

Virtual Science Lab

Hochschule Kaiserslautern

Philipp Lauer, Marc Zintel

Vorwort

Der vorliegende Projektbericht gibt einen Überblick über das entworfene Virtual Science Lab. Dabei handelt es sich um eine Virtual Reality Anwendung, die dem Benutzer intuitiv Versuche aus verschiedenen Wissenschaftszweigen veranschaulichen und erklären soll. Als Programmierumgebung wurde hierzu Unity gewählt, die spezifische Hardware, für die die Anwendung optimiert wurde ist die HTC Vive Pro. Die bereits vorhandene Version von den Autoren und Anatoli Schäfer wurde um einige weitere Labore für Elektrotechnik, Biologie, Mathematik und Informatik, zusätzlich zu den bereits vertretenen Chemie- und Physiklaboren ergänzt.

Durch diese Erweiterung offeriert die Anwendung eine größere Anzahl an verschiedenen Versuchen und es besteht die Annahme, dass durch das spielerische und native Ausprobieren, der Nutzer länger bereit ist, sich indirekt über Wissenschaft zu informieren.

Ein Ausblick in die Zukunft zeigt, dass diese Annahme in Tests ausgewertet werden muss und dass eine Steuerung für verschiedenste Hardware-Varianten evaluiert werden muss.

Inhaltsverzeichnis

1	Idee	1
2	Steuerung	3
3	Bisheriger Stand	5
4	Neuerungen	9
4.1	Biologie	11
4.2	Mathematik	15
4.3	Informatik	18
4.4	Teilchenlabor	21
4.5	Elektrotechnik	23
5	Code	27
5.1	Allgemeine Scripte	27
5.1.1	Scene_Management.cs	28
5.1.2	Globals.cs	30
5.1.3	Load_Publics.cs	31
5.1.4	Sev_Seg_Counter.cs	34
5.1.5	Display_Meter_5_D.cs	39
5.2	Szenenspezifische Scripte	44
5.2.1	Beschleunigung.cs	44
6	Verworfene Ideen	47
7	Ausblick	49
8	Fazit	51
	Literaturverzeichnis	53

Kapitel 1

Idee

Der Grundgedanke hinter dem Virtual Science Lab ist es, Wissenschaft möglichst anschaulich und spielerisch zu vermitteln. Dabei kommen die technischen Mittel der Virtual Reality zum Einsatz. Diese soll sicherstellen, dass die Versuche möglichst nativ durchgeführt werden können, das heißt dass beispielsweise Gegenstände durch Hinführen der Hand und anschließendes Zugreifen eines Buttons aufgehoben werden können. Auch ein normales Bewegen im Raum ist möglich, weshalb die Annahme besteht, dass die Einstiegshürde deutlich geringer ist, als beispielsweise eine Steuerung mit Maus und Tastatur oder Gamepad. Auch der visuelle Eindruck soll durch den Einsatz von Virtual Reality gesteigert werden, da man sich frei im Raum drehen und bewegen kann und so zur Erkundung angeregt wird. Das Virtual Science Lab wird zunächst auf der HTC Vive Pro entwickelt und evaluiert.

Kapitel 2

Steuerung

Gesteuert wird die gesamte Anwendung mit den Controllern der HTC Vive Pro. Außerdem wird durch Sensoren im VR-Labor ermöglicht, sich innerhalb des Raumes von drei auf vier Metern (bis zu zehn auf zehn Meter möglich) frei im Raum zu bewegen. Verwendet man die wireless Variante, ermöglicht dies ein sehr freies und angenehmes Erlebnis. Da man in der virtuellen Realität schnell den Überblick verlieren kann, wie weit die jeweiligen Wände des Raumes noch in Wirklichkeit entfernt sind, wird ein Gitter in der Anwendung angezeigt, wenn man sich dieser zu sehr nähert.

