

# Semesterprojekt Physik Engines

Kim Lan Vu, Michel Steiner, Asha Schwegler

30. Mai 2023



# Inhaltsverzeichnis

<b>1 Zusammenfassung</b>	<b>3</b>
<b>2 Aufbau des Experiments</b>	<b>4</b>
2.1 Aufbau des Lab 2, „Würfel 1 bewegt sich und stösst“ . . . . .	4
2.2 Aufbau des Lab 3, „Beide Würfel schwingen gedämpft“ . . . . .	4
<b>3 Physikalische Beschreibung der einzelnen Vorgänge</b>	<b>6</b>
3.1 Lab 2: Würfel bewegt sich und stösst . . . . .	6
3.1.1 Konstante Kraft . . . . .	6
3.1.2 Elastischer Stoss . . . . .	7
3.1.3 Inelastischer Stoss . . . . .	8
3.2 Teil 3: Beide Würfel schwingen gedämpft . . . . .	8
3.2.1 Radialer Anteil der Gewichtskraft . . . . .	9
3.2.2 Zentripetalkraft . . . . .	9
3.2.3 Reibungskraft . . . . .	11
3.2.4 Resultierende Kraft . . . . .	11
<b>4 Beschreibung der Implementierung inklusive Screenshots aus Unity</b>	<b>12</b>
4.1 Lab 2: Würfel bewegt sich und stösst . . . . .	12
4.2 Lab 3: Beide Würfel schwingen gedämpft . . . . .	15
<b>5 Resultate mit grafischer Darstellung</b>	<b>17</b>
5.1 Lab 2 . . . . .	17
5.2 Lab 3 . . . . .	19
<b>6 Rückblick und Lehren aus dem Versuch</b>	<b>21</b>
<b>A Anhang</b>	<b>22</b>
A.1 Code für das Lab 2 und 3 . . . . .	22
A.2 Code für die Datenaufbereitung des Elastischen Stosses des Lab 2 . . . . .	43
A.3 Code für die Datenaufbereitung des Inelastischen Stosses des Lab 2 . . . . .	44
A.4 Code für die Datenaufbereitung des Schwunges von Julia des Lab 3 . . . . .	46
A.5 Code für die Datenaufbereitung des Schwunges von Romeo des Lab 3 . . . . .	46

# 1 Zusammenfassung

An der ZHAW wird im Physik Engine, kurz PE, physikalische Zusammenhänge von Bewegungsrichtungen und Beschleunigung, sowie potenzieller und kinetischer Energie im Raum und Zeit durch Simulationen auf Körper untersucht. Dabei werden über das Semester hinweg in Kleingruppen zwei Labs bearbeitet.

Diese Untersuchungen werden mit den physikalischen Gesetzen in der Game-Engine Unity simuliert. Dabei werden laufend Daten zu den aktuellen Parameter einzelner Objekte gesammelt und mit diversen Grafiken veranschaulicht. Die aus den zwei Experimenten gewonnenen Erkenntnisse werden in diesem Bericht niedergeschrieben.

## 2 Aufbau des Experiments

### 2.1 Aufbau des Lab 2, „Würfel 1 bewegt sich und stösst“

Für den Aufbau des Experimentes sind zwei Würfel mit den Dimensionen von 1.5m Seitenlänge und dem Gewicht von 2 Kilogramm gegeben. Wie in der Abbildung 1 zu entnehmen ist, wird der linke Würfel Julia und der rechte Romeo benannt. Daneben existiert eine Feder die horizontal an einer Wand befestigt ist. Bei dem gesamten Experiment wird der Reibungswiderstand ignoriert.

Ablauf des Experimentes:

1. Romeo wird mit einer konstanten Kraft (grüner Pfeil in Abbildung 1) auf 2m/s nach rechts beschleunigt.



Abbildung 1: Beschleunigung des Würfels

2. Romeo trifft nun auf die Feder. Dabei soll die Federkonstante (gelber Pfeil in Abbildung 2) so gewählt werden, dass Romeo elastisch zurückprallt ohne die Wand zu berühren.



Abbildung 2: Elastischer Zusammenstoß mit der Feder

3. Nach dem abgefederten Stoss gleitet Romeo zurück in die Richtung aus der er gekommen ist und stösst inelastisch mit Julia zusammen. Über einen FixedJoint haften die Beiden nun zusammen und gleiten mit der übertragener Energie (blaue Pfeile in Abbildung 3) weiter nach links.



Abbildung 3: Inelastischer Zusammenstoß mit dem anderen Würfel

### 2.2 Aufbau des Lab 3, „Beide Würfel schwingen gedämpft“

Nach dem zweiten Experiment gleiten die beiden Würfel noch einige Meter weiter, bevor sie von zwei Kranhaken an je einem 6 Meter langen Seil im Schwerpunkt aufgefangen werden. Dadurch schwingen Romeo und Julia hin und her. Während dieser Pendelbewegung verlieren sie kontinuierlich Impuls aufgrund des Luftwiderstands, bis sie schließlich zum Stillstand kommen.

Der gesamte Versuchsaufbau ist in Abbildung 3 dargestellt. Es sollte jedoch beachtet werden, dass die Seile erst während des Experiments sichtbar werden. Diese sind pink markiert und in Abbildung 12 zu sehen.

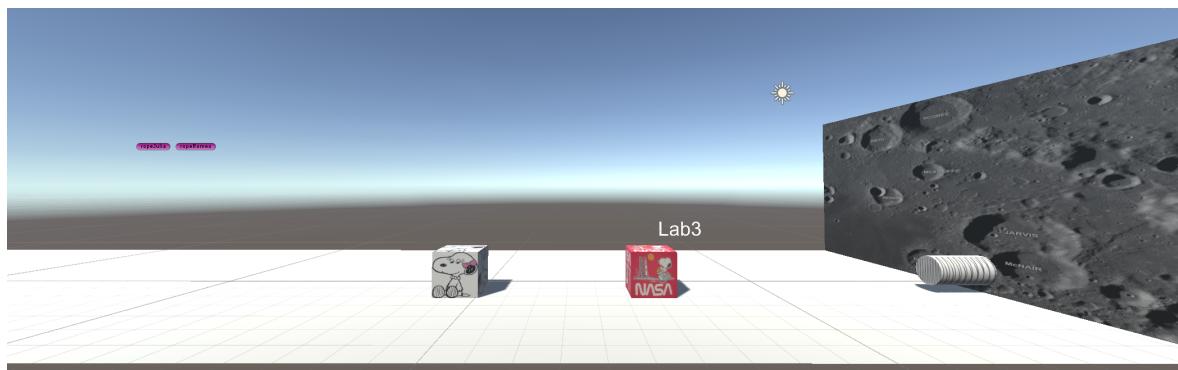


Abbildung 4: Schwingung des Würfels

### 3 Physikalische Beschreibung der einzelnen Vorgänge

In diesem Kapitel werden die physikalischen Vorgänge des Versuches beschrieben. Die gegebenen Massen sind:

- Gewicht[m] = 2kg
- Velocity[v] = 2m/s
- Würfelseite = 1.5m

#### 3.1 Lab 2: Würfel bewegt sich und stösst

Es werden drei Vorgänge beschrieben, die Beschleunigung durch die konstante Kraft, einen elastischen Stoß und einen inelastischen Stoß. Ein Würfel, namens Romeo, wird durch die konstante Kraft beschleunigt, bis maximal eine Geschwindigkeit von 2m/s erreicht wird. Romeo trifft auf eine Feder zu, die an einer Wand befestigt ist. Dabei geschieht ein elastischer Stoß und der Würfel gleitet wieder zurück und stößt dabei einen zweiten Würfel, Julia, diesmal passiert der Stoß inelastisch. Sämtliche Vorgänge erfolgen ohne Reibungskräfte.

##### 3.1.1 Konstante Kraft

Um die konstante Kraft zu berechnen nehmen wir die gewünschte Geschwindigkeit und berechnen damit die Beschleunigung, weil die Kraft sowohl von der Masse wie auch der Beschleunigung abhängt und gegeben ist durch die Formel[1]

$$F = m * a$$

Um dieses Anfangswertproblems zu lösen leiten wir die Geschwindigkeit ab[1]:

$$\begin{aligned} \dot{v} &= a \\ 2m * s^{-1} &\rightarrow -2m * s^{-2} \rightarrow a = [\frac{2m}{s^2}] \end{aligned}$$

Die Zeit, die gebraucht wird um den Würfel zu beschleunigen, wird durch folgende Formel beschrieben [1]:

$$v = a * t \rightarrow t = \frac{v}{a} \rightarrow \frac{2m/s}{2m/s^2} = 1s$$

Somit können wir nun die Kraft ausrechnen:

$$F = 2kg * \frac{2m}{s^2} \Rightarrow \frac{4kg*m}{s^2} = 4N$$

4N werden deshalb als konstante Kraft angewendet, damit auch die gewünschte Geschwindigkeit erreicht wird, danach wird keine Kraft mehr hinzugefügt und Romeo gleitet auf die Feder zu.

### 3.1.2 Elastischer Stoss

Beim elastischen Stoss ist die kinetische Energie vom Stosspartner vor und nach der Kollision gleich [1]. Gemäss Auftrag wird die Federlänge und Federkonstante so dimensioniert, dass der Würfel nicht auf die Wand trifft. Die kinetische Energie des Würfels wird mit folgender Formel berechnet [1]:

$$E_{kin_{Romeo}} = \frac{1}{2} * m * v^2$$

Setzt man die Massen von diesem Projekt ein erhält man:

$$\frac{1}{2} * 2kg * (\frac{2m}{s})^2 = 4J$$

Während des Stosses wird die kinetische Energie auf die Feder übertragen. Die Feder speichert diese Energie in Form von potentieller Energie, da sie zusammengedrückt wird. Sobald sie Romeo zurückstößt, wird diese Energie in eine kinetische zurückgewandelt.

Um die Federkonstante zu berechnen, nehmen wir die Tatsache der Energieerhaltung zu Nutze und setzen die ausgerechnete kinetische Energie gleich mit der potentiellen Energie der Feder.

Die Formel für die potentielle Energie der Feder lautet[1]:

$$E_{pot_{Feder}} = \frac{1}{2} * k * x^2$$

Die Gleichsetzung der Energien, sieht folgendermassen aus[1]:

$$E_{kin_{Romeo}} = E_{pot_{Feder}}$$

$$\frac{1}{2} * m * v^2 = \frac{1}{2} * k * x^2$$

Diese Gleichung stellen wir um und lösen nach der Federkonstante k auf:

$$k = \frac{m*v^2}{x^2}$$

Mit den eingesetzten Massen und die gewählte maximale Auslenkung erhalten wir:

$$\frac{2kg*(2m/s)^2}{(1.7m)^2} = 2.77N/m$$

Jetzt wo wir die Federkonstante und Länge haben, können wir einen langsamen Stoss gewährleisten.

### 3.1.3 Inelastischer Stoss

Beim vollständigen inelastischen Stoss, werden beide Stosspartner nach der Kollision verbunden sein und dieselbe Geschwindigkeit haben [1]. Die Formeln, die wir für diesen Vorgang brauchen sind, die der Impulse der beiden Körper:

$$Impuls_{Romeo} = m_{Romeo} * v_{Romeo} = 2kg * 2m/s = 4Ns$$

$$Impuls_{Julia} = m_{Julia} * v_{Julia} = 2kg * 0m/s = 0Ns$$

Bei diesem Vorgang wird ein Teil des Impulses von Romeo auf Julia übertragen. Der Gesamtimpuls bleibt vor und nach dem Stoss erhalten und wird durch den Impulserhaltungssatz beschrieben:

$$Gesamtimpuls = Impuls_{Romeo} + Impuls_{Julia}$$

$$m_{Romeo} * v_{Romeo} + m_{Julia} * v_{Julia} = (m_{Romeo} + m_{Julia}) * v_{Ende}$$

$$Gesamtimpuls = 4Ns + 0Ns = 4Ns$$

Die Endgeschwindigkeit ist die Geschwindigkeit, die beide Körper gemeinsam haben nach dem Stoss. Bei diesem Vorgang wird ein Teil des Impulses von Romeo auf Julia übertragen. Der Gesamtimpuls bleibt erhalten vor und nach dem Stoss und wird durch den Impulserhaltungssatz beschrieben:[1]:

$$v_{Ende} = \frac{Gesamtimpuls}{(m_{Romeo} + m_{Julia})}$$

$$v_{Ende} = \frac{4Ns}{2kg + 2kg} = 1m/s$$

Die Relation zwischen der kinetischen Energie und des Impulses, können wir folgendermassen herleiten

$$E_{kin} = \frac{1}{2} * m * v^2 = \frac{(mv)^2}{2m} = \frac{p^2}{2m}$$

Wenden wir dies nach dem Stoss an, sehen wir, dass die kinetische Energie geringer wird:

$$E_{kin\ Ende} = \frac{p^2}{2(m_{Romeo} + m_{Julia})}$$

## 3.2 Teil 3: Beide Würfel schwingen gedämpft

Bei diesem Versuch spielen gleich mehrere Kräfte eine Rolle um eine gedämpfte Schwingung zu verursachen. Zum einen die Gewichtskraft und zum anderen die Reibungskraft der turbulenten viskosen Luftreibung, die zusammen die Zentripetalkraft ergeben. Neben den bestehenden bekannten Größen aus Versuch Lab 2 kommen folgende Größen hinzu:

- R = Seillänge 6m
- $c_w = 1.1$
- $\rho_{Luft} = 1.2kg/m^3$
- g =  $9.81m/s^2$
- A = Stirnfläche Würfel =  $1.5 * 1.5 = 2.25$

In den folgenden Beispielrechnungen werden die Zahlen genommen für den Würfel Romeo, wobei dasselbe gilt für den Würfel Julia. Hinzukommmt, dass in C# die Cosinus, Sinus und Arctangens im Bogenmaß berechnet werden, also werden sie in diesem Kapitel auch so aufgeführt.

