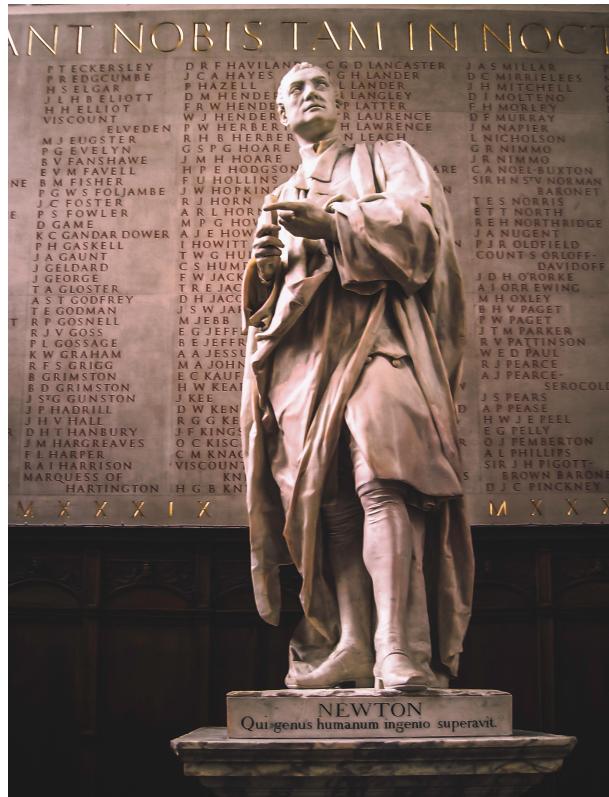


Semesterprojekt Physik Engines

Kim Lan Vu, Michel Steiner, Asha Schwegler

30. Mai 2023



Inhaltsverzeichnis

1 Zusammenfassung	3
2 Aufbau des Experiments	4
2.1 Aufbau des Lab 2, „Würfel 1 bewegt sich und stösst“	4
2.2 Aufbau des Lab 3, „Beide Würfel schwingen gedämpft“	4
3 Physikalische Beschreibung der einzelnen Vorgänge	6
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	8
3.2.2 Zentripetalkraf	8
3.2.3 Reibungskraft	8
3.2.4 Resultierende Kraft	8
4 Beschreibung der Implementierung inklusive Screenshots aus Unity	9
4.1 Lab 2: Würfel bewegt sich und stösst	9
4.2 Lab 3: Beide Würfel schwingen gedämpft	12
5 Resultate mit grafischer Darstellung	14
5.1 Lab 2	14
5.2 Lab 3	16
6 Rückblick und Lehren aus dem Versuch	18
A Anhang	19
A.1 Code für das Lab 2 und 3	19
A.2 Code für die Datenaufbereitung des Elastischen Stosses des Lab 2	40
A.3 Code für die Datenaufbereitung des Inelastischen Stosses des Lab 2	41
A.4 Code für die Datenaufbereitung des Schwunges von Julia des Lab 3	43
A.5 Code für die Datenaufbereitung des Schwunges von Romeo des Lab 3	44

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.

Im ersten Lab werden die inelastischen und elastischen Stöße untersucht.

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 abgedeferten 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.

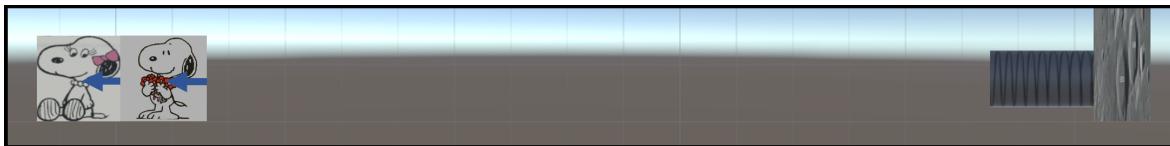


Abbildung 3: Inelastischer Zusammenstoß mit dem anderen Würfel

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 10 zu sehen.