Da einer der Räume für die potenzielle Nutzung der maximal möglichen Raumgröße entworfen wurde, ist eine Teleport-Möglichkeit unerlässlich. Diese ist jedoch auch im Flur nötig und in den anderen Räumen möglich. Durch Drücken des Touchpads des Controllers erscheint ein Marker, den man auf die am Boden befindliche Stelle platzieren muss, an die teleportiert werden soll. Durch Loslassen wird man zur gewünschten Stelle gebracht. Die einzige andere Taste, die zur Bedienung verwendet wird, ist der Hairtrigger. Mit diesem lassen sich Gegenstände durch Berührung und gleichzeitiges gedrückt Haltens des Triggers hochheben. Ein Teleport mit einem aufgehobenen Gegenstand im selben Controller ist nicht möglich, dazu sind beide Controller notwendig. Mit einem muss man den Gegenstand mit dem Hairtrigger aufnehmen und halten, mit dem anderen wird über das Touchpad teleportiert.

Kapitel 3

Bisheriger Stand

Die Ausgangslage zu Projektstart, bildete die Vorarbeit im Fach ?Augmented und Virtual Reality?, bei dem das Virtual Science Lab zusammen noch mit Anatoli Schäfer erste Züge angenommen hat. In diesem Kontext wurden 4 Laborräume kreiert, sowie eine Outdoor Area, die lediglich zur Erkundung im virtuellen Raum dienen sollte. Als Startpunkt des Projektes diente ein Nachbau des Virtual Reality Labor der Hochschule Kaiserslautern, Standort Zweibrücken. Da das Projekt voraussichtlich dort durchgeführt werden sollte, zieht der User die Brille an und befindet sich danach im gleichen Raum, in dem er ohne Brille gestanden hat ? nur eben virtuell. Des Weiteren bekam das Projekt ein Chemielabor, sowie zwei Laborräume mit Versuchen die eine Mischung zwischen Physik und Chemie zeigen. Diese waren ein Versuch zur Flammenfärbung, ein Leitbarkeitstest und eine Reaktion zwischen zwei Stoffen (Müllverarbeitung). Die letzten beiden Räume wurden durch einen gewinkelten Gang miteinander verbunden, der lediglich dem Zweck dienen sollte. In allen Räumen sind Erklärungen zu den Versuchen in verschiedener Form platziert (Klemmbrett, Beamer, Plakate).