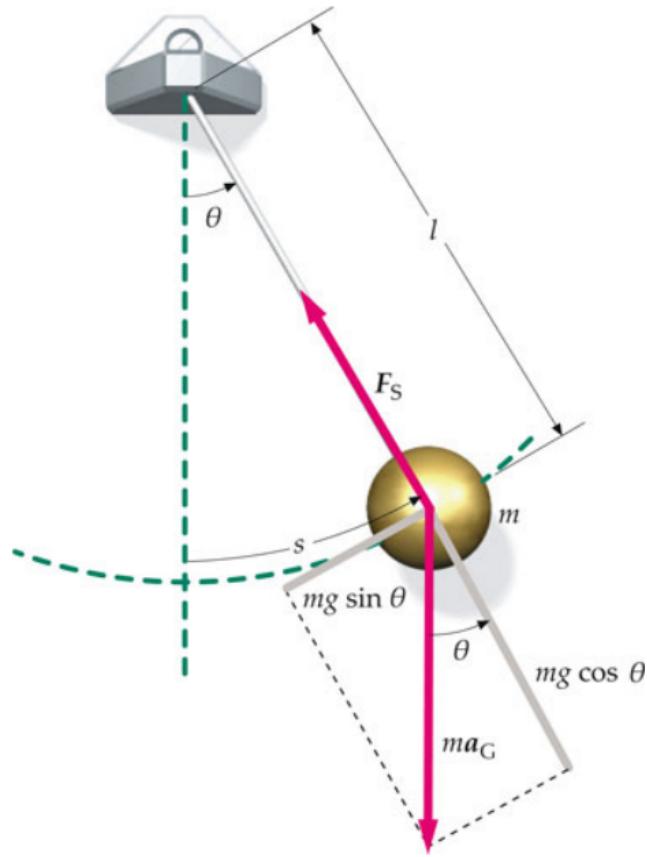


Abbildung 5: Kräfte auf eine Pendelmasse [1]

### 3.2.1 Radialer Anteil der Gewichtskraft

Die Gewichtskraft ist die Kraft, die durch die Wirkung der Gravitation, auf den Körper wirkt. In einer Schwingung wird der Winkel in der Formelberechnung mitberücksichtigt, da die Punktmasse tangential beschleunigt wird.[1]

$$F_g = m * g * \cos(\alpha)$$

**Aus dem Versuch:**

$$F_g = 2 * 9.81 * \cos(0) = 19.62N$$

### 3.2.2 Zentripetalkraft

Diese Kraft ist dem radialen Vektor entgegengesetzt, vorausgesetzt dieser Vektor zeigt vom Kreismittelpunkt nach aussen. Die positive Richtung zeigt demnach nach innen. Diese Kraft ist die Komponente vom resultierenden Kraft, die senkrecht zur Schwingung steht und zum Mittelpunkt weist. In diesem Fall wird sie durch die Gravitationskraft und die Reibungskraft hervorgerufen und ist somit keine eigenständige neu Kraft. Die Zentripetalkraft hat die allgemeine Formel [1]:

$$F_z = m * \frac{v^2}{R}$$

**Aus dem Versuch:**

$$F_z = 2 * \frac{1.004936^2}{6} = 0.17N$$

Wie bereits erwähnt, hängt die Zentripetalkraft von mehreren Komponenten ab, sowohl in horizontaler als auch in vertikaler Richtung. Um dies zu berücksichtigen, werden die horizontalen und vertikalen Komponenten der Ortsvektoren zur Zentripetalkraft mit den entsprechenden Komponenten der Gewichtskraft addiert. Anschliessend werden sie gemäss den Regeln der Trigonometrie mit dem Sinus oder Cosinus des Winkels multipliziert. Die folgende Skizze veranschaulicht diesen Sachverhalt:

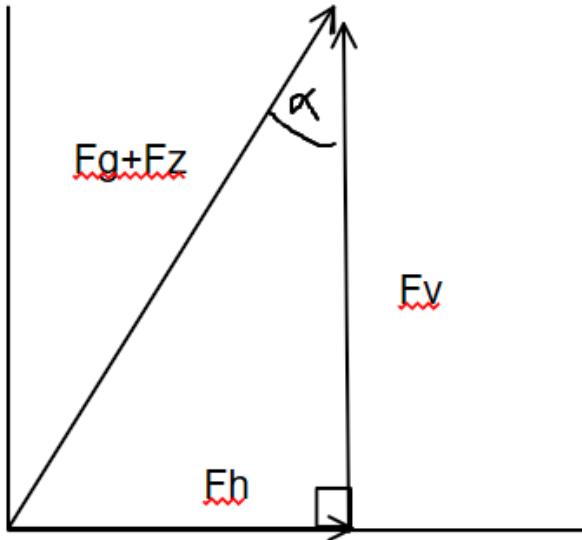


Abbildung 6: Kräftediagramm Skizze

$$H = F_z + F_g$$

$$\arctan(\alpha) = \frac{\text{Aufhängepunktseil.x} - \text{PositionWürfel.x}}{\text{Aufhängepunktseil.y} - \text{PositionWürfel.y}}$$

$$\frac{A}{H} = \cos(\alpha) \rightarrow A = (F_z + F_g) * \cos(\alpha) = F_v$$

$$\frac{G}{H} = \sin(\alpha) \rightarrow G = (F_z + F_g) * \sin(\alpha) = F_h$$

$$\text{Kraftvektor : } (F_h, F_v, 0)$$

#### Aus dem Versuch:

Aufhängepunktseil so gewählt dass Betrag des Vektors 6 ergibt, und bei Winkel 0 also steht der Würfel direkt unter dem Aufhängepunkt, so ergibt es für x und z Koordinaten des Differenzvektors null, und zu y vom Aufhängepunkt 6 dazu addieren.

Position Würfel =  $(-20.01, 5.04, -1.5)$

Aufhängepunkt =  $(-20.01, 11.04, -1.5)$

Differenzvektor =  $(0, 6, 0)$

$$H = 0.17N + 19.62N = 19.77$$

$$\arctan(0) = \frac{0}{0} = 0$$

$$\frac{A}{H} = \cos(0) \rightarrow A = (0.17N + 19.62N) * \cos(0) = 19.77$$

$$\frac{G}{H} = \sin(0) \rightarrow G = (0.17N + 19.62N) * \sin(0) = 0$$

Kraftvektor :  $(19.77, 0, 0)$

### 3.2.3 Reibungskraft

Gemäss der Aufgabenstellung wirken auf die Würfel noch turbulente viskose Reibungskräfte. Diese Reibungskräfte führen zur Dämpfung des schwingenden Systems, da sie dem System mechanische Energie entziehen, die in Form von Wärmeenergie dissipiert wird. Die Berechnung dieser Kraft erfolgt mithilfe der folgenden Formel:

$$F_r = -0.5 * A * \rho_{Luft} * c_w * v^2 * \vec{e}_v$$

Aus dem Versuch:

$$F_r = -0.5 * 2.25 * 1.2 \text{kg/m}^3 * 1.1 * 1.004936^2 * (-1.00, 0.00, 0.00) = (1.5, 0, 0)$$

### 3.2.4 Resultierende Kraft

Zu allerletzt wird die vorher berechnete Kraft tangential zur Richtung Geschwindigkeit  $\vec{e}_v$  dazu addiert und ergibt die resultierende Kraft, die auf die Würfel gesamthaft wirkt.

$$(F_h, F_v, 0) + F_r$$

Aus dem Versuch:

$$(19.77, 0, 0 + (1.5, 0, 0)) = (21.27N, 0, 0)$$

## 4 Beschreibung der Implementierung inklusive Screenshots aus Unity

Der Programcode für Unity wird in C# geschrieben und im folgenden Kapitel wird auf die Implementation in Unity eingegangen.

### 4.1 Lab 2: Würfel bewegt sich und stösst

Alle Kräfte und Berechnungen befinden sich im Code CubeController.cs, welches im Anhang ersichtlich ist. Zur Kontrolle der Werte und Grafik Erstellung werden zwei verschiedene CSV Dateien erstellt, eine für den elastischen Stoss relevanten Werte und eine für den inelastischen Stoss.

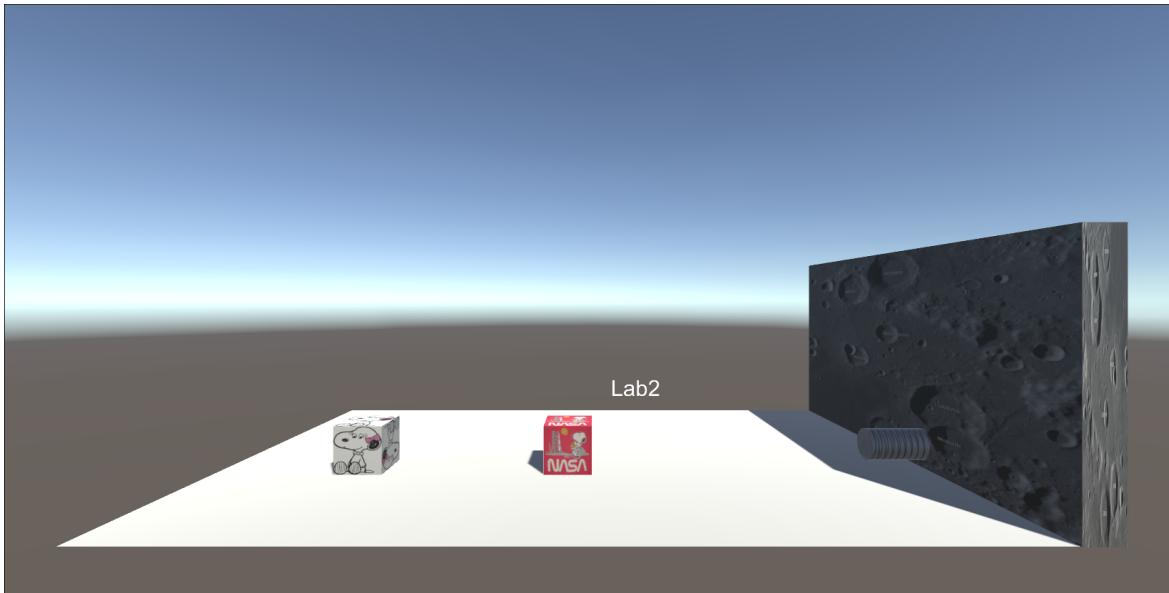


Abbildung 7: Experiment Übersicht

Folgend sind einige wichtige Eigenschaften der Lab relevanten Objekte in Unity aufgelistet:

- Julia:

Für die Seitenlänge ist die Scale auf 1.5 angepasst, da in Unity beim Würfel eine Seitenlänge von 1 gilt. Wichtig ist nicht zu vergessen, das Material auf reibungslos zu ändern, sonst gleiten die Würfel nicht korrekt.

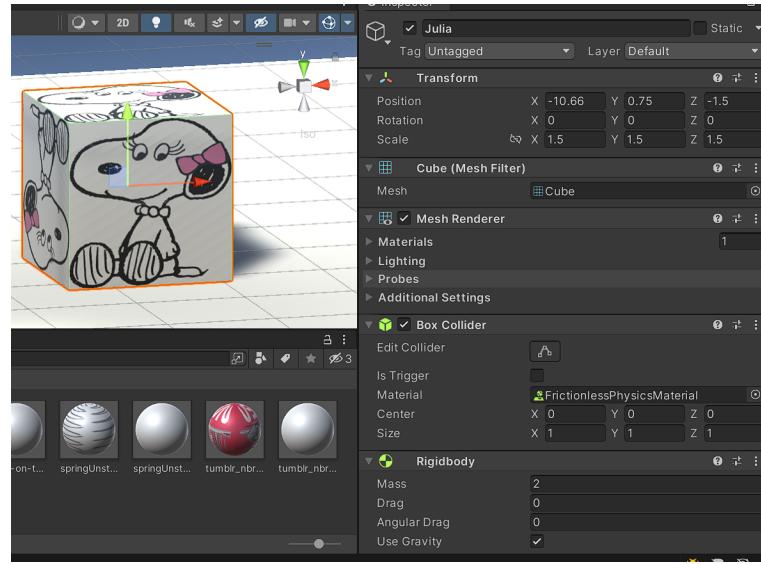


Abbildung 8: Einstellung Julia

- Romeo:

Die Eigenschaften sind ausser der Position und Farbe gleich wie bei Julia. Romeo besitzt zudem Variabel, über welcher gewisse Parameter an den ihm angehängten Code übergeben werden kann.

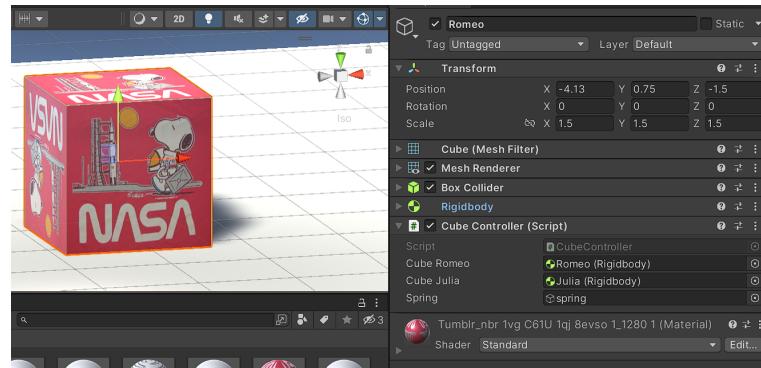


Abbildung 9: Einstellung Romeo

- Spring:

Wie in Abbildung 9 zu sehen ist die Feder nur als GameObject und nicht als Rigidbody im Code angegeben. Die Ausrichtung wurde entlang der Y-Achse belassen und um 90 Grad rotiert damit der Zylinder liegend erscheint.

