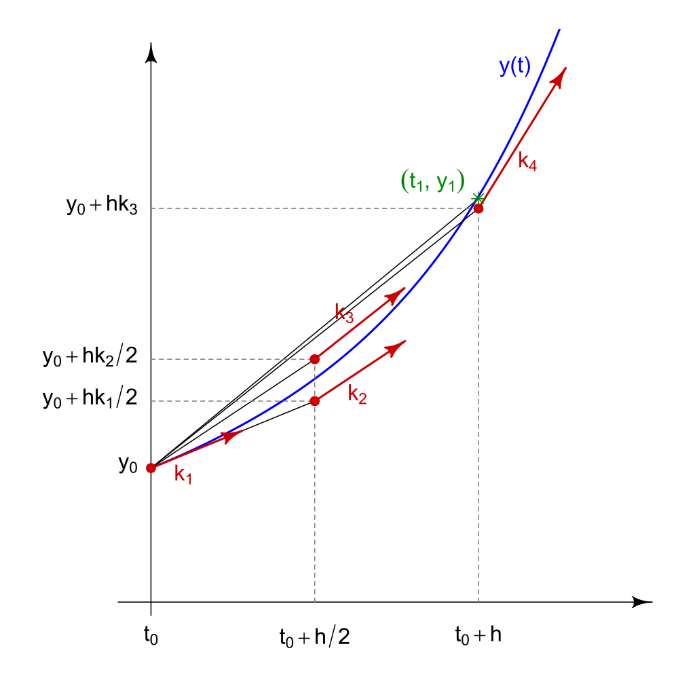
Die Geschwindigkeit ergibt sich entsprechend folgendermaßen:

Auf den ersten Blick wird erkenntlich, dass sich je nach Größe des Zeitschritts die Genauigkeit verändert. Wenn ein kleinerer Zeitschritt gewählt werden würde, stiege auch die Genauigkeit der Approximation an. Dementsprechend erhöht sich aber auch die benötigte Rechenzeit.

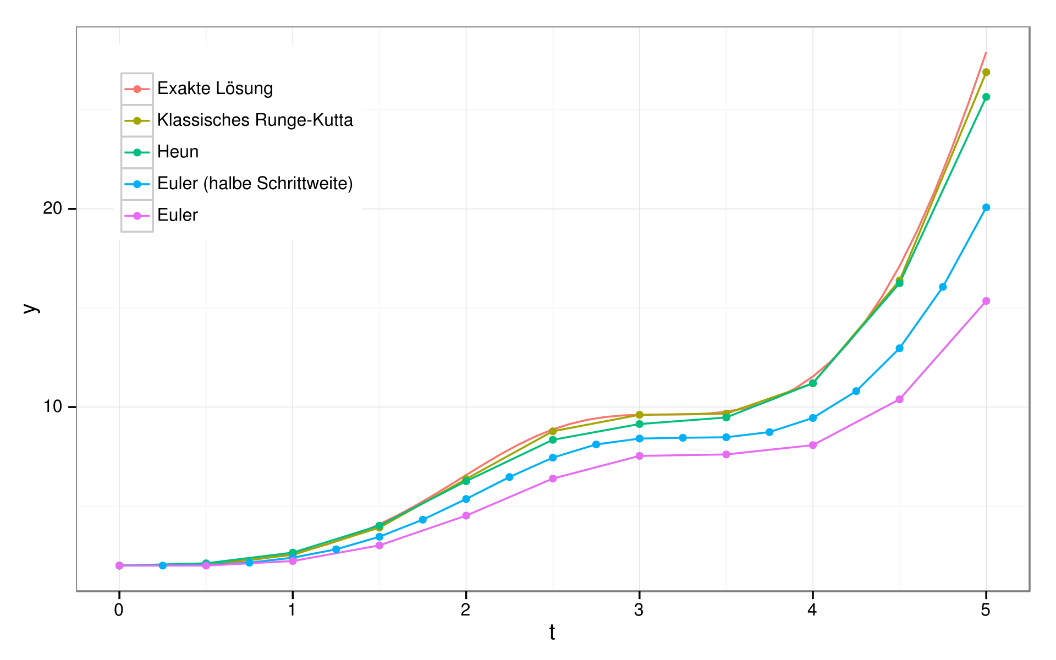
### Das klassische Runge-Kutta-Verfahren

Da unsere Annäherung an die elliptischen Bahnen der Planeten so genau wie möglich erfolgen soll, müssen die Positionen auch so genau wie möglich berechnet werden. Das Runge-Kutta-Verfahren ist ein moderneres Verfahren 4. Ordnung. Für die nächste Positionsbestimmung werden im Gegensatz zum Euler-Verfahren mehrere Werte bezüglich des Zeitschritts mit einbezogen. Der Fehler beträgt hierbei O(h5) und ist dementsprechend sehr gering.