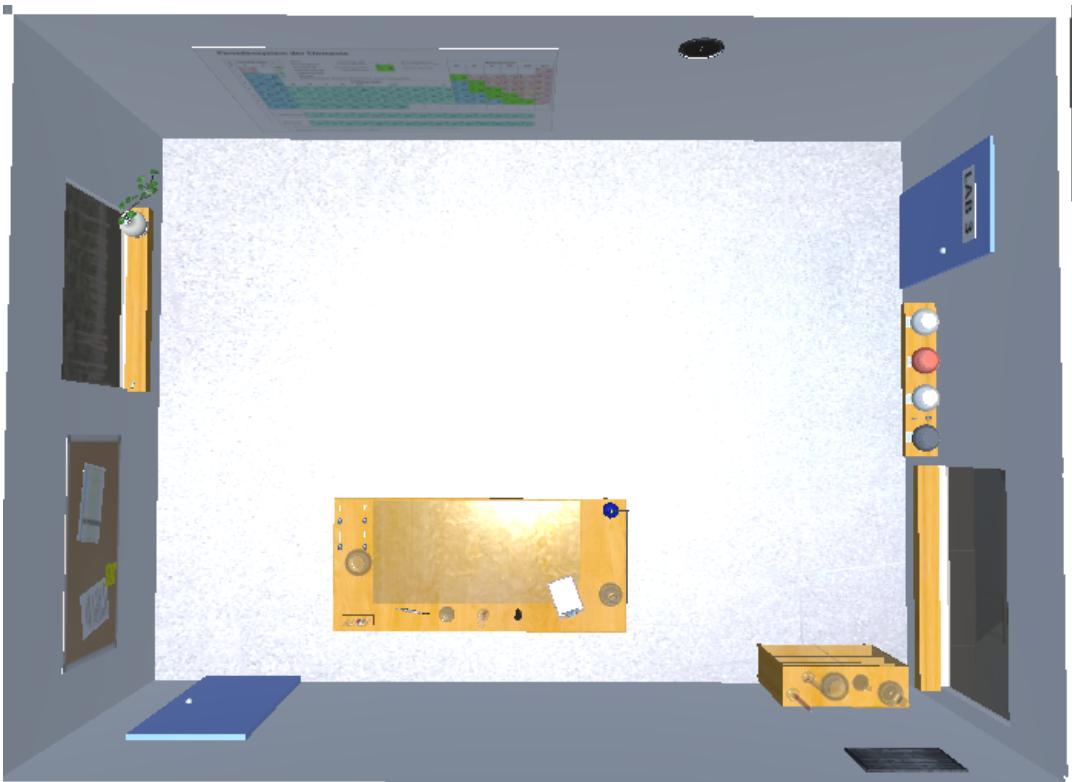


Abbildung 3.1: Grundriss Chemielabor

Flammenfärbung:

Im Raum befinden sich verschiedene Löffel, ein Bunsenbrenner (auf Grund fehlender Modelle ein Zylinder mit Knopf) und verschiedene Stoffe auf einer Ablage (Calcium, Kalium, Lithium und Kupfer). Man nimmt einen Löffel, nimmt einen beliebigen Stoff auf, schaltet den Bunsenbrenner an und hält den Löffel in die Flamme. Anschließend betrachtet man die gefärbte Flamme

Müllverarbeitung:

Der Raum ist angefüllt mit Körpern aus Styropor. In der Mitte des Raumes befindet sich ein Tisch mit einer Schale voll Aceton. Nimmt man die Styroporkörper auf und wirft sie in die Schale lösen sie sich nach einiger Zeit auf.

Leitbarkeitstest:

Auf einem Tisch befindet sich ein Behältnis mit Wasser. Durch das Gefäß läuft ein Draht der einen Lichtschalter und eine Glühbirne verbindet.

Da destilliertes Wasser keinen Strom leitet bewirkt das Tasten des Schalters im Ausgangszustand nichts. Erst muss ein leitbarer Stoff, in diesem Fall Salz dem Wasser hinzugegeben werden. Anschließend lässt sich die Lampe einschalten.

Das Git-Repository (vgl. [Zin]) des bestehenden Projektes wurde fortgesetzt. Dort lässt sich neben der aktuellsten Version auch der Ausgangsstand vor Projektbeginn herunterladen. Dieses ist die Version 1.0 vom 15. August 2019, zu finden im Projekt unter Releases.

Kapitel 4

Neuerungen

Da bereits erste Erfahrungen mit Unity gemacht werden konnten, wurden zu Beginn des Projektes die Ziele höhergesteckt als bei der bisherigen Ausführung. Neben vielen Wissenschaftszweigen wie Mathematik, Informatik, Biologie etc. sollte auch ein Android Build erstellt werden, der auf preiswerten Geräten ausgerollt werden sollte. Die Idee dahinter ist es, dass in einem Klassenraum jeder für sich einen Versuch selber durchführen soll, statt lediglich aus mehreren Metern entfernt zuzuschauen, wie der Lehrer den Versuch zum zigsten Mal in seiner Laufbahn macht.

Außerdem sollte der bisherige Aufbau insgesamt etwas abgeändert werden, sowie bestehende Bugs behoben werden. Ideen kamen viele, meist bestand das Problem in der bildlichen Vorstellung. Wie soll ein theoretischer Versuch möglichst anschaulich in VR umgesetzt werden und anschließend sinnvoll und nativ durchführbar sein. Die Outdoor Area wurde komplett entfernt, ebenso wie der gewinkelte Gang zwischen dem Leitbarkeitstest und der Müllverarbeitung.