- Mesh: Cylinder
- Collider direction: Y-Axis

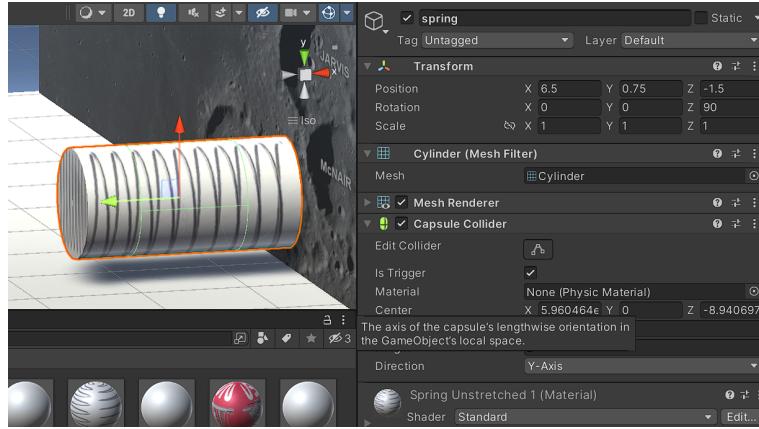


Abbildung 10: Einstellung Feder

- Plane
  - Collider Material: FrictionlessPhysicsMaterial

Da es sich beim CubeController um ein Unity Code handelt, wird von der Klasse Monobeviour geerbt, damit Methoden wie FixedUpdate oder OnCollisionEnter benutzt werden kann. Die im vorhinein berechnete konstante Kraft 4, sowie Beschleunigungszeit 1 wird im Code als Konstanten am Anfang deklariert. Für die Federauslenkung wird 1.7 gewählt, da die Feder eine Länge von 2 hat und vorher gestoppt werden muss bevor Romeo auf die Wand auftrefft. In der Start Methode wird aus den gegebenen Werten die Federkonstante berechnet.

```

1 //Maximale Auslenkung gerechnet anhand der linken seite des Feders
2 springMaxDeviation = spring.transform.position.x - springLength / 2;
```

Es wird auch die Position ermittelt, welche Romeo zum ersten Mal besitzt, wenn er auf die Feder auftrefft (springMaxDeviation) wie in Abbildung 11 rot markiert ist.

```

1 // Energieerhaltungsgesetz kinEnergie = PotEnergie : 1/2*m*v^2 = 1/2k * x^2
2 springConstant = (float)((Romeo.mass * Math.Pow(2.0, 2)) /
    (Math.Pow(springContraction, 2.0)));
```

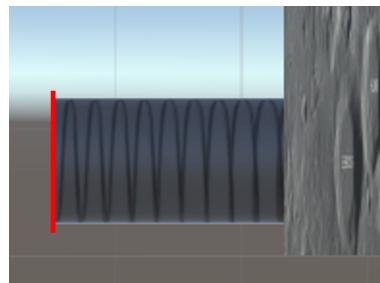


Abbildung 11: Feder mit markierter Berührungs punkt

Danach werden die Timeseries Listen deklariert, für die CSV Dateien. Das Hinzufügen der Werte passiert kontinuierlich in der Methode FixedUpdate.

In FixedUpdate gibt es zwei If-Bedingungen. Die erste ist zum Hinzufügen der konstanten Kraft mit der Methode .AddForce bis die Beschleunigungszeit vorüber ist. Die zweite If-Bedingung ist zur

Überprüfung, ob Romeo die Feder berührt. Dafür muss die Position der rechten Kante von Romeo berechnet werden und mit der springMaxDeviation verglichen werden. Für den inelastischen Stoss wird die Komponente FixedJoint in der Methode OnCollision implementiert. So bleiben Romeo und Julia nach ihrer Kollision zusammen und gleiten gemeinsam in den Sonnenuntergang.

## 4.2 Lab 3: Beide Würfel schwingen gedämpft

Für die dritte Aufgabe sind neben dem CubeController.cs noch weitere Scripte dazugekommen. Diese sind:

- LineCrontroller.cs
- LineJulia.cs
- SwingJulia.cs
- SwingRomeo.cs

Zur Kontrolle der Werte und Grafik Erstellung werden zwei verschiedene CSV Dateien erstellt, eine für den Schwung von Julia und eine für den Schwung von Romeo. Der gesamte Labaufbau in Unity ist in der Abbildung 12 zur Laufzeit visuell dargestellt.

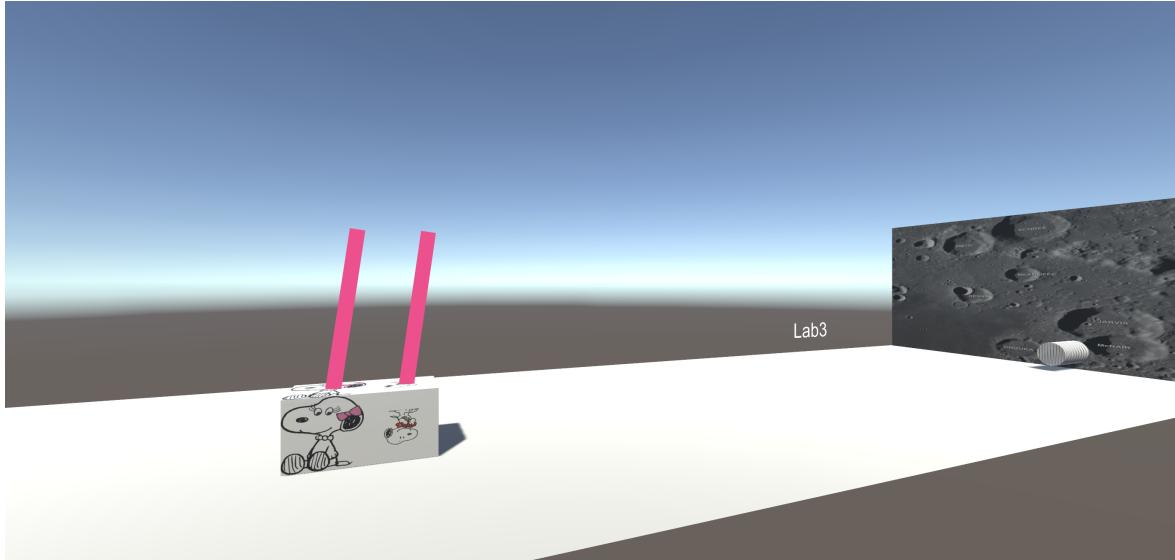


Abbildung 12: Experiment Übersicht

Folgend sind die neuen wichtigen Eigenschaften des Lab relevanten Objekte in Unity aufgelistet:

- Seile: Die Seile die in der Abbildung 12 in Pink ersichtlich sind, werden von den Würfel zu den vordefinierten Punkten wie in Abbildung 13 ersichtlich sind, gezeichnet.

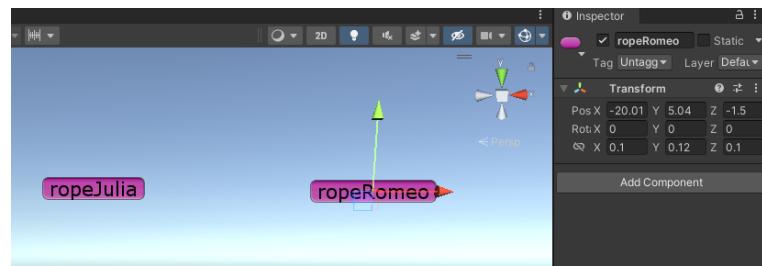


Abbildung 13: Experiment Übersicht

## 5 Resultate mit grafischer Darstellung

### 5.1 Lab 2

Während dem Durchlauf des Experimentes des Lab 2, werden diverse Daten physikalische Vorgänge gesammelt. Diese Daten umfassen Ort und Geschwindigkeit, sowie kinetische und potentielle Energie. Nachfolgend werden alle Daten als Funktion der Zeit in der Abbildung 14 bis Abbildung 18 aufgegliedert.

In Abbildung 14 ist deutlich zu erkennen, wie Romeo während der Beschleunigungsphase in den ersten Sekunden an Impuls gewinnt. Nach fünf Sekunden Gleitphase stösst Romeo auf die Feder, wodurch er abgebremst wird und Impuls verliert, bis er schliesslich bei null ankommt. Der Impuls wird jedoch in der gespannten Feder gespeichert und beim Entspannen der Feder wieder auf den Würfel übertragen. Dadurch hat der Würfel nach der Beschleunigungsphase wieder den gleichen Impuls wie zuvor. Da der Gesamtimpuls erhalten bleibt, haben Romeo und Julia nun beide ein Impuls von 2 Ns. In Abbildung 15 sieht man die addierten Impulse und wie dies nach der Kollision in der Tat gleich bleibt.

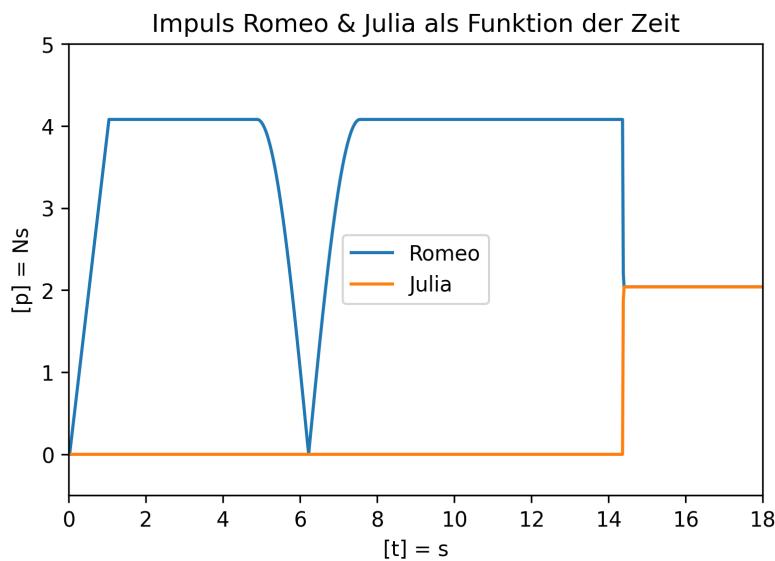


Abbildung 14: Impuls Romeo & Julia als Funktion der Zeit

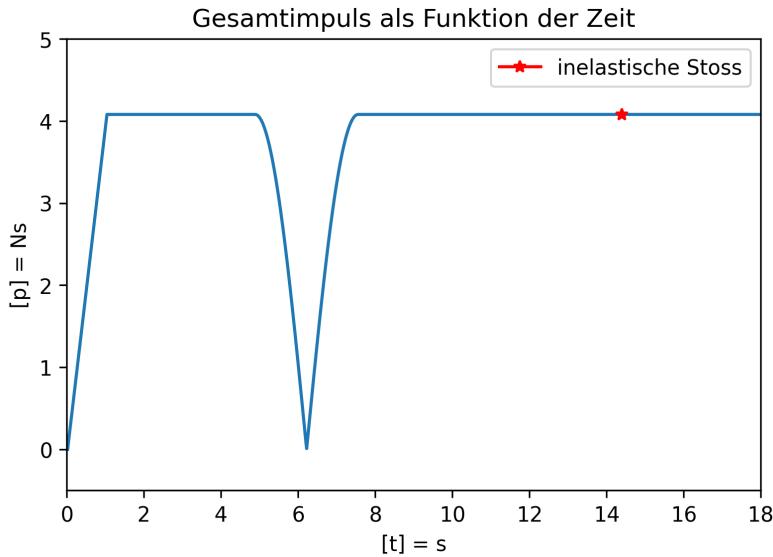


Abbildung 15: Gesamtmpuls als Funktion der Zeit

Im Ort-Zeit Diagramm in der Abbildung 16 ist ersichtlich, dass Romeo bis Sekunde 5 sich bewegt und danach auf die Feder auftritt. Romeo bewegt sich dann gleichmäig in die entgegengesetzte Richtung und nach 14 Sekunde ist zu sehen wie beide Würfel dann zusammen sich bewegen.