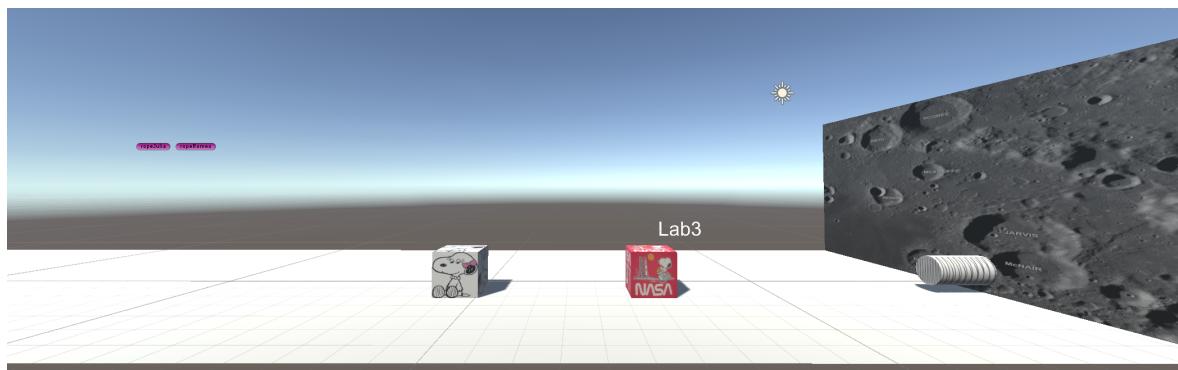


Abbildung 4: Schwingung des Würfels

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

$$F = m * a$$

Um dieses Anfangswertproblems zu lösen leiten wir die Geschwindigkeit ab

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

Die Zeit, die gebraucht wird um den Würfel zu beschleunigen, wird durch folgende Formel beschrieben

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

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 so dimensioniert, dass der Würfel nicht auf die Wand trifft. Die kinetische Energie des Würfels wird mit folgender Formel berechnet

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

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 übetragen. Die Feder speichert diese Energie in Form von potentieller Energie, da sie zusammengedrückt wird. Sobald sie Romeo zurück stößt, 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

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

Die Gleichsetzung der Energien, sieht folgendermassen aus

$$E_{kinRomeo} = E_{potFeder}$$

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

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 langsam Stoss gewährleisten.

3.1.3 Inelastischer Stoss

Beim vollständigen inelastischen Stoss, werden beide Stosspartner nach der Kollision verbunden sein und dieselbe Geschwindigkeit haben die der Impulse der beiden Körper:

$$Impuls_{Romeo} = m_{Romeo} * v_{Romeo}$$

$$Impuls_{Julia} = m_{Julia} * v_{Julia}$$

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:

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

Die Endgeschwindigkeit ist die Geschwindigkeit, die beide Körper gemeinsam haben nach dem Stoss. 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 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
- $c_w = 1.1$
- $\rho_{Luft} = 1.2 \text{ kg/m}^3$
- $g = 9.81 \text{ m/s}^2$

3.2.1 Radialer Anteil der Gewichtskraft

Die Gewichtskraft ist die Kraft, die durch die Wirkung der Gravitation, auf den Körper wirkt. Die Richtung hängt bei einer Kreisbewegung vom Winkel ab, diese wird in der Formelberechnung berücksichtigt.

$$F_g = m * g *$$

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 Kreisbewegung steht und zum Mittelpunkt weist. In diesem Fall wird sie durch die Gravitationskraft und die Reibungskraft hervorgerufen und ist somit keine eigenständige neue Kraft. Die Zentripetalkraft hat die allgemeine Formel:

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

3.2.3 Reibungskraft

3.2.4 Resultierende Kraft

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.