Gegeben sei wieder ein Startpunkt y0, Startzeitpunkt t0 und eine Schrittweite h. Zudem ist eine exakte Lösung in Form der Funktion y(t) gegeben.

Es wird zunächst durch das Eulerverfahren in dem Startpunkt bestimmt, wo der zukünftige Punkt liegen könnte (k1). Nun wird bei der Hälfte der Schrittweite die Steigung bzw. der mögliche Zielpunkt berechnet (k2). Anschließend wird, ausgehend vom Startpunkt, mit der gleichen Steigung wieder bis zu der halben Schrittweite gegangen und erneut die Steigung berechnet (k3). Diese Steigung wird wiederum im Startpunkt benutzt, um von dort mit der gesamten Schrittweite eine temporäre Position zu berechnen. Von dieser Position wird das letzte Mal die Steigung berechnet (k4). Um nun den nächstmöglichen Punkt vollständig zu berechnen bzw. anzunähern, muss der Durschnitt der gesamten Steigungen betrachtet werden. Dabei haben die Steigungen unterschiedliche Gewichtungen. Der neue Punkt (t1, y1) ist schlussendlich der gesuchte nächste Punkt.

### Vergleich der Verfahren

****In der folgenden Abbildung ist eine Darstellung der Annäherung verschiedener numerischer Verfahren an eine exakte Lösung zu sehen. Deutlich wird hier, dass das Euler-Verfahren bei gleichem Zeitschritt gegenüber den anderen Verfahren viel ungenauer ist. Wird der Zeitschritt bei diesem Verfahren halbiert und noch weiter verkleinert, so sind durchaus achtbare Ergebnisse möglich. Das klassische Runge-Kutta-Verfahren ist hingegen bei der gleichen Schrittweite schon sehr genau. Daher fiel unsere Entscheidung bei der Simulation des Sonnensystems auf dieses Verfahren. Die genaue Implementierung des klassischen Runge-Kutta-Verfahrens unter Beachtung der physikalischen Einflüsse ist im Code zu sehen(..\Assets\Scripts\Camera\AttractionManager.cs).

## Fehlerberechnung

Die Fehlerberechnung wurde durchgeführt, um abzuschätzen wie genau unsere Simulation ist. Benutzte Werte sind der Mittelwert der Abweichungen und die Standartabweichung der Abweichungen (Formel: )

### Messwerte

**Quelle:** simulationsinterne Logging-Funktion mit X-, Y- und Z-Koordinate pro Tag

**Startdatum:** das Standartstartdatum 01.01.1959

**Simulationsdauer:** ca. 30 Jahre, ein Saturnjahr

### Referenzwerte

**Quelle:** HORIZONS Web-Interface, CSV-Datei mit X-, Y- und Z-Koordinate pro Tag

**Umfang:** Daten von 01.01.19 59 bis 31.12.2018, nachträglich auf 11664 Zeilen gekürzt

### Durchführung

Als erstes für jeden Planeten die Abweichung der Log-Daten von den Referenzdaten aus dem NASA-Horizon System berechnet. Als nächstes wird der Mittelwert der Abweichung ermittelt, um besser die Abweichung einschätzen zu können. Als letztes wird die Standartabweichung berechnet, um ab zu schätzen wie die Abweichungen um den Mittelwert der Abweichungen streuen.