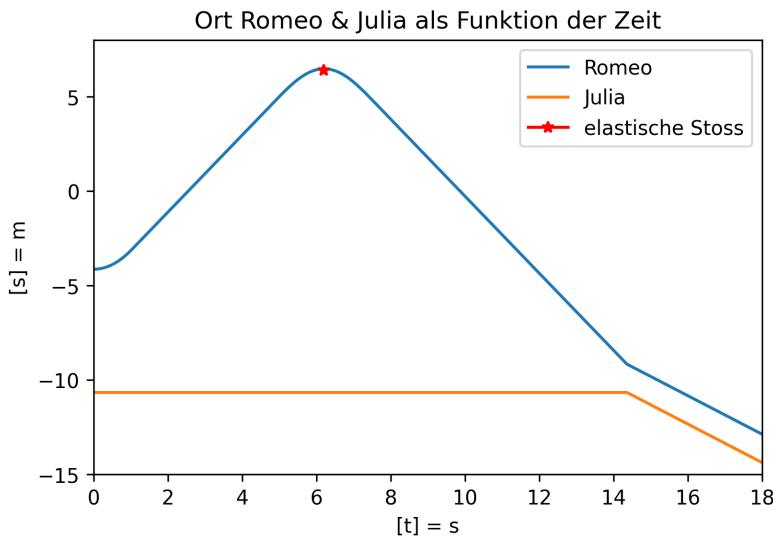


Abbildung 16: Ort Romeo & Julia als Funktion der Zeit

Gemäss Berechnung erreicht Romeo nach einer Sekunde die maximale Geschwindigkeit von 2 m/s und bewegt sich weiter mit dieser Geschwindigkeit bis es kurz vor Sekunde 6 auf die Feder trifft. Dann verändert sich wegen den Rückstoss die Richtung der Geschwindigkeit. Beim inelastischen Stoss kleben Romeo und Julia zusammen und fahren mit der Schwerpunktgeschwindigkeit vEnde von 1 m/s, wie im Kapitel physikalische Beschreibung berechnet, weiter. In der Abbildung 18 sieht man, dass dies in unseren Versuch auch übereinstimmt.

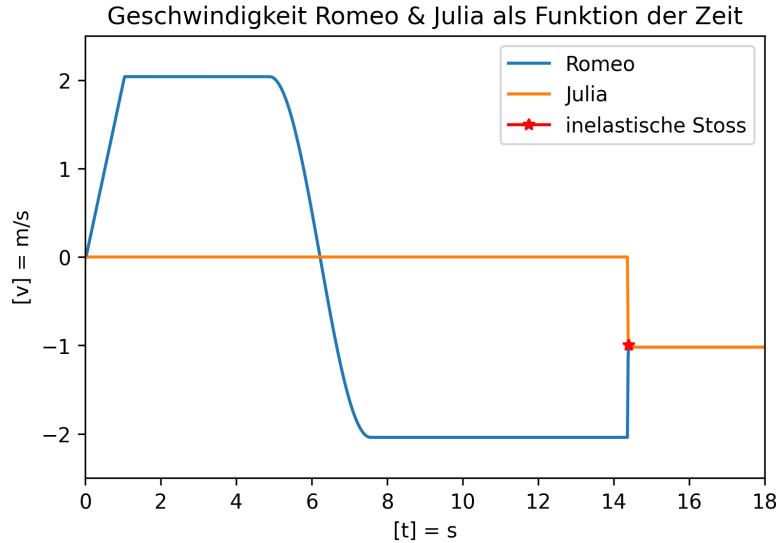


Abbildung 17: Geschwindigkeit Romeo & Julia als Funktion der Zeit

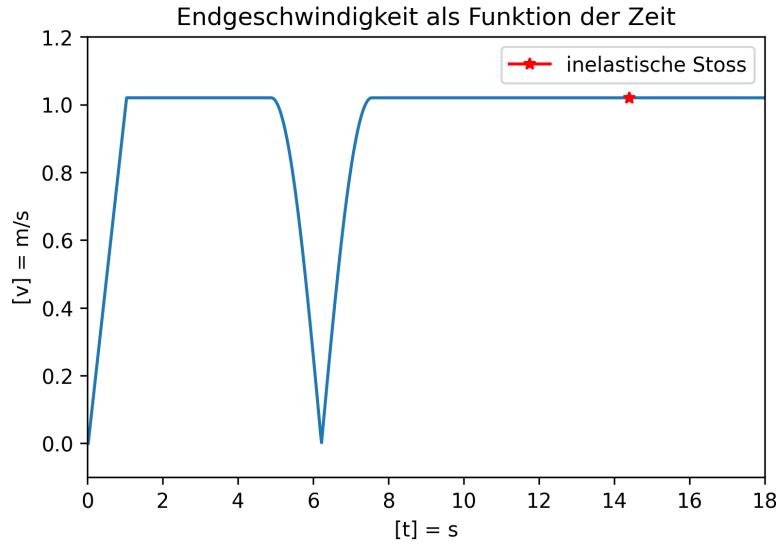


Abbildung 18: Endgeschwindigkeit als Funktion der Zeit

## 5.2 Lab 3

Die beiden gleitenden Würfel Romeo und Julia sind durch Seile festgehalten. Der Gesamtimpuls, den sie besitzen, wird in eine schwingende Bewegung umgewandelt. Während ihrer Pendelbewegung verlieren sie jedoch nach und nach Energie aufgrund des Luftwiderstands. Dadurch nimmt ihre Auslenkung, wie in Abbildung 19 zu sehen ist, kontinuierlich ab. Jedoch sind die beiden Graphlinien von Romeo und Julia nicht kongruent, wir gehen davon aus, dass dies auf den Fixjoint von der Unity-Engine zurückzuführen ist. Dies wird auch in Abbildung 20 deutlich, wo die Position der beiden Würfel sich immer weniger weit vom Mittelpunkt: -20m bei Romeo und -21.5m bei Julia entfernen.

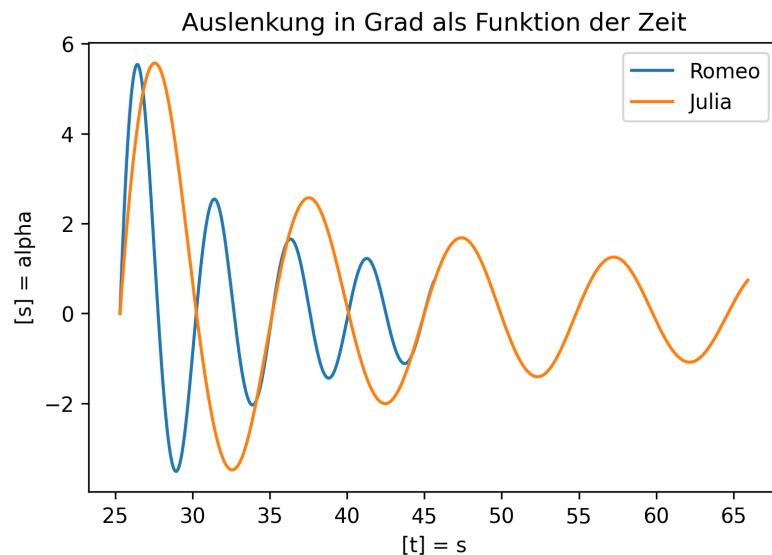


Abbildung 19: Auslenkung in Grad als Funktion der Zeit

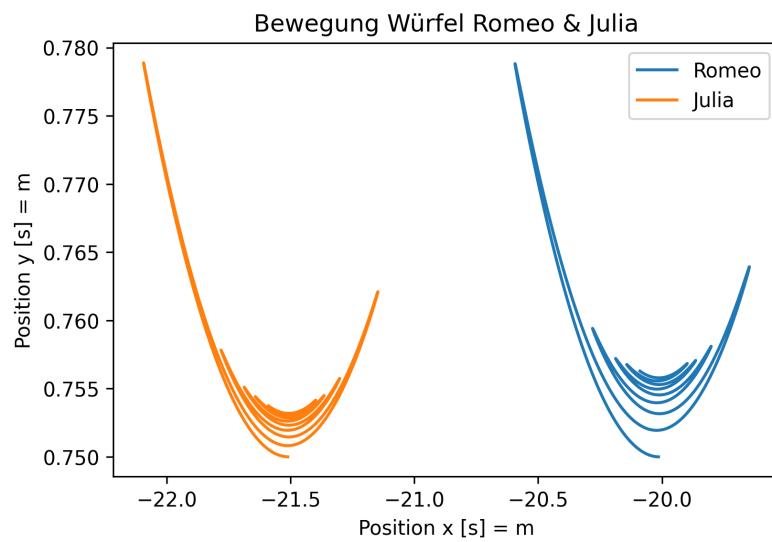


Abbildung 20: Romeo & Julia Ortsdiagramm

## 6 Rückblick und Lehren aus dem Versuch

Uns wurde beim Programmieren mit Unity bewusst, wie wichtig es ist Gleitkommazahlen zu berücksichtigen, da Unity iterativ arbeitet. Dadurch entstanden Ungenauigkeiten, die das Experiment nicht korrekt durchlaufen liessen. Darüber hinaus war es erforderlich, bei den Labs verschiedene Aspekte wie Speicherung, Übergabe und Schleifen in der Programmierung zu berücksichtigen.

Besonders faszinierend ist die Art und Weise, wie Unity trotz seiner Funktion als Game-Engine mit den physikalischen Gesetzen umgeht. Obwohl es keine dedizierte Physik-Engine ist, erweist sich die Integration der physikalischen Aspekte als bemerkenswert.

Darüber hinaus hat es uns grossen Spass bereitet, uns sowohl mit der Physik als auch mit dem Programmieren intensiv auseinanderzusetzen. Die Kombination dieser beiden Bereiche war äusserst spannend und ermöglichte uns ein tieferes Verständnis der Zusammenhänge.

Insgesamt war dieses Projekt im Verlauf des Semesters herausfordernd, aber machbar. Die Idee mit den Bonuspunkten, d.h. eine maximale Note von 5.5, wenn Lab 2 abgeschlossen ist, und eine Note von 6.0, wenn alle 3 Labs erledigt wurden, finden wir sehr gut. Dies würde sich auch für zukünftige Studierende anbieten und motivierend wirken.

# A Anhang

## A.1 Code für das Lab 2 und 3

Nachfolgend sind die einzelnen Codesfile aufgelistet, die für den gesamten Labaufbau benötigt wurden.

Listing 1: CubeController.cs

```
1 using System;
2 using System.Collections.Generic;
3 using System.IO;
4 using Unity.VisualScripting;
5 using UnityEditor.SceneManagement;
6 using UnityEngine;
7
8
9 public class CubeController : MonoBehaviour
10 {
11     public Rigidbody Romeo;
12     public Rigidbody Julia;
13     public GameObject spring;
14     public GameObject ropeRomeo;
15     public GameObject ropeJulia;
16
17     // public Vector3 ropeAnchor;
18
19     private float currentTimestep; // s
20     private float cubeJuliaTimestep;
21
22     private List<List<float>> timeSeriesElasticCollision;
23     private List<List<float>> timeSeriesInelasticCollision;
24
25
26     private bool rotationTriggered = false;
27
28
29
30     private string filePath;
31     private byte[] fileData;
32     float springPotentialEnergy = 0f;
33     float cubeRomeoKinetic = 0f;
34     float cubeRomeoImpulse = 0f;
35     float cubeJuliaImpulse = 0f;
36     float GesamtImpuls = 0f;
37     float ImpulsCheck = 0f;
38     float forceOnJulia = 0f;
39     float velocityEnd = 0f;
40     float cubeKineticEnd = 0f;
41     float constantForce = 4f;
42     double starttime = 0;
43     double accelerationTime = 1.0;
44     float springConstant = 0f;
45     float springMaxDeviation = 0f;
46     float springContraction = 1.7f;
47     float springLength = 0f;
48     float R = 6f; //Radius Rope
49     //public float rotationTrigger;
50
51     float g = 9.81f; //Gravity
52
53     float alphaRomeo = 0f; //Angle between crane and rope
54     float alphaJulia = 0f; //Angle between crane and rope
```

```

55
56     float cCube = 1.1f;
57     float constantAirFriction = 1.2f;
58
59     float areaRomeo = 2.25f;
60     float areaJulia = 2.25f;
61
62
63
64
65     // Start is called before the first frame update
66     void Start()
67     {
68
69         Romeo = GetComponent<Rigidbody>();
70         Julia = GetComponent<Rigidbody>();
71
72         starttime = Time.fixedTimeAsDouble;
73
74
75         timeSeriesElasticCollision = new List<List<float>>();
76         timeSeriesInelasticCollision = new List<List<float>>();
77
78
79
80         springLength = spring.GetComponent<MeshFilter>().mesh.bounds.size.y *
81             spring.transform.localScale.y;
82
83         // Maximale Auslenkung gerechnet anhand der linken seite des Feders
84         springMaxDeviation = spring.transform.position.x - springLength / 2;
85         // Energieerhaltungsgesetz kinEnergie = PotEnergie :  $1/2*m*v^2 = 1/2k * x^2$ 
86         springConstant = (float)((Romeo.mass * Math.Pow(2.0, 2)) /
87             (Math.Pow(springContraction, 2.0)));
88
89     }
90
91     // Update is called once per frame
92     void Update()
93     {
94     }
95     // FixedUpdate can be called multiple times per frame
96     void FixedUpdate()
97     {
98         double currentTime = Time.fixedTimeAsDouble-starttime;
99
100        if (accelarationTime >= currentTime)
101        {
102            constantForce = 4f;
103            Romeo.AddForce(new Vector3(constantForce, 0f, 0f));
104        }
105
106        //  $1/2*m*v^2$ 
107        cubeRomeoKinetic = Math.Abs((float)(0.5 * Romeo.mass * Math.Pow(Romeo.velocity.x,
108                                     2.0)));
109
110        float collisionPosition = Romeo.transform.position.x + Romeo.transform.localScale.x /
111                                     2;
112
113        if (collisionPosition >= springMaxDeviation)