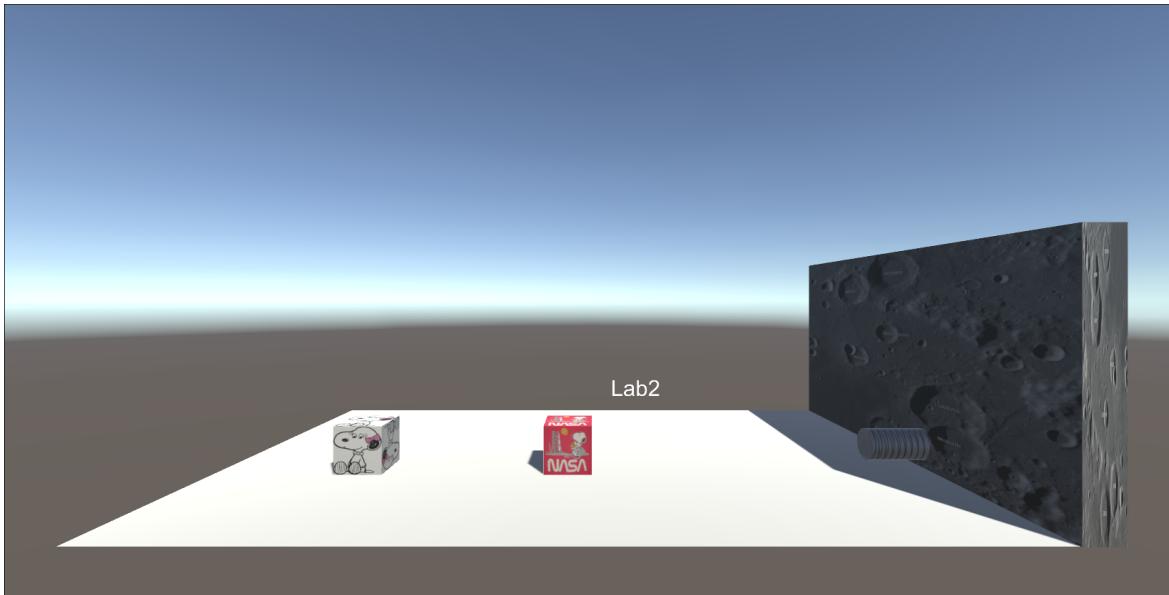


Abbildung 5: Experiment Übersicht

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.

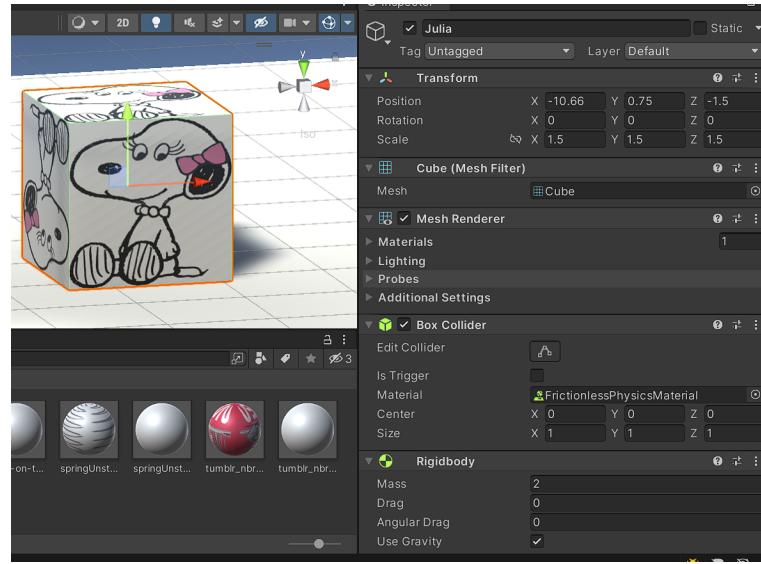


Abbildung 6: Einstellung Julia

- 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.

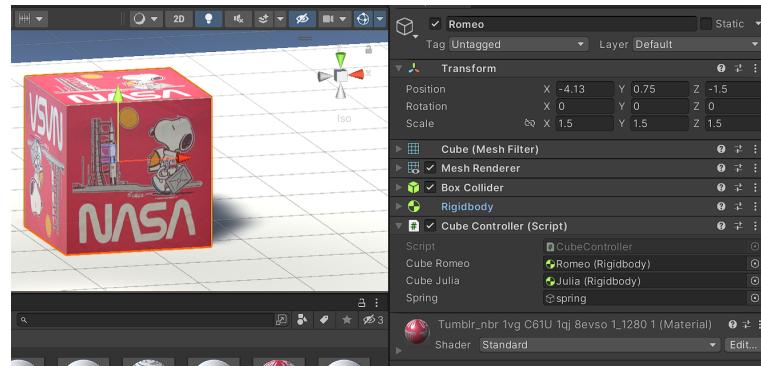


Abbildung 7: Einstellung Romeo

- Spring:

Wie in Abbildung 7 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

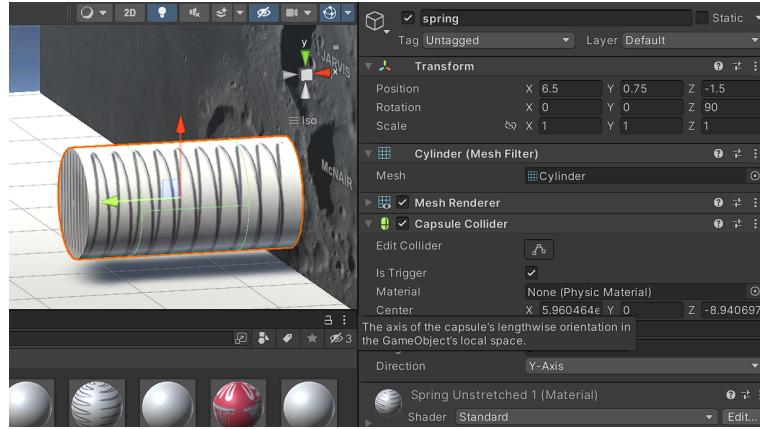


Abbildung 8: Einstellung Feder

- 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 9 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)));
```

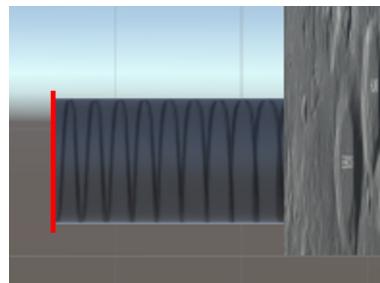


Abbildung 9: Feder mit markierter Berührungs punkt

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 [10](#) zur Laufzeit visuell dargestellt.

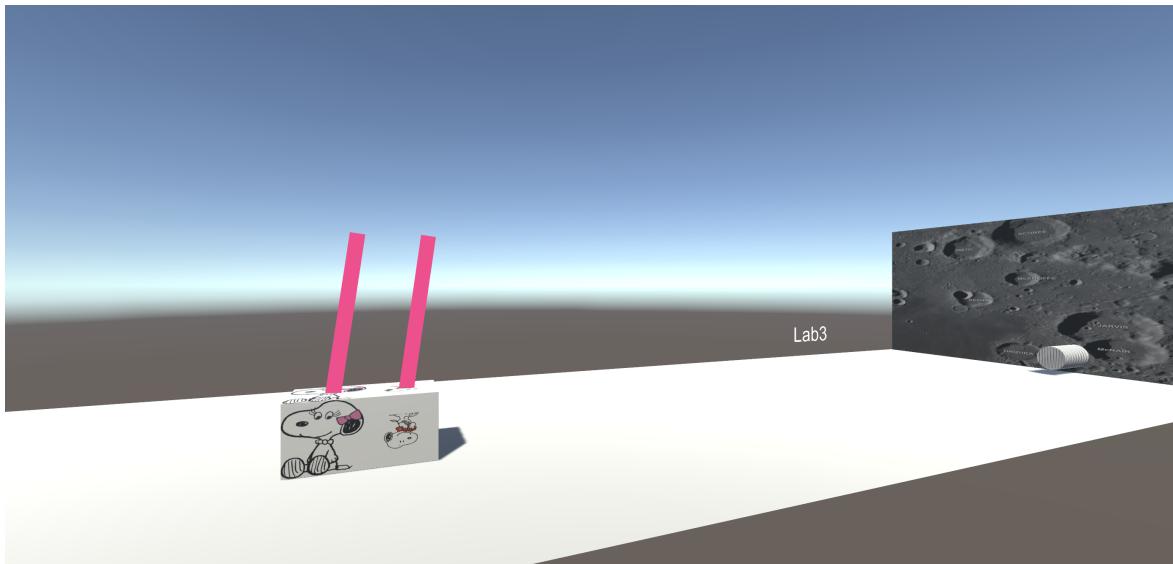


Abbildung 10: Experiment Übersicht

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