Die ganzen Labore sollten durch einen einzelnen breiten Flur mit Türen zu jedem Versuch zugänglich werden. Leitbarkeitstest und Müllverarbeitung wurden zusätzlich zusammengelegt unter dem Oberbegriff ?Physik?. Das Chemielabor hat ein richtiges Bunsenbrenner-Modell erhalten. Das VR- Lab, das vorher keinen eigenen Versuch erhalten hatte, bekam nun einen eigenen Versuch, beziehungsweise eher ein bekanntes Knobelspiel: *Die Türme von Hanoi* in Lebensgröße. Die neu gefertigten Räume mit den dazugehörigen Versuchen werden im Folgenden mit den aufgetretenen Problemen veranschaulicht:



Abbildung 4.1: Grundriss Physiklabor



Abbildung 4.2: Neuer Flur mit Türen zum jeweiligen Laborraum



Abbildung 4.3: Grundriss Virtual Reality Labor

4.1 Biologie

Das Biologie Labor hat einen Mikroskop-Versuch. Auf einem Tisch im Raum liegen drei Glasplatten mit verschiedenen Inhalten (Blut, Pflanze, Textil). Diese Platten kann man aufnehmen und in das Mikroskop [2], das auf dem danebenstehenden Tisch (soll zur Bewegung im Raum animieren) steht einlegen. Klickt man nun erneut auf das Mikroskop gelangt man in das zu betrachtende Material. Durch eine Falltür auf dem Boden kann man wieder zurück ins Labor gelangen. Die Falltür soll dazu dienen, dass man auch die Höhe des Raumes mit einbezieht und der User gezwungen ist, sich zu bücken.

In diesem Labor bestand ein Problem darin, die Szenen für Pflanze, Blut und Textil zu gestalten, da Unity nicht von sich aus eine Sphere von innen texturiert. Das heißt, wenn man eine große Sphere als Art Kuppel anlegen will, funktioniert dies nicht einfach, indem man die Kamera innerhalb der Sphere platziert. Nach einer Recherche mit einigen Umwegen war die Lösung sehr