```

```

113     {
114         float springForceX = (collisionPosition - springMaxDeviation) * -springConstant;
115         springPotentialEnergy =(float)(0.5 * springConstant * Math.Pow(collisionPosition -
116             springMaxDeviation, 2.0));
117         Romeo.AddForce(new Vector3(springForceX, 0f, 0f));
118         ChangeCubeTexture();
119         currentTimeStep += Time.deltaTime;
120         timeSeriesElasticCollision.Add(new List<float>() { currentTimeStep,
121             Romeo.position.x, Romeo.velocity.x, springPotentialEnergy, cubeRomeoKinetic,
122             springForceX });
123     }
124
125     // 1/2*m*v^2
126     cubeRomeoKinetic = Math.Abs((float)(0.5 * Romeo.mass * Math.Pow(Romeo.velocity.x,
127         2.0)));
128     cubeRomeoImpulse = Math.Abs(Romeo.mass * Romeo.velocity.x);
129     cubeJuliaImpulse = Math.Abs(Julia.mass * Julia.velocity.x);
130     GesamtImpluls = cubeJuliaImpulse + cubeRomeoImpulse;
131     velocityEnd = (cubeRomeoImpulse + cubeJuliaImpulse) / (Romeo.mass + Julia.mass);
132     ImpulsCheck = (Romeo.mass + Julia.mass) * velocityEnd;
133     cubeKineticEnd = Math.Abs((float)(0.5 * (Romeo.mass + Julia.mass) *
134         Math.Pow(velocityEnd, 2.0)));
135     forceOnJulia = Math.Abs(Julia.mass * velocityEnd - Julia.velocity.x);
136
137     cubeJuliaTimeStep += Time.deltaTime;
138     timeSeriesInelasticCollision.Add(new List<float>() { cubeJuliaTimeStep,
139         Romeo.position.x, Romeo.velocity.x, Romeo.mass, cubeRomeoImpulse,
140         cubeRomeoKinetic, Julia.position.x, Julia.velocity.x, Julia.mass,
141         cubeJuliaImpulse, velocityEnd, cubeKineticEnd, forceOnJulia, GesamtImpluls,
142         ImpulsCheck });
143
144
145     }
146     void OnApplicationQuit()
147     {
148         WriteElasticTimeSeriesToCsv();
149         WriteInelasticTimeSeriesToCsv();
150
151     }
152     void WriteElasticTimeSeriesToCsv()
153     {
154         using (var streamWriter = new StreamWriter("time_seriesElastic.csv"))
155         {
156             streamWriter.WriteLine("currentTimeStep, cubeRomeo.position.x,
157                 cubeRomeo.velocity.x, springPotentialEnergy, cubeRomeoKinetic, springForceX");
158
159             foreach (List<float> timeStep in timeSeriesElasticCollision)
160             {
161                 streamWriter.WriteLine(string.Join(", ", timeStep));
162                 streamWriter.Flush();
163             }
164         }
165     }
166     void WriteInelasticTimeSeriesToCsv()
167     {
168         using (var streamWriter = new StreamWriter("time_seriesInelastic.csv"))

```

```

165     {
166         streamWriter.WriteLine("cubeJuliaTimeStep, cubeRomeo.position.x,
167             cubeRomeo.velocity.x,cubeRomeo.mass, cubeRomeoImpulse, cubeRomeoKinetic,
168             cubeJulia.position.x, cubeJulia.velocity.x,cubeJulia.mass, cubeJuliaImpulse,
169             velocityEnd, cubeKineticEnd, forceOnJulia, GesamtImpluls, ImpulsCheck }");
170
171         foreach (List<float> timeStep in timeSeriesInelasticCollision)
172         {
173             streamWriter.WriteLine(string.Join(", ", timeStep));
174             streamWriter.Flush();
175         }
176     }
177
178
179
180     void ChangeCubeTexture()
181     {
182         // the path of the image
183         filePath = "Assets/Images/snoopy-flower-cynthia-t-thomas.jpg";
184         // 1.read the bytes array
185         fileData = File.ReadAllBytes(filePath);
186         // 2.create a texture named tex
187         Texture2D tex = new Texture2D(2, 2);
188         // 3.load inside tx the bytes and use the correct image size
189         tex.LoadImage(fileData);
190         // 4.apply tex to material.mainTexture
191         GetComponent<Renderer>().material.mainTexture = tex;
192     }
193
194     void OnCollisionEnter(Collision collision)
195     {
196         if (collision.rigidbody != Julia)
197         {
198             return;
199         }
200         if (collision.rigidbody == Julia)
201         {
202             FixedJoint joint = gameObject.AddComponent<FixedJoint>();
203             ContactPoint[] contacts = new ContactPoint[collision.contactCount];
204             collision.GetContacts(contacts);
205             ContactPoint contact = contacts[0];
206             joint.anchor = transform.InverseTransformPoint(contact.point);
207             joint.connectedBody =
208                 collision.contacts[0].otherCollider.transform.GetComponent<Rigidbody>();
209
210             // Stops objects from continuing to collide and creating more joints
211             joint.enableCollision = false;
212         }
213     }
214
215
216
217 }
218
219
220
221
222 }
```

---

```

1 using System;
2 using System.Collections.Generic;
3 using System.IO;
4 using Unity.VisualScripting;
5 using UnityEditor.SceneManagement;
6 using UnityEngine;
7
8
9 public class CubeController : MonoBehaviour
10 {
11     public Rigidbody Romeo;
12     public Rigidbody Julia;
13     public GameObject spring;
14     public GameObject ropeRomeo;
15     public GameObject ropeJulia;
16
17     // public Vector3 ropeAnchor;
18
19     private float currentTimestep; // s
20     private float cubeJuliaTimestep;
21
22     private List<List<float>> timeSeriesElasticCollision;
23     private List<List<float>> timeSeriesInelasticCollision;
24
25
26     private bool rotationTriggered = false;
27
28
29
30     private string filePath;
31     private byte[] fileData;
32     float springPotentialEnergy = 0f;
33     float cubeRomeoKinetic = 0f;
34     float cubeRomeoImpulse = 0f;
35     float cubeJuliaImpulse = 0f;
36     float GesamtImpluls = 0f;
37     float ImpulsCheck = 0f;
38     float forceOnJulia = 0f;
39     float velocityEnd = 0f;
40     float cubeKineticEnd = 0f;
41     float constantForce = 4f;
42     double starttime = 0;
43     double accelerationTime = 1.0;
44     float springConstant = 0f;
45     float springMaxDeviation = 0f;
46     float springContraction = 1.7f;
47     float springLength = 0f;
48     float R = 6f; //Radius Rope
49     //public float rotationTrigger;
50
51     float g = 9.81f; //Gravity
52
53     float alphaRomeo = 0f; //Angle between crane and rope
54     float alphaJulia = 0f; //Angle between crane and rope
55
56     float cCube = 1.1f;
57     float constantAirFriction = 1.2f;
58
59     float areaRomeo = 2.25f;

```

```

60 float areaJulia = 2.25f;
61
62
63
64
65 // Start is called before the first frame update
66 void Start()
67 {
68
69     Romeo = GetComponent<Rigidbody>();
70     Julia = GetComponent<Rigidbody>();
71
72     starttime = Time.fixedTimeAsDouble;
73
74     timeSeriesElasticCollision = new List<List<float>>();
75     timeSeriesInelasticCollision = new List<List<float>>();
76
77
78
79
80     springLength = spring.GetComponent<MeshFilter>().mesh.bounds.size.y *
81         spring.transform.localScale.y;
82
83     //Maximale Auslenkung gerechnet anhand der linken seite des Feders
84     springMaxDeviation = spring.transform.position.x - springLength / 2;
85     // Energieerhaltungsgesetz kinEnergie = PotEnergie :  $1/2*m*v^2 = 1/2k * x^2$ 
86     springConstant = (float)((Romeo.mass * Math.Pow(2.0, 2)) /
87         (Math.Pow(springContraction, 2.0)));
88
89 }
90
91 // Update is called once per frame
92 void Update()
93 {
94 }
95 // FixedUpdate can be called multiple times per frame
96 void FixedUpdate()
97 {
98     double currentTime = Time.fixedTimeAsDouble-starttime;
99
100    if (accelerationTime >= currentTime)
101    {
102        constantForce = 4f;
103        Romeo.AddForce(new Vector3(constantForce, 0f, 0f));
104    }
105
106    //  $1/2*m*v^2$ 
107    cubeRomeoKinetic = Math.Abs((float)(0.5 * Romeo.mass * Math.Pow(Romeo.velocity.x,
108                                2.0)));
109
110    float collisionPosition = Romeo.transform.position.x + Romeo.transform.localScale.x /
111                                2;
112
113    if (collisionPosition >= springMaxDeviation)
114    {
115        float springForceX = (collisionPosition - springMaxDeviation) * -springConstant;
116        springPotentialEnergy =(float)(0.5 * springConstant * Math.Pow(collisionPosition -
117                                         springMaxDeviation, 2.0));
118        Romeo.AddForce(new Vector3(springForceX, 0f, 0f));
119    }
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1090
1091
1092
1093
1094
1095
1095
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1188
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1288
1289
1290
1291
1292
1293
1294
1295
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1390
1391
1392
1393
1394
1394
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1494
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1594
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1694
1695
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1794
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1890
1891
1892
1893
1894
1894
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2089
2090
2091
2092
2093
2094
2095
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2189
2190
2191
2192
2193
2194
2195
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2288
2289
2289
2290
2291
2292
2293
2294
2295
2295
2296
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2388
2389
2389
2390
2391
2392
2393
2394
2395
2395
2396
2397
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2438
2439
2440
2441
```

```

117     ChangeCubeTexture();
118     currentTimeStep += Time.deltaTime;
119     timeSeriesElasticCollision.Add(new List<float>() { currentTimeStep,
120         Romeo.position.x, Romeo.velocity.x, springPotentialEnergy, cubeRomeoKinetic,
121         springForceX });
122     }
123     // 1/2*m*v^2
124     cubeRomeoKinetic = Math.Abs((float)(0.5 * Romeo.mass * Math.Pow(Romeo.velocity.x,
125         2.0)));
126     cubeRomeoImpulse = Math.Abs(Romeo.mass * Romeo.velocity.x);
127     cubeJuliaImpulse = Math.Abs(Julia.mass * Julia.velocity.x);
128     GesamtImpluls = cubeJuliaImpulse + cubeRomeoImpulse;
129     velocityEnd = (cubeRomeoImpulse + cubeJuliaImpulse) / (Romeo.mass + Julia.mass);
130     ImpulsCheck = (Romeo.mass + Julia.mass) * velocityEnd;
131     cubeKineticEnd = Math.Abs((float)(0.5 * (Romeo.mass + Julia.mass) *
132         Math.Pow(velocityEnd, 2.0)));
133     forceOnJulia = Math.Abs(Julia.mass * velocityEnd - Julia.velocity.x);
134
135
136
137     }
138     void OnApplicationQuit()
139     {
140         WriteElasticTimeSeriesToCsv();
141         WriteInelasticTimeSeriesToCsv();
142
143
144
145     }
146     void WriteElasticTimeSeriesToCsv()
147     {
148         using (var streamWriter = new StreamWriter("time_seriesElastic.csv"))
149         {
150             streamWriter.WriteLine("currentTimeStep, cubeRomeo.position.x,
151                 cubeRomeo.velocity.x, springPotentialEnergy, cubeRomeoKinetic, springForceX");
152
153             foreach (List<float> timeStep in timeSeriesElasticCollision)
154             {
155                 streamWriter.WriteLine(string.Join(", ", timeStep));
156                 streamWriter.Flush();
157             }
158         }
159     }
160
161     void WriteInelasticTimeSeriesToCsv()
162     {
163         using (var streamWriter = new StreamWriter("time_seriesInelastic.csv"))
164         {
165             streamWriter.WriteLine("cubeJuliaTimeStep, cubeRomeo.position.x,
166                 cubeRomeo.velocity.x,cubeRomeo.mass, cubeRomeoImpulse, cubeRomeoKinetic,
167                 cubeJulia.position.x, cubeJulia.velocity.x,cubeJulia.mass, cubeJuliaImpulse,
168                 velocityEnd, cubeKineticEnd, forceOnJulia, GesamtImpluls, ImpulsCheck ");
169     }

```

```

167         foreach (List<float> timeStep in timeSeriesInelasticCollision)
168     {
169         streamWriter.WriteLine(string.Join(",", timeStep));
170         streamWriter.Flush();
171     }
172 }
173 }
174 }
175
176
177
178
179
180 void ChangeCubeTexture()
181 {
182     // the path of the image
183     filePath = "Assets/Images/snoopy-flower-cynthia-t-thomas.jpg";
184     // 1.read the bytes array
185     fileData = File.ReadAllBytes(filePath);
186     // 2.create a texture named tex
187     Texture2D tex = new Texture2D(2, 2);
188     // 3.load inside tx the bytes and use the correct image size
189     tex.LoadImage(fileData);
190     // 4.apply tex to material.mainTexture
191     GetComponent<Renderer>().material.mainTexture = tex;
192 }
193
194 void OnCollisionEnter(Collision collision)
195 {
196     if (collision.rigidbody != Julia)
197     {
198         return;
199     }
200     if (collision.rigidbody == Julia)
201     {
202         FixedJoint joint = gameObject.AddComponent<FixedJoint>();
203         ContactPoint[] contacts = new ContactPoint[collision.contactCount];
204         collision.GetContacts(contacts);
205         ContactPoint contact = contacts[0];
206         joint.anchor = transform.InverseTransformPoint(contact.point);
207         joint.connectedBody =
208             collision.contacts[0].otherCollider.transform.GetComponent<Rigidbody>();
209
210         // Stops objects from continuing to collide and creating more joints
211         joint.enableCollision = false;
212     }
213 }
214
215
216
217 }
218
219
220
221
222 }

```

Listing 2: LineJulia.cs

---

```

1 using System.Collections;