- Seile: Die Seile die in der Abbildung [10](#) in Pink ersichtlich sind, werden von den Würfel zu den vordefinierten Punkten wie in Abbildung [11](#) ersichtlich sind, gezeichnet.

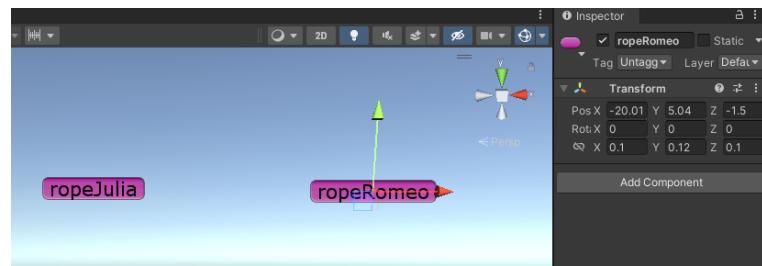


Abbildung 11: Experiment Übersicht

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 12 bis Abbildung 16 aufgegliedert.

In Abbildung 12 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 13 sieht man die addierten Impulse und wie dies nach der Kollision in der Tat gleich bleibt.

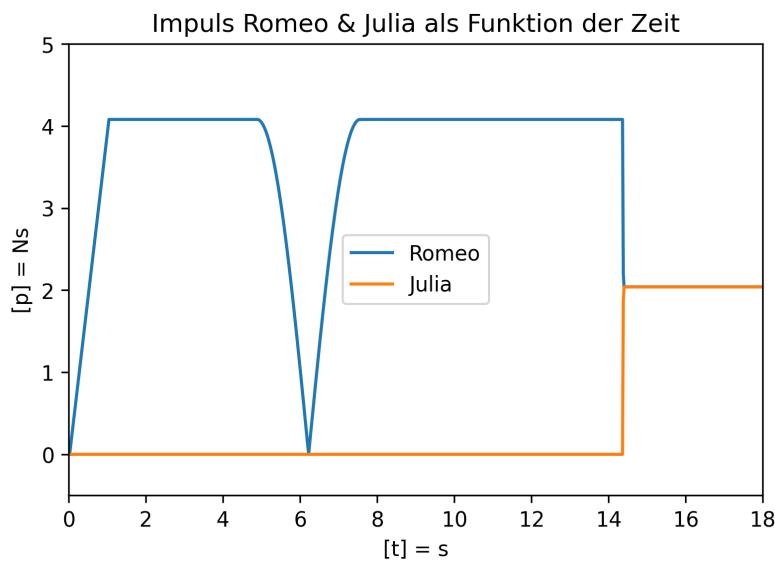


Abbildung 12: Impuls Romeo & Julia als Funktion der Zeit

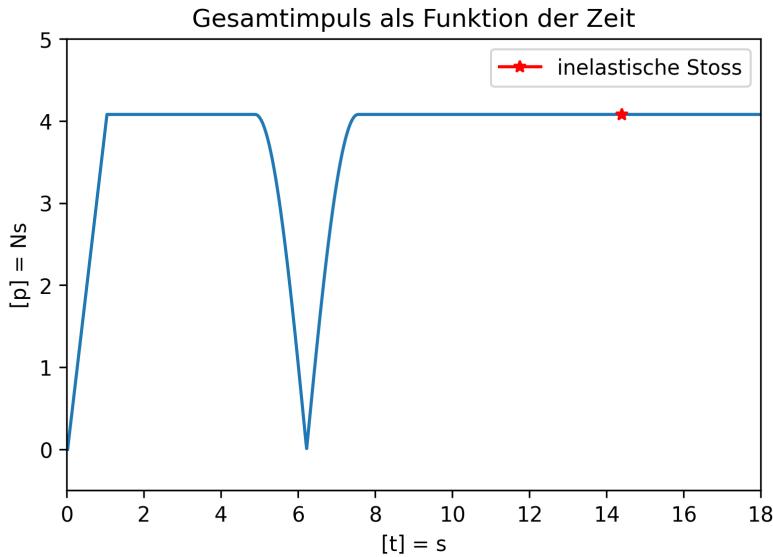


Abbildung 13: Gesamtmpuls als Funktion der Zeit

Im Ort-Zeit Diagramm in der Abbildung 14 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.