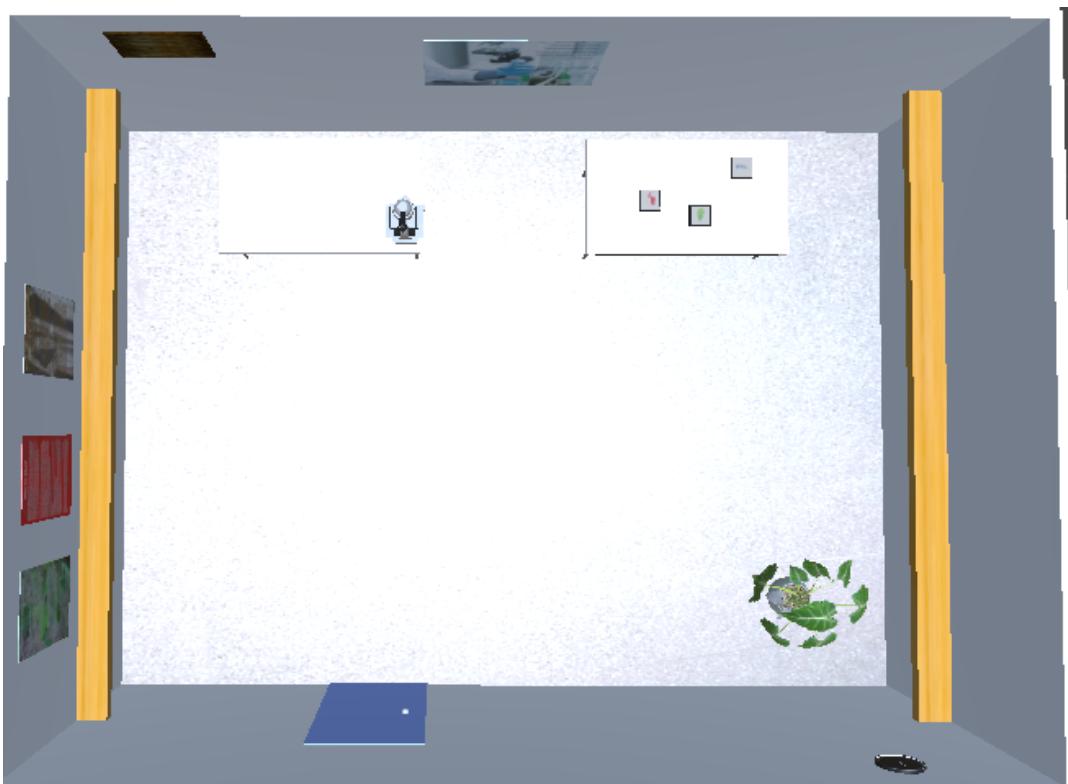


Abbildung 4.4: Grundriss Biologielabor



Abbildung 4.5: Biologielabor mit Mikroskop und Glasplatten

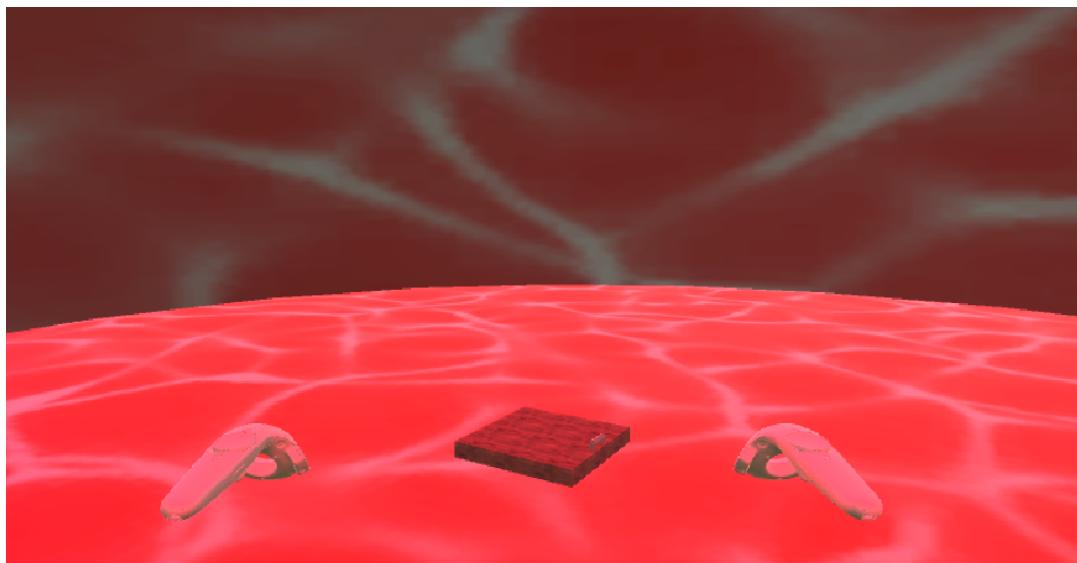


Abbildung 4.6: Biologielabor ? Blutzelle

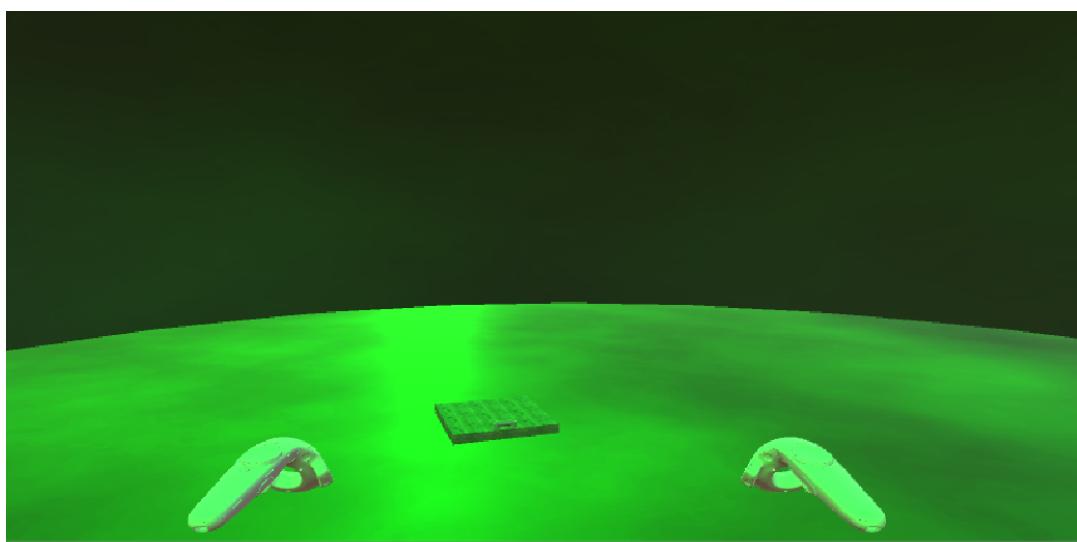


Abbildung 4.7: Biologielabor ? Pflanzenzelle

simpel, es wurde ein Double Side Shader statt dem Standard Shader ausgewählt, sowie die Belichtung angepasst.