```

```

2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class LineJulia : MonoBehaviour
6 {
7     private LineRenderer lineRenderer;
8     private SwingJulia swingJulia;
9     private Rigidbody Julia;
10
11    [SerializeField] private Transform[] cubeTransforms;
12    private Vector3 initialPosition;
13
14    private bool firstRun = true;
15    private bool isSwinging = false;
16
17    private GameObject ropeJulia;
18    // Start is called before the first frame update
19    void Start()
20    {
21        swingJulia = FindObjectOfType(typeof(SwingJulia)) as SwingJulia;
22        lineRenderer = GetComponent<LineRenderer>();
23        Julia = GetComponent<Rigidbody>();
24    }
25
26    // Update is called once per frame
27    void FixedUpdate()
28    {
29        lineRenderer.positionCount = cubeTransforms.Length;
30        for (int i = 0; i < cubeTransforms.Length; i++)
31        {
32            if (cubeTransforms[1].position.x <= -21.51f)
33            {
34                if (firstRun)
35                {
36                    initialPosition = swingJulia.GetPostion();
37                    firstRun = false;
38                    isSwinging = true;
39                }
40                lineRenderer.SetPosition(i, cubeTransforms[i].position);
41            }
42        }
43    }
44
45    if (isSwinging)
46    {
47        swingJulia.MakeSwingJulia(initialPosition);
48    }
49
50    }
51
52    }
53
54 }

```

---

```

1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class LineJulia : MonoBehaviour
6 {
7     private LineRenderer lineRenderer;

```

```

8     private SwingJulia swingJulia;
9     private Rigidbody Julia;
10
11    [SerializeField] private Transform[] cubeTransforms;
12    private Vector3 initialPosition;
13
14    private bool firstRun = true;
15    private bool isSwinging = false;
16
17    private GameObject ropeJulia;
18    // Start is called before the first frame update
19    void Start()
20    {
21        swingJulia = FindObjectOfType(typeof(SwingJulia)) as SwingJulia;
22        lineRenderer = GetComponent<LineRenderer>();
23        Julia = GetComponent<Rigidbody>();
24    }
25
26    // Update is called once per frame
27    void FixedUpdate()
28    {
29        lineRenderer.positionCount = cubeTransforms.Length;
30        for (int i = 0; i < cubeTransforms.Length; i++)
31        {
32            if (cubeTransforms[1].position.x <= -21.51f)
33            {
34                if (firstRun)
35                {
36                    initialPosition = swingJulia.GetPostion();
37                    firstRun = false;
38                    isSwinging = true;
39                }
40                lineRenderer.SetPosition(i, cubeTransforms[i].position);
41            }
42        }
43    }
44
45    if (isSwinging)
46    {
47        swingJulia.MakeSwingJulia(initialPosition);
48    }
49
50
51    }
52}
53
54}

```

Listing 3: **RopeSpawn.cs**

```

1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class RopeSpawn : MonoBehaviour
6 {
7     [SerializeField]
8     private GameObject partPrefab, parentObject;
9
10    [SerializeField]
11    [Range(1, 1000)]

```

```

12     private int length = 1;
13
14     [SerializeField]
15     private float partDistance = 0.21f;
16
17     [SerializeField] private bool reset, spawn, snapFirst, snapLast;
18     // Start is called before the first frame update
19     void Start()
20     {
21
22     }
23
24     // Update is called once per frame
25     void Update()
26     {
27         if (reset)
28         {
29             foreach (GameObject tmp in GameObject.FindGameObjectsWithTag("Player"))
30             {
31                 Destroy(tmp);
32
33             }
34         }
35
36         if (spawn)
37         {
38             Spawn();
39             spawn = false;
40         }
41     }
42
43
44     public void Spawn()
45     {
46         int count = (int)(length / partDistance);
47         for (int x = 0; x < count; x++)
48         {
49             GameObject tmp;
50             tmp = Instantiate(partPrefab, new
51                 Vector3(transform.position.x, transform.position.y + partDistance *
52                 (x+1), transform.position.z), Quaternion.identity, parentObject.transform);
53             tmp.transform.eulerAngles = new Vector3(180, 0, 0);
54             tmp.name = parentObject.transform.childCount.ToString();
55
56             if (x == 0)
57             {
58                 Destroy(tmp.GetComponent<CharacterJoint>());
59                 if (snapFirst)
60                 {
61                     tmp.GetComponent<Rigidbody>().constraints = RigidbodyConstraints.FreezeAll;
62                 }
63                 else
64                 {
65                     tmp.GetComponent<CharacterJoint>().connectedBody =
66                     parentObject.transform.Find((parentObject.transform.childCount -
67                     1).ToString()).GetComponent<Rigidbody>();
68
69             }
70         }
71     }

```

```

71     if (snapLast)
72     {
73         parentObject.transform.Find((parentObject.transform.childCount).ToString()).GetComponent<Rigidbody>()
74             = RigidbodyConstraints.FreezeAll;
75     }
76 }



---


1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class RopeSpawn : MonoBehaviour
6  {
7      [SerializeField]
8      private GameObject partPrefab, parentObject;
9
10     [SerializeField]
11     [Range(1, 1000)]
12     private int length = 1;
13
14     [SerializeField]
15     private float partDistance = 0.21f;
16
17     [SerializeField] private bool reset, spawn, snapFirst, snapLast;
18     // Start is called before the first frame update
19     void Start()
20     {
21
22     }
23
24     // Update is called once per frame
25     void Update()
26     {
27         if (reset)
28         {
29             foreach (GameObject tmp in GameObject.FindGameObjectsWithTag("Player"))
30             {
31                 Destroy(tmp);
32             }
33         }
34     }
35
36         if (spawn)
37     {
38         Spawn();
39         spawn = false;
40     }
41
42 }
43
44     public void Spawn()
45     {
46         int count = (int)(length / partDistance);
47         for (int x = 0; x < count; x++)
48         {
49             GameObject tmp;
50             tmp = Instantiate(partPrefab, new
51                 Vector3(transform.position.x, transform.position.y + partDistance *
52                     (x+1), transform.position.z), Quaternion.identity, parentObject.transform);
53             tmp.transform.eulerAngles = new Vector3(180, 0, 0);

```

```

52     tmp.name = parentObject.transform.childCount.ToString();
53
54     if (x == 0)
55     {
56         Destroy(tmp.GetComponent<CharacterJoint>());
57         if (snapFirst)
58         {
59             tmp.GetComponent<Rigidbody>().constraints = RigidbodyConstraints.FreezeAll;
60         }
61     }
62     else
63     {
64         tmp.GetComponent<CharacterJoint>().connectedBody =
65         parentObject.transform.Find((parentObject.transform.childCount -
66             1).ToString()).GetComponent<Rigidbody>();
67     }
68 }
69
70     if (snapLast)
71     {
72         parentObject.transform.Find((parentObject.transform.childCount).ToString()).GetComponent<Rigidbody>()
73             = RigidbodyConstraints.FreezeAll;
74     }
75 }
76 }
```

Listing 4: SwingJulia.cs

```

1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using System.IO;
5 using UnityEngine;
6 using static UnityEditor.PlayerSettings;
7
8 public class SwingJulia : MonoBehaviour
9 {
10
11     public Rigidbody Julia;
12
13
14     private float cubeJuliaTimeStep;
15     private List<List<float>> timeSeriessRopeSwingJulia;
16     private Vector3 g = new Vector3(0f, -9.81f, 0f);
17     private bool rotationTriggered = false;
18     double starttime = 0;
19     float R = 6f; //Radius Ropefloat g = 9.81f; //Gravity
20     float alphaJulia = 0f; //Angle between crane and rope
21     float cCube = 1.1f;
22     float constantAirFriction = 1.2f; float areaJulia = 2.25f;
23
24     // Start is called before the first frame update
25     void Start()
26     {
27         timeSeriessRopeSwingJulia = new List<List<float>>();
28         starttime = Time.fixedTimeAsDouble;
29         //Julia = GetComponent<Rigidbody>();
30     }
31 }
```

```

// Update is called once per frame
33 void FixedUpdate()
34 {
35     cubeJuliaTimeStep += Time.deltaTime;
36 }
37 public Vector3 GetPostion()
38 {
39     return new Vector3(Julia.position.x, Julia.position.y + 6, Julia.position.z);
40 }
41 public void MakeSwingJulia(Vector3 connectedPos)
42 {
43     var ropeCubeJulia = connectedPos - Julia.position; // Endpunkt - Anfangspunkt
44     alphaJulia = (float)Math.Atan(ropeCubeJulia.x / ropeCubeJulia.y);
45     var FG = Julia.mass * g.magnitude * Math.Cos(alphaJulia);
46     var FZ = Julia.mass * (Math.Pow(Julia.velocity.magnitude, 2.0f)) / (R);
47     var normalizedVelocityJulia = Julia.velocity.normalized;
48     var FR = (float)(-0.5 * areaJulia * constantAirFriction * cCube *
49         Mathf.Pow(Julia.velocity.magnitude, 2.0f)) * normalizedVelocityJulia;
50     var FH = (FG + FZ) * Math.Sin(alphaJulia);
51     var FV = (FG + FZ) * Math.Cos(alphaJulia);
52     var centripedalForceJulia = new Vector3((float)FH, (float)FV, 0.0f) ;
53     var forceJ = centripedalForceJulia;
54     Julia.AddForce(forceJ);
55     var degree = ConvertRadiansToDegrees(alphaJulia);
56     cubeJuliaTimeStep += Time.deltaTime;
57     timeSeriessRopeSwingJulia.Add(new List<float>() { cubeJuliaTimeStep,
58         Julia.position.x, Julia.position.y, alphaJulia, (float)degree, (float)FH,
59         (float)FV, forceJ.x, forceJ.y, forceJ.z });
60 }
61 void OnApplicationQuit()
62 {
63     WriteTimeSeriessRopeSwingJuliaToCsv();
64 }
65 void WriteTimeSeriessRopeSwingJuliaToCsv()
66 {
67     using (var streamWriter = new StreamWriter("timeSeriesRopeJulia.csv"))
68     {
69         streamWriter.WriteLine("currentTimeStep, cubeJulia.position.x,
70             cubeJulia.position.y, alphaJulia, degree, horizonForceJulia,
71             verticalForceJulia, -frictionForceJulia.x + horizonForceJulia,
72             -frictionForceJulia.y + verticalForceJulia, -frictionForceJulia.z +
73             zAxisForceJulia");
74
75         foreach (List<float> timeStep in timeSeriessRopeSwingJulia)
76         {
77             streamWriter.WriteLine(string.Join(", ", timeStep));
78             streamWriter.Flush();
79         }
80     }
81 }
82 public static double ConvertRadiansToDegrees(double radians)
83 {
84     double degrees = (180 / Math.PI) * radians;
85     return (degrees);
86 }
87 }

```

```
1 using System;  
2 using System.Collections;
```

```

3 using System.Collections.Generic;
4 using System.IO;
5 using UnityEngine;
6 using static UnityEditor.PlayerSettings;
7
8 public class SwingJulia : MonoBehaviour
9 {
10
11     public Rigidbody Julia;
12
13
14     private float cubeJuliaTimeStep;
15     private List<List<float>> timeSeriessRopeSwingJulia;
16     private Vector3 g = new Vector3(0f, -9.81f, 0f);
17     private bool rotationTriggered = false;
18     double starttime = 0;
19     float R = 6f; //Radius Ropefloat g = 9.81f; //Gravity
20     float alphaJulia = 0f; //Angle between crane and rope
21     float cCube = 1.1f;
22     float constantAirFriction = 1.2f; float areaJulia = 2.25f;
23
24     // Start is called before the first frame update
25     void Start()
26     {
27         timeSeriessRopeSwingJulia = new List<List<float>>();
28         starttime = Time.fixedTimeAsDouble;
29         //Julia = GetComponent<Rigidbody>();
30     }
31
32     // Update is called once per frame
33     void FixedUpdate()
34     {
35         cubeJuliaTimeStep += Time.deltaTime;
36     }
37     public Vector3 GetPostion()
38     {
39         return new Vector3(Julia.position.x, Julia.position.y + 6, Julia.position.z);
40     }
41     public void MakeSwingJulia(Vector3 connectedPos)
42     {
43         var ropeCubeJulia = connectedPos - Julia.position; // Endpunkt - Anfangspunkt
44         alphaJulia = (float)Math.Atan(ropeCubeJulia.x / ropeCubeJulia.y);
45         var FG = Julia.mass * g.magnitude * Math.Cos(alphaJulia);
46         var FZ = Julia.mass * (Math.Pow(Julia.velocity.magnitude, 2.0f)) / (R);
47         var normalizedVelocityJulia = Julia.velocity.normalized;
48         var FR = (float)(-0.5 * areaJulia * constantAirFriction * cCube *
49             Mathf.Pow(Julia.velocity.magnitude, 2.0f)) * normalizedVelocityJulia;
50         var FH = (FG + FZ) * Math.Sin(alphaJulia);
51         var FV = (FG + FZ) * Math.Cos(alphaJulia);
52         var centripedalForceJulia = new Vector3((float)FH, (float)FV, 0.0f) ;
53         var forceJ = centripedalForceJulia;
54         Julia.AddForce(forceJ);
55         var degree = ConvertRadiansToDegrees(alphaJulia);
56         cubeJuliaTimeStep += Time.deltaTime;
57         timeSeriessRopeSwingJulia.Add(new List<float>() { cubeJuliaTimeStep,
58             Julia.position.x, Julia.position.y, alphaJulia, (float)degree, (float)FH,
59             (float)FV, forceJ.x, forceJ.y, forceJ.z });
60     }
61
62     void OnApplicationQuit()
63     {
64         WriteTimeSeriessRopeSwingJuliaToCsv();