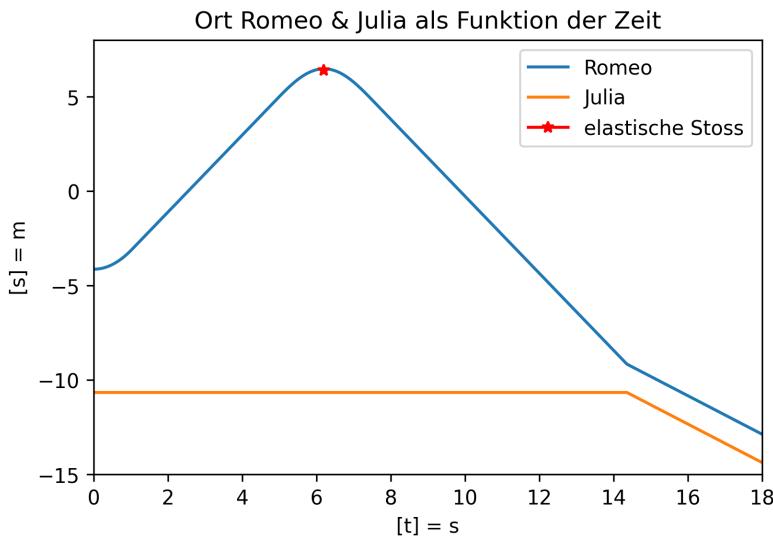


Abbildung 14: Ort Romeo & Julia als Funktion der Zeit

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 Rücksprall 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 Physikalisch Beschreibung berechnet, weiter. In der Abbildung 16 sieht man, dass dies in unseren Versuch auch übereinstimmt.

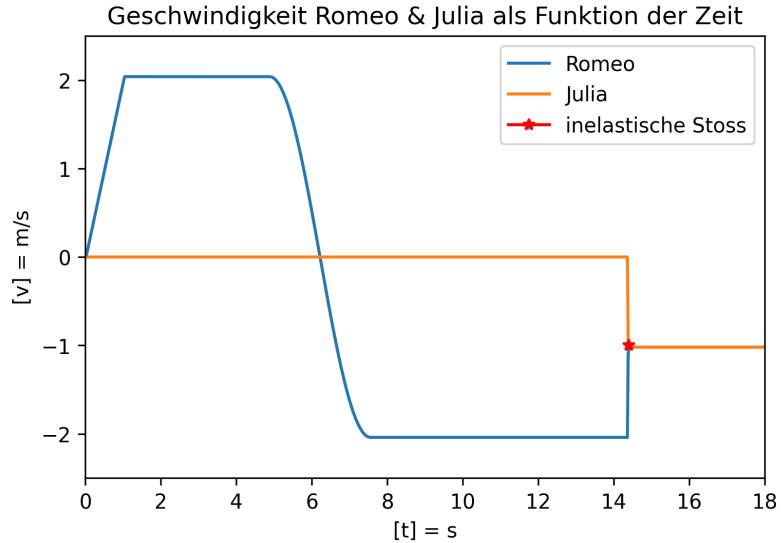


Abbildung 15: Geschwindigkeit Romeo & Julia als Funktion der Zeit

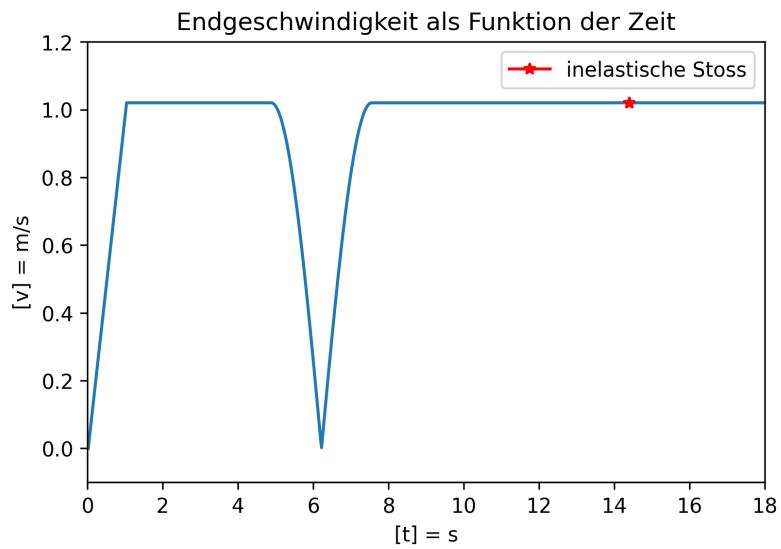


Abbildung 16: Endgeschwindigkeit als Funktion der Zeit