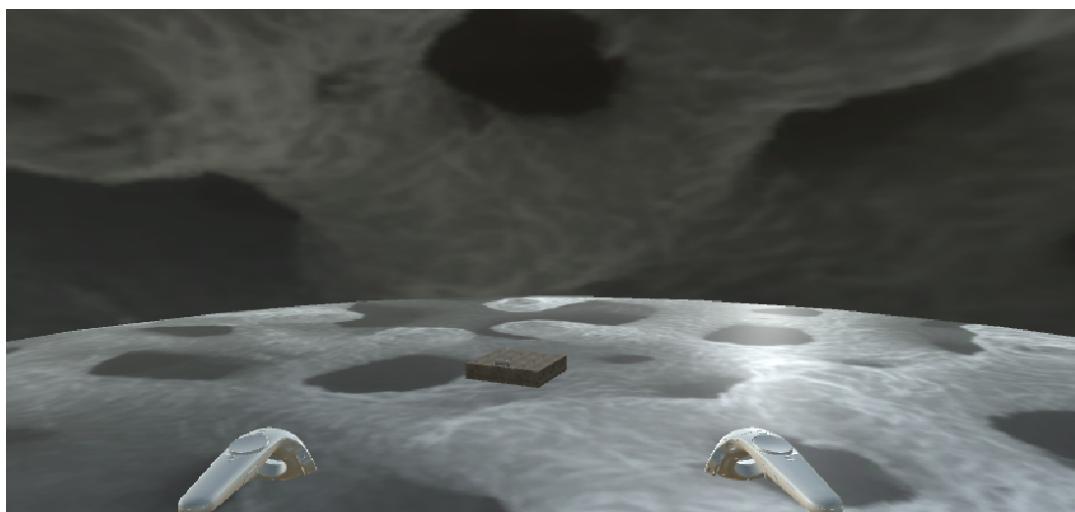


Abbildung 4.8: Biologielabor ? Textilfaser

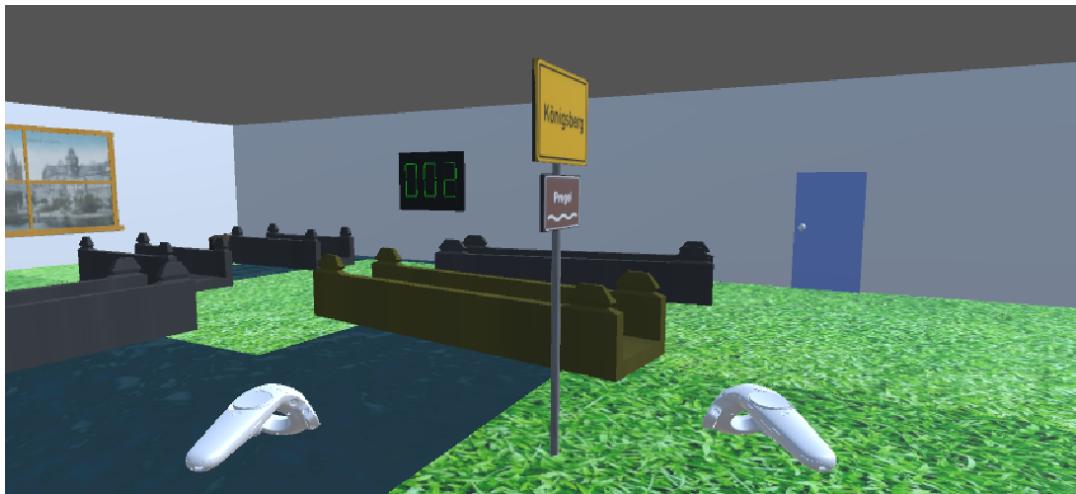


Abbildung 4.9: Königsberger Brückenproblem mit geänderter Brückenfarbe (bereits benutzt) und Zähler