```

```

62 }
63 void WriteTimeSeriesRopeSwingJuliaToCsv()
64 {
65     using (var streamWriter = new StreamWriter("timeSeriesRopeJulia.csv"))
66     {
67         streamWriter.WriteLine("currentTimeStep, cubeJulia.position.x,
68             cubeJulia.position.y, alphaJulia, degree, horizonForceJulia,
69             verticalForceJulia, -frictionForceJulia.x + horizonForceJulia,
70             -frictionForceJulia.y + verticalForceJulia, -frictionForceJulia.z +
71             zAxisForceJulia");
72
73         foreach (List<float> timeStep in timeSeriesRopeSwingJulia)
74         {
75             streamWriter.WriteLine(string.Join(",", timeStep));
76             streamWriter.Flush();
77         }
78     }
79 }
80 public static double ConvertRadiansToDegrees(double radians)
81 {
82     double degrees = (180 / Math.PI) * radians;
83     return (degrees);
84 }
85 }
```

Listing 5: **RopeSpawn.cs**

```

1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class RopeSpawn : MonoBehaviour
6 {
7     [SerializeField]
8     private GameObject partPrefab, parentObject;
9
10    [SerializeField]
11    [Range(1, 1000)]
12    private int length = 1;
13
14    [SerializeField]
15    private float partDistance = 0.21f;
16
17    [SerializeField] private bool reset, spawn, snapFirst, snapLast;
18    // Start is called before the first frame update
19    void Start()
20    {
21    }
22
23    // Update is called once per frame
24    void Update()
25    {
26        if (reset)
27        {
28            foreach (GameObject tmp in GameObject.FindGameObjectsWithTag("Player"))
29            {
30                Destroy(tmp);
31            }
32        }
33    }
34 }
```

```

34     }
35
36     if (spawn)
37     {
38         Spawn();
39         spawn = false;
40     }
41
42 }
43
44 public void Spawn()
45 {
46     int count = (int)(length / partDistance);
47     for (int x = 0; x < count; x++)
48     {
49         GameObject tmp;
50         tmp = Instantiate(partPrefab, new
51             Vector3(transform.position.x, transform.position.y + partDistance *
52                 (x+1), transform.position.z), Quaternion.identity, parentObject.transform);
53         tmp.transform.eulerAngles = new Vector3(180, 0, 0);
54         tmp.name = parentObject.transform.childCount.ToString();
55
56         if (x == 0)
57         {
58             Destroy(tmp.GetComponent<CharacterJoint>());
59             if (snapFirst)
60             {
61                 tmp.GetComponent<Rigidbody>().constraints = RigidbodyConstraints.FreezeAll;
62             }
63             else
64             {
65                 tmp.GetComponent<CharacterJoint>().connectedBody =
66                     parentObject.transform.Find((parentObject.transform.childCount -
67                         1).ToString()).GetComponent<Rigidbody>();
68             }
69         }
70
71         if (snapLast)
72         {
73             parentObject.transform.Find((parentObject.transform.childCount).ToString()).GetComponent<Rigidbody>()
74                 = RigidbodyConstraints.FreezeAll;
75         }
76     }

```

---

```

1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class RopeSpawn : MonoBehaviour
6 {
7     [SerializeField]
8     private GameObject partPrefab, parentObject;
9
10    [SerializeField]
11    [Range(1, 1000)]
12    private int length = 1;
13

```

```

14     [SerializeField]
15     private float partDistance = 0.21f;
16
17     [SerializeField] private bool reset, spawn, snapFirst, snapLast;
18     // Start is called before the first frame update
19     void Start()
20     {
21
22     }
23
24     // Update is called once per frame
25     void Update()
26     {
27         if (reset)
28     {
29         foreach (GameObject tmp in GameObject.FindGameObjectsWithTag("Player"))
30         {
31             Destroy(tmp);
32
33         }
34     }
35
36         if (spawn)
37     {
38             Spawn();
39             spawn = false;
40         }
41
42     }
43
44     public void Spawn()
45     {
46         int count = (int)(length / partDistance);
47         for (int x = 0; x < count; x++)
48     {
49         GameObject tmp;
50         tmp = Instantiate(partPrefab, new
51             Vector3(transform.position.x, transform.position.y + partDistance *
52             (x+1), transform.position.z), Quaternion.identity, parentObject.transform);
53         tmp.transform.eulerAngles = new Vector3(180, 0, 0);
54         tmp.name = parentObject.transform.childCount.ToString();
55
56         if (x == 0)
57     {
58             Destroy(tmp.GetComponent<CharacterJoint>());
59             if (snapFirst)
60             {
61                 tmp.GetComponent<Rigidbody>().constraints = RigidbodyConstraints.FreezeAll;
62             }
63             else
64             {
65                 tmp.GetComponent<CharacterJoint>().connectedBody =
66                     parentObject.transform.Find((parentObject.transform.childCount -
67                     1).ToString()).GetComponent<Rigidbody>();
68             }
69         }
70
71         if (snapLast)
72     {

```

```

73         parentObject.transform.Find((parentObject.transform.childCount).ToString()).GetComponent<Rigidbody>()
74             = RigidbodyConstraints.FreezeAll;
75     }
76 }
```

Listing 6: SwingRomeo.cs

```

1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using System.IO;
5 using UnityEngine;
6
7 public class SwingRomeo : MonoBehaviour
8 {
9     public Rigidbody Romeo;
10
11     private List<List<float>> timeSeriessRopeSwingRomeo;
12
13
14     double starttime = 0;
15     private float currentTimestep; // s
16     private Vector3 g= new Vector3(0f, -9.81f, 0f);
17     float alphaRomeo = 0f; //Angle between crane and rope
18     private bool rotationTriggered = false;
19     float cCube = 1.1f;
20     float constantAirFriction = 1.2f;
21     float areaRomeo = 2.25f;
22     float R = 6f; //Radius Rope
23
24     // Start is called before the first frame update
25     void Start()
26     {
27         timeSeriessRopeSwingRomeo = new List<List<float>>();
28         starttime = Time.fixedTimeAsDouble;
29         // Romeo = GetComponent<Rigidbody>();
30     }
31
32     // Update is called once per frame
33     void FixedUpdate()
34     {
35         currentTimestep += Time.deltaTime;
36     }
37
38
39
40     public Vector3 GetPostion()
41     {
42         return new Vector3(Romeo.position.x, Romeo.position.y + 6, Romeo.position.z);
43     }
44
45     public void MakeSwing(Vector3 connectedPos)
46     {
47         var ropeCubeRomeo = connectedPos - Romeo.position; // Endpunkt - Anfangspunkt
48         Debug.Log("diff " + ropeCubeRomeo);
49         alphaRomeo = (float)Math.Atan(ropeCubeRomeo.x / ropeCubeRomeo.y);
50         Debug.Log("alpha " + alphaRomeo);
51         var FG = Romeo.mass * g.magnitude * Math.Cos(alphaRomeo);
52         var FZ = Romeo.mass * (Math.Pow(Romeo.velocity.magnitude, 2.0f)) / (R);
```

```

54     var normalizedVelocityRomeo = Romeo.velocity.normalized;
55     var FR = (float)(-0.5 * areaRomeo * constantAirFriction * cCube *
56         Mathf.Pow(Romeo.velocity.magnitude, 2.0f)) * normalizedVelocityRomeo;
57     var FH = (FG + FZ) * Math.Sin(alphaRomeo);
58     var FV = (FG + FZ) * Math.Cos(alphaRomeo);
59     var centripetalForceRomeo = new Vector3((float)FH, (float)FV, 0.0f) ;
60     var force = centripetalForceRomeo + FR;
61     Romeo.AddForce(force);
62     var degree = ConvertRadiansToDegrees(alphaRomeo);
63     //currentTimeStep += Time.deltaTime;
64     timeSeriessRopeSwingRomeo.Add(new List<float>() { currentTimeStep, Romeo.position.x,
65         Romeo.position.y, alphaRomeo, (float)degree, (float)FH, (float)FV, force.x,
66         force.y, force.z });
67 }
68
69 void OnApplicationQuit()
70 {
71     WriteTimeSeriessRopeSwingRomeoToCsv();
72 }
73 void WriteTimeSeriessRopeSwingRomeoToCsv()
74 {
75     using (var streamWriter = new StreamWriter("timeSeriesRopeRomeo.csv"))
76     {
77         streamWriter.WriteLine("currentTimeStep, cubeRomeo.position.x,
78             cubeRomeo.position.y,alphaRomeo,degree,horizonForceRomeo,verticalForceRomeo,
79             -frictionForceRomeo.x + horizonForceRomeo, -frictionForceRomeo.y +
80             verticalForceRomeo, -frictionForceRomeo.z + zAxisForceRomeo");
81
82         foreach (List<float> timeStep in timeSeriessRopeSwingRomeo)
83         {
84             streamWriter.WriteLine(string.Join(", ", timeStep));
85             streamWriter.Flush();
86         }
87     }
88 }
89
90 }

```

---

```

1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using System.IO;
5 using UnityEngine;
6
7 public class SwingRomeo : MonoBehaviour
8 {
9     public Rigidbody Romeo;
10
11     private List<List<float>> timeSeriessRopeSwingRomeo;
12
13     double starttime = 0;
14     private float currentTimeStep; // s
15     private Vector3 g= new Vector3(0f, -9.81f, 0f);
16     float alphaRomeo = 0f; //Angle between crane and rope

```

```

18     private bool rotationTriggered = false;
19     float cCube = 1.1f;
20     float constantAirFriction = 1.2f;
21     float areaRomeo = 2.25f;
22     float R = 6f; //Radius Rope
23
24     // Start is called before the first frame update
25     void Start()
26     {
27         timeSeriesRopeSwingRomeo = new List<List<float>>();
28         startTime = Time.fixedTimeAsDouble;
29         // Romeo = GetComponent<Rigidbody>();
30     }
31
32     // Update is called once per frame
33     void FixedUpdate()
34     {
35         currentTimeStep += Time.deltaTime;
36     }
37
38 }
39
40
41     public Vector3 GetPosition()
42     {
43         return new Vector3(Romeo.position.x, Romeo.position.y + 6, Romeo.position.z);
44     }
45
46     public void MakeSwing(Vector3 connectedPos)
47     {
48         var ropeCubeRomeo = connectedPos - Romeo.position; // Endpunkt - Anfangspunkt
49         Debug.Log("diff " + ropeCubeRomeo);
50         alphaRomeo = (float)Math.Atan(ropeCubeRomeo.x / ropeCubeRomeo.y);
51         Debug.Log("alpha " + alphaRomeo);
52         var FG = Romeo.mass * g.magnitude * Math.Cos(alphaRomeo);
53         var FZ = Romeo.mass * (Math.Pow(Romeo.velocity.magnitude, 2.0f)) / (R);
54         var normalizedVelocityRomeo = Romeo.velocity.normalized;
55         var FR = (float)(-0.5 * areaRomeo * constantAirFriction * cCube *
56             Mathf.Pow(Romeo.velocity.magnitude, 2.0f)) * normalizedVelocityRomeo;
57         var FH = (FG + FZ) * Math.Sin(alphaRomeo);
58         var FV = (FG + FZ) * Math.Cos(alphaRomeo);
59         var centripetalForceRomeo = new Vector3((float)FH, (float)FV, 0.0f) ;
60         var force = centripetalForceRomeo + FR;
61         Romeo.AddForce(force);
62         var degree = ConvertRadiansToDegrees(alphaRomeo);
63         //currentTimeStep += Time.deltaTime;
64         timeSeriesRopeSwingRomeo.Add(new List<float>() { currentTimeStep, Romeo.position.x,
65             Romeo.position.y, alphaRomeo, (float)degree, (float)FH, (float)FV, force.x,
66             force.y, force.z });
67     }
68
69     void OnApplicationQuit()
70     {
71         WriteTimeSeriesRopeSwingRomeoToCsv();
72     }
73     void WriteTimeSeriesRopeSwingRomeoToCsv()
74     {
75         using (var streamWriter = new StreamWriter("timeSeriesRopeRomeo.csv"))
76         {
77             streamWriter.WriteLine("currentTimeStep, cubeRomeo.position.x,
78                 cubeRomeo.position.y, alphaRomeo, degree, horizonForceRomeo, verticalForceRomeo,
```

```

        -frictionForceRomeo.x + horizonForceRomeo, -frictionForceRomeo.y +
        verticalForceRomeo, -frictionForceRomeo.z + zAxisForceRomeo");

76
77     foreach (List<float> timeStep in timeSeriessRopeSwingRomeo)
78     {
79         streamWriter.WriteLine(string.Join(",", timeStep));
80         streamWriter.Flush();
81     }
82 }
83 }
84
85 public static double ConvertRadiansToDegrees(double radians)
86 {
87     double degrees = (180 / Math.PI) * radians;
88     return (degrees);
89 }
90 }
```

## A.2 Code für die Datenaufbereitung des Elastischen Stosses des Lab 2

Nachfolgend ist der Code abgebildet, welche für die Visuelle Datenaufbereitung des Elastischen Stosses der Grafiken benötigt wurde.