### Berechnete Abweichungen

Die berechneten durchschnittlichen Abweichungen liegen bei:

**Merkur:** 0.271899517 ± 0.115495631 AU bei einer größten Halbachse von 0,387099273 AU

**Venus:** 0.065021841 ± 0.050744825 AU bei einer größten Halbachse von 0,723 AU

**Erde:** 0.060185074 ± 0.045403665 AU bei einer größten Halbachse von 1 AU

**Mars:** 0.221118521 ± 0.124092649 AU bei einer größten Halbachse von 1,524 AU

**Jupiter:** 0.27501267 ± 0.151079617 AU bei einer größten Halbachse von 5,203 AU

**Saturn:** 0.428685792 ± 0.122507339 AU bei einer größten Halbachse von 9,5826 AU

**Uranus:** 1.145036857 ± 0.743700934 AU bei einer größten Halbachse von 19,201 AU

**Neptun:** 0.920648795 ± 0.480065445 AU bei einer größten Halbachse von 30,070 AU

### Analyse

Bei genauerer Betrachtung der Daten wird deutlich, dass der Fehler immer weiterwächst. Dies hängt vor allem damit zusammen, dass unsere Simulation einen kleinen initialen Fehler in sich trägt. Bei uns werden keine Monde bzw. andere Objekte simuliert. Weiterhin wird die ganze Berechnung immer wieder zwischen Datentypen hin und her gecastet. Allein diese beiden Faktoren lassen den Fehler über die Zeit immer größer werden.

Am Beispiel der ersten Abbildung ist erkennbar, dass die Genauigkeit sehr stark schwankt. Eine Vermutung dazu ist, dass die initiale Geschwindigkeit nicht korrekt ist. Dadurch ist der Fehler beim Maximalabstand zur Sonne am höchsten und am Geringsten in der „Ausgangsposition“.

Weiterhin wird angenommen, dass der Jupiter die Sonne in kleinen Schritten aus seiner Position reißt und somit die anderen Planeten immer mehr beeinflusst und aus der Umlaufbahn stößt.

### Fehlerquellen

Mögliche Fehlerquellen in unserer Simulation sind Fehler in den Berechnungen & Rundungsfehler im Computer. Grade die Rundungsfehler im Computer und ungenaue Berechnungen haben bei der Messung über dreißig Jahre in der Simulation vermutlich einen Großteil der Abweichung verursacht. Verringern könnte man diese Fehler vielleicht indem man eine andere Engine benutzt oder das N-Körperproblem genauer betrachtet.

# Fazit

Abschließend bleibt über das Projekt zusagen, dass wir als Gruppe einiges über das Thema Physik im Zusammenhang mit dem Sonnensystem gelernt haben. Weiterhin konnten wir uns alle in das Thema Game Engine in Bezug von Unity einarbeiten. Dies hat den Vorteil, da ein paar Mitglieder der Gruppe den Weg einschlagen möchten, mit solcher Software zu arbeiten.

Auch zu erwähnen ist, dass der Aufwand für einen funktionsfähigen Prototypen sehr hoch angesetzt war, dies aber erst im Nachhinein deutlich wurde. Dies lag vor allem an den vielen verschiedenen Formeln und Fehlern, die sich in den Code dadurch eingeschleust haben.

Wenn wir als Gruppe dieses Thema noch einmal angehen würden, ist uns diesmal bewusst, dass wir andere Software benutzen würden. Die Unity Engine erfüllt ihren Job in angemessenen Rahmen, ist aber nicht auf eine solche Simulation ausgelegt. Alle Daten mussten skaliert und umgerechnet werden. Weiterhin haben wir eine eigene Physik Engine (Runge Kutta Verfahren) geschrieben, da die von Unity genutzte (Euler Verfahren) unseren Fehler deutlich vergrößern würde.

Frei nach dem Prinzip hinterher ist man schlauer, würden wir das Thema Simulation anders angehen.

# Abbildungsverzeichnis

# Quellen

<https://unity3d.com/de>

<https://ssd.jpl.nasa.gov/horizons.cgi>

<https://de.wikipedia.org/wiki/Klassisches_Runge-Kutta-Verfahren>

<http://nbodyphysics.com/blog/>

<http://spiff.rit.edu/richmond/nbody/OrbitRungeKutta4.pdf>

<http://www.tat.physik.uni-tuebingen.de/~kley/lehre/astroprakt/chaos/chaos.pdf>

<https://www.leifiphysik.de/mechanik/weltbilder-keplersche-gesetze/keplersche-gesetze>

<https://www.physnet.uni-hamburg.de/TUHH/Versuchsanleitung/Fehlerrechnung.pdf>

<https://www.spektrum.de/lexikon/physik/fehlerrechnung/4814>

Buch: Michael Oberguggenberger, Alexander Ostermann: Analysis für Informatiker, Seite 275