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 17 zu sehen ist, kontinuierlich ab. Dies wird auch in Abbildung 18 deutlich, wo die Position der beiden Würfel sich immer weniger weit vom Mittelpunkt: -20m bei Romeo und -21.5m bei Julia entfernen.

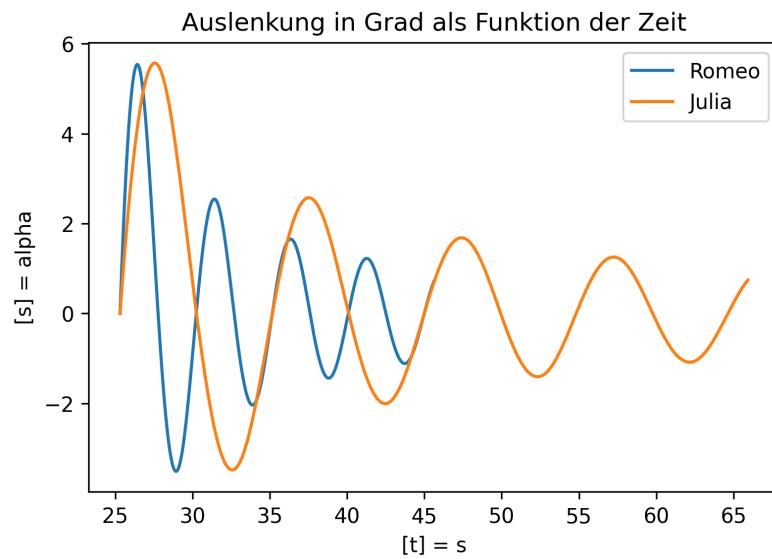


Abbildung 17: Auslenkung in Grad als Funktion der Zeit

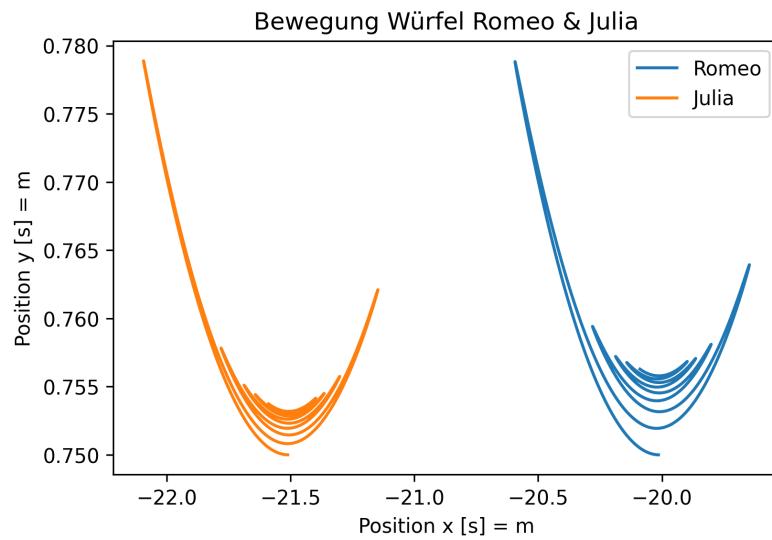


Abbildung 18: Romeo & Julia Ortsdiagramm

6 Rückblick und Lehren aus dem Versuch

Uns wurde beim Programmieren mit Unity ist es bewusst, wie wichtig es ist Gleitkommazahlen zu berücksichtigen, da Unity iterativ arbeitet. Dadurch entstanden Ungenauigkeiten, die das Experiment nicht korrekt durchlaufen liessen. Dariüber 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;