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 df = pd.read_csv("./TimeSeries/time_seriesElastic_pt2.csv")
5
6 #plt.figure(figsize=(20,20))
7 #plt.subplot(4,1,1)
8 plt.plot(df["currentTimeStep"], df[" cubeRomeo.position.x"])
9 plt.ylabel("[s] = m")
10 plt.xlabel("[t] = s")
11 plt.title("Ort als Funktion der Zeit")
12 plt.savefig('../Semesterprojekt Physik Engines/images/Elastisch/OrtAlsFunktionDerZeit.png',
13             dpi=300, bbox_inches='tight')
14 plt.show()
15
16 #plt.subplot(4,1,2)
17 plt.plot(df["currentTimeStep"], df[" cubeRomeo.velocity.x"])
18 plt.ylabel("[v] = m/s")
19 plt.xlabel("[t] = s")
20 plt.title("Geschwindigkeit als Funktion der Zeit")
21 plt.savefig('../Semesterprojekt Physik
22             Engines/images/Elastisch/GeschwindigkeitAlsFunktionDerZeit.png', dpi=300,
23             bbox_inches='tight')
24 plt.show()
25
26 #%%
27 plt.subplot(4,1,3)
28 plt.plot(df["currentTimeStep"], df[" cubeRomeoKinetic"])
29 plt.ylabel("[J] = Nm")
30 plt.xlabel("[t] = s")
31 plt.title("Kinetische Energie als Funktion der Zeit")
32 plt.show()
33
34 plt.subplot(4,1,4)
35 plt.plot(df["currentTimeStep"], df[" springPotentialEnergy"])
36 plt.ylabel("[J] = Nm")
37 plt.xlabel("[t] = s")
```

```

35 plt.title("Potentielle Energie als Funktion der Zeit")
36
37 plt.show()

```

### A.3 Code für die Datenaufbereitung des Inelastischen Stosses des Lab 2

Nachfolgend ist der Code abgebildet, welche für die Visuelle Datenaufbereitung des Inelastischen Stosses der Grafiken benötigt wurde.

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4
5 df = pd.read_csv("./TimeSeries/time_seriesInelastic_pt2.csv")
6
7 #plt.figure(figsize=(20,10))
8 #plt.subplot(4,1,1)
9 plt.plot(df["cubeJuliaTimeStep"], df[" GesamtImpluls"])
10 plt.ylabel("[p] = Ns")
11 plt.xlabel("[t] = s")
12 plt.xlim(0,19)
13 plt.ylim(-0.5,5)
14 plt.title("Gesamtmomentum als Funktion der Zeit")
15 plt.savefig('../Semesterprojekt Physik Engines/images/Inelastisch/GesamtImpluls.png',
16             dpi=300, bbox_inches='tight')
17 plt.show()
18
19 #plt.figure(figsize=(20,20))
20 #plt.subplot(4,1,1)
21 plt.plot(df["cubeJuliaTimeStep"], df[" cubeRomeoImpulse"], label="Romeo")
22 plt.plot(df["cubeJuliaTimeStep"], df[" cubeJuliaImpulse"], label="Julia")
23 plt.ylabel("[p] = Ns")
24 plt.xlabel("[t] = s")
25 plt.xlim(0,19)
26 plt.ylim(-0.5,5)
27 plt.legend()
28 plt.title("Impuls Romeo & Julia als Funktion der Zeit")
29 plt.savefig('../Semesterprojekt Physik Engines/images/Inelastisch/ImpulsRomeoJulia.png',
30             dpi=300, bbox_inches='tight')
31 plt.show()
32
33 #plt.figure(figsize=(20,20))
34 #plt.subplot(4,1,1)
35 plt.plot(df["cubeJuliaTimeStep"], df[" cubeRomeo.position.x"], label="Romeo")
36 plt.plot(df["cubeJuliaTimeStep"], df[" cubeJulia.position.x"], label="Julia")
37 plt.ylabel("[s] = m")
38 plt.xlabel("[t] = s")
39 plt.xlim(0,19)
40 plt.ylim(-15,10)
41 plt.legend()
42 plt.title("Ort Romeo & Julia als Funktion der Zeit")
43 plt.savefig('../Semesterprojekt Physik
44             Engines/images/Inelastisch/OrtRomeoJuliaAlsFunktionDerZeit.png', dpi=300,
45             bbox_inches='tight')
46 plt.show()
47
48 #plt.subplot(4,1,2)
49 plt.plot(df["cubeJuliaTimeStep"], df[" cubeRomeo.velocity.x"], label="Romeo")
50 plt.plot(df["cubeJuliaTimeStep"], df[" cubeJulia.velocity.x"], label="Julia")
51 plt.ylabel("[v] = m/s")

```

```

48 plt.xlabel("[t] = s")
49 plt.xlim(0,19)
50 plt.ylim(-2.5,2.5)
51 plt.legend()
52 plt.title("Geschwindigkeit Romeo & Julia als Funktion der Zeit")
53 plt.savefig('../Semesterprojekt Physik
      Engines/images/Inelastisch/GeschwindigkeitRomeoJulia.png', dpi=300, bbox_inches='tight')
54 plt.show()
55
56
57 #plt.subplot(4,1,2)
58 plt.plot(df["cubeJuliaTimeStep"], df[" velocityEnd"])
59 plt.ylabel("[v] = m/s")
60 plt.xlabel("[t] = s")
61 plt.xlim(0,19)
62 plt.ylim(-0.1,1.2)
63 plt.title("Endgeschwindigkeit als Funktion der Zeit")
64 plt.savefig('../Semesterprojekt Physik Engines/images/Inelastisch/Endgeschwindigkeit.png',
      dpi=300, bbox_inches='tight')
65 plt.show()
66
67 #%%
68 #plt.subplot(4,1,2)
69 plt.plot(df["cubeJuliaTimeStep"], df[" cubeRomeo.velocity.x"])
70 plt.ylabel("[v] = m/s")
71 plt.xlabel("[t] = s")
72 plt.xlim(0,20)
73 plt.title("Geschwindigkeit Romeo als Funktion der Zeit")
74 plt.savefig('../Semesterprojekt Physik Engines/images/Inelastisch/GeschwindigkeitRomeo.png',
      dpi=300, bbox_inches='tight')
75 plt.show()
76
77 #plt.subplot(4,1,3)
78 plt.plot(df["cubeJuliaTimeStep"], df[" cubeJulia.velocity.x"])
79 plt.ylabel("[v] = m/s")
80 plt.xlabel("[t] = s")
81 plt.xlim(0,20)
82 plt.title("Geschwindigkeit Julia als Funktion der Zeit")
83 plt.savefig('../Semesterprojekt Physik Engines/images/Inelastisch/GeschwindigkeitJulia.png',
      dpi=300, bbox_inches='tight')
84 plt.show()
85
86
87 plt.subplot(4,1,4)
88 plt.plot(df["cubeJuliaTimeStep"], df[" cubeRomeoKinetic"])
89 plt.ylabel("[J] = Nm")
90 plt.xlabel("[t] = s")
91 plt.title("Energie Romeo als Funktion der Zeit")
92 plt.show()
93
94 plt.subplot(4,1,3)
95 plt.plot(df["cubeJuliaTimeStep"], df[" cubeKineticEnd"])
96 plt.ylabel("[J] = Nm")
97 plt.xlabel("[t] = s")
98 plt.title("Endkinetik als Funktion der Zeit")
99 plt.show()
100 plt.savefig('../Semesterprojekt Physik Engines/images/Inelastisch/cubeKineticEnd.png',
      dpi=300, bbox_inches='tight')
101
102 plt.subplot(4,1,4)
103 plt.plot(df["cubeJuliaTimeStep"], df[" forceOnJulia"])
104 plt.ylabel("N")

```

```

105 plt.xlabel("[t] = s")
106 plt.title("Kraft auf Julia als Funktion der Zeit")
107 plt.savefig('../Semesterprojekt Physik Engines/images/Inelastisch/forceOnJulia.png',
108             dpi=300, bbox_inches='tight')

```

## A.4 Code für die Datenaufbereitung des Schwunges von Julia des Lab 3

Nachfolgend ist der Code abgebildet, welche für die Visuelle Datenaufbereitung des Schwung von Julia.

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon May 29 11:15:58 2023
4
5 @author: Asha
6 """
7
8 import pandas as pd
9 import matplotlib.pyplot as plt
10
11
12 df_rom = pd.read_csv("./timeSeriesRopeRomeo.csv")
13 df_jul = pd.read_csv("./timeSeriesRopeJulia.csv")
14
15 #plt.figure(figsize=(20,20))
16 plt.plot(df_rom["cubeRomeo.position.x"], df_rom["cubeRomeo.position.y"], label="Romeo")
17 plt.plot(df_jul["cubeJulia.position.x"], df_jul["cubeJulia.position.y"], label="Julia")
18 plt.xlabel("Position x [s] = m")
19 plt.ylabel("Position y [s] = m")
20 plt.title("Bewegung Wrfel Romeo & Julia")
21 plt.legend()
22 plt.savefig('../Semesterprojekt Physik Engines/images/ropeJulia/Ortdiagramm.png', dpi=300,
22             bbox_inches='tight')
23 plt.show()
24
25
26 plt.plot(df_rom["currentTimeStep"],df_rom["degree"], label="Romeo")
27 plt.plot(df_jul["currentTimeStep"],df_jul["degree"], label="Julia")
28 plt.ylabel("[s] = alpha")
29 plt.xlabel("[t] = s")
30 plt.legend()
31 plt.title("Auslenkung in Grad als Funktion der Zeit")
32 plt.savefig('../Semesterprojekt Physik Engines/images/ropeJulia/AuslenkungDeg.png', dpi=300,
32             bbox_inches='tight')
33 plt.show()

```

## A.5 Code für die Datenaufbereitung des Schwunges von Romeo des Lab 3

Nachfolgend ist der Code abgebildet, welche für die Visuelle Datenaufbereitung des Schwung von Romeo.

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon May 29 11:02:14 2023
4
5 @author: Asha
6 """
7
8 import pandas as pd
9 import matplotlib.pyplot as plt

```

```

10
11 df = pd.read_csv("./TimeSeries/timeSeriesRopeRomeo.csv")
12
13 #plt.figure(figsize=(20,20))
14 plt.plot(df[" cubeRomeo.position.x"], df[" cubeRomeo.position.y"])
15
16 plt.xlabel("Position x [s] = m")
17 plt.ylabel("Position y [s] = m")
18 plt.title("Bewegung Wrfel Romeo")
19 plt.savefig('../Semesterprojekt Physik Engines/images/ropeRomeo/Ortdiagramm.png', dpi=300,
20             bbox_inches='tight')
21 plt.show()
22
23 plt.plot(df["currentTimeStep"],df["degree"])
24 plt.ylabel("[s] = alpha")
25 plt.xlabel("[t] = s")
26 plt.title("Auslenkung in Grad als Funktion der Zeit")
27 plt.savefig('../Semesterprojekt Physik Engines/images/ropeRomeo/AuslenkungDeg.png', dpi=300,
28             bbox_inches='tight')
29 plt.show()
30 #%%
31
32 #plt.figure(figsize=(20,20))
33 plt.plot(df["currentTimeStep"],df[" cubeRomeo.position.x"])
34 plt.ylabel("[s] = m")
35 plt.xlabel("[t] = s")
36 plt.title("Bewegung Wrfel Romeo x")
37 plt.savefig('../Semesterprojekt Physik Engines/images/ropeRomeo/x(t).png', dpi=300,
38             bbox_inches='tight')
39 plt.show()
40
41 plt.plot(df["currentTimeStep"],df[" cubeRomeo.position.y"])
42 plt.ylabel("[s] = m")
43 plt.xlabel("[t] = s")
44 plt.title("Bewegung Wrfel Romeo y")
45 plt.savefig('../Semesterprojekt Physik Engines/images/ropeRomeo/y(t).png', dpi=300,
46             bbox_inches='tight')
47 plt.show()
48
49 plt.figure()
50 plt.plot(df["currentTimeStep"],df["alphaRomeo"])
51 plt.ylabel("[s] = alpha")
52 plt.xlabel("[t] = s")
53 plt.title("Auslenkung in Rad als Funktion der Zeit")
54 plt.ylim(25,54)
55 plt.xticks([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13])
56 plt.savefig('../Semesterprojekt Physik Engines/images/ropeRomeo/AuslenkungRad.png', dpi=300,
57             bbox_inches='tight')
58 plt.show()

```

## Abbildungsverzeichnis

1	Beschleunigung des Würfels . . . . .	4
2	Elastischer Zusammenstoss mit der Feder . . . . .	4
3	Inelastischer zusammenstoss mit dem anderen Würfel . . . . .	4
4	Schwingung des Würfels . . . . .	5
5	Kräfte auf eine Pendelmasse [1] . . . . .	9
6	Kräftediagramm Skizze . . . . .	10
7	Experiment Übersicht . . . . .	12
8	Einstellung Julia . . . . .	13
9	Einstellung Romeo . . . . .	13
10	Einstellung Feder . . . . .	14
11	Feder mit markierter Berührungs punkt . . . . .	14
12	Experiment Übersicht . . . . .	15
13	Experiment Übersicht . . . . .	16
14	Impuls Romeo & Julia als Funktion der Zeit . . . . .	17
15	GesamtImpluls als Funktion der Zeit . . . . .	18
16	Ort Romeo & Julia als Funktion der Zeit . . . . .	18
17	Geschwindigkeit Romeo & Julia als Funktion der Zeit . . . . .	19
18	Endgeschwindigkeit als Funktion der Zeit . . . . .	19
19	Auslenkung in Grad als Funktion der Zeit . . . . .	20
20	Romeo & Julia Ortsdiagramm . . . . .	20

## Literatur

- [1] Tipler, Paul A. u.a. *Physik Für Studierende Der Naturwissenschaften Und Technik*. Springer Spektrum. ISBN: 978-3-662-58280-0.