4.2 Mathematik

Ein Raum des Projektes sollte planmäßig die bisherigen Maße von 3x4 Meter durch 10x10 Meter ersetzen, sodass auch Vorführungen in einem größeren Raum sinnvoll sind. Dabei fiel die Wahl auf das Mathematiklabor. In diesem wurde die Darstellung der Kardioiden-Funktion aus der Outdoor Szene recyclet, welche man nun mit der dazugehörigen Gleichung durch ein Fenster außerhalb des Raumes betrachten kann.

Außerdem wurde das Königsberger Brückenproblem raumfüllend gestaltet. Dem Benutzer ist es also möglich, sich frei innerhalb des Raumes zu bewegen, um zu versuchen das nicht lösbarer Problem zu lösen. Die Lösung ist etwas versteckt in Form eines ?toten Briefkastens?, der sich unter ein paar Steinen befindet.

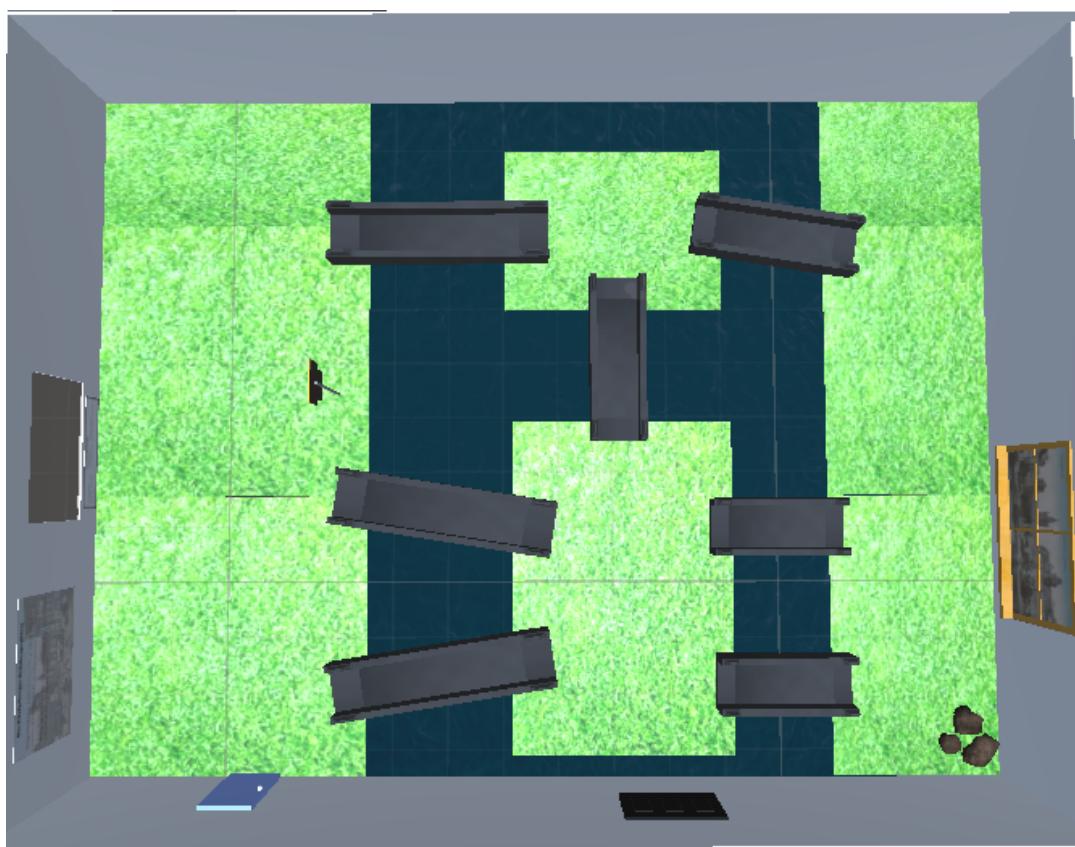


Abbildung 4.10: Grundriss Mathematiklabor