```



```

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
168     foreach (List<float> timeStep in timeSeriesInelasticCollision)
169     {
170         streamWriter.WriteLine(string.Join(",", timeStep));
171         streamWriter.Flush();
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 + FR;
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 + FR;
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_romeo = pd.read_csv("./TimeSeries/timeSeriesRopeRomeo.csv")
13 df_julia = pd.read_csv("./TimeSeries/timeSeriesRopeJulia.csv")
14
15 #plt.figure(figsize=(20,20))
16 plt.plot(df_romeo["cubeRomeo.position.x"], df_romeo["cubeRomeo.position.y"], label="Romeo")
17 plt.plot(df_julia["cubeJulia.position.x"], df_julia["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_romeo["currentTimeStep"],df_romeo["degree"], label="Romeo")
27 plt.plot(df_julia["currentTimeStep"],df_julia["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()
34
35
36 #%%
37 #plt.figure(figsize=(20,20))
38 plt.plot(df["alphaJulia"], df["currentTimeStep"])
39 plt.ylabel("[t] = s")
40 plt.xlabel("[s] = alpha")
41 plt.title("Auslenkung in Rad als Funktion der Zeit")
42 plt.savefig('../Semesterprojekt Physik Engines/images/ropeJulia/Auslenkung.png', dpi=300,
42             bbox_inches='tight')
43 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
14 plt.figure(figsize=(20,20))
15 plt.plot(df["cubeRomeo.position.x"], df["cubeRomeo.position.y"])
16
17 plt.xlabel("Position x [s] = m")
18 plt.ylabel("Position y [s] = m")
19 plt.title("Bewegung Wrfel Romeo")
20 plt.savefig('../Semesterprojekt Physik Engines/images/ropeRomeo/Ortdiagramm.png', dpi=300,
21             bbox_inches='tight')
22 plt.show()
23
24 plt.plot(df["currentTimeStep"],df["degree"])
25 plt.ylabel("[s] = alpha")
26 plt.xlabel("[t] = s")
27 plt.title("Auslenkung in Grad als Funktion der Zeit")
28 plt.savefig('../Semesterprojekt Physik Engines/images/ropeRomeo/AuslenkungDeg.png', dpi=300,
29             bbox_inches='tight')
30 plt.show()
31
32 #%%
33 plt.figure(figsize=(20,20))
34 plt.plot(df["currentTimeStep"],df["cubeRomeo.position.x"])
35 plt.ylabel("[s] = m")
36 plt.xlabel("[t] = s")
37 plt.title("Bewegung Wrfel Romeo x")
38 plt.savefig('../Semesterprojekt Physik Engines/images/ropeRomeo/x(t).png', dpi=300,
39             bbox_inches='tight')
40 plt.show()
41
42 plt.plot(df["currentTimeStep"],df["cubeRomeo.position.y"])
43 plt.ylabel("[s] = m")
44 plt.xlabel("[t] = s")
45 plt.title("Bewegung Wrfel Romeo y")
46 plt.savefig('../Semesterprojekt Physik Engines/images/ropeRomeo/y(t).png', dpi=300,
47             bbox_inches='tight')
48 plt.show()
49
50 plt.figure()
51 plt.plot(df["currentTimeStep"],df["alphaRomeo"])
52 plt.ylabel("[s] = alpha")
53 plt.xlabel("[t] = s")
54 plt.title("Auslenkung in Rad als Funktion der Zeit")
55 plt.ylim(25,54)
56 # plt.xticks([1, 2, 3, 4, 5,6,7,8,9,10,11,12,13])
```

```
54 plt.savefig('..../Semesterprojekt Physik Engines/images/ropeRomeo/AuslenkungRad.png', dpi=300,  
55 bbox_inches='tight')  
55 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	Experiment Übersicht	9
6	Einstellung Julia	10
7	Einstellung Romeo	10
8	Einstellung Feder	11
9	Feder mit markierter Berührungs punkt	11
10	Experiment Übersicht	12
11	Experiment Übersicht	13
12	Impuls Romeo & Julia als Funktion der Zeit	14
13	GesamtImpluls als Funktion der Zeit	15
14	Ort Romeo & Julia als Funktion der Zeit	15
15	Geschwindigkeit Romeo & Julia als Funktion der Zeit	16
16	Endgeschwindigkeit als Funktion der Zeit	16
17	Auslenkung in Grad als Funktion der Zeit	17
18	Romeo & Julia Ortsdiagramm	17

Literatur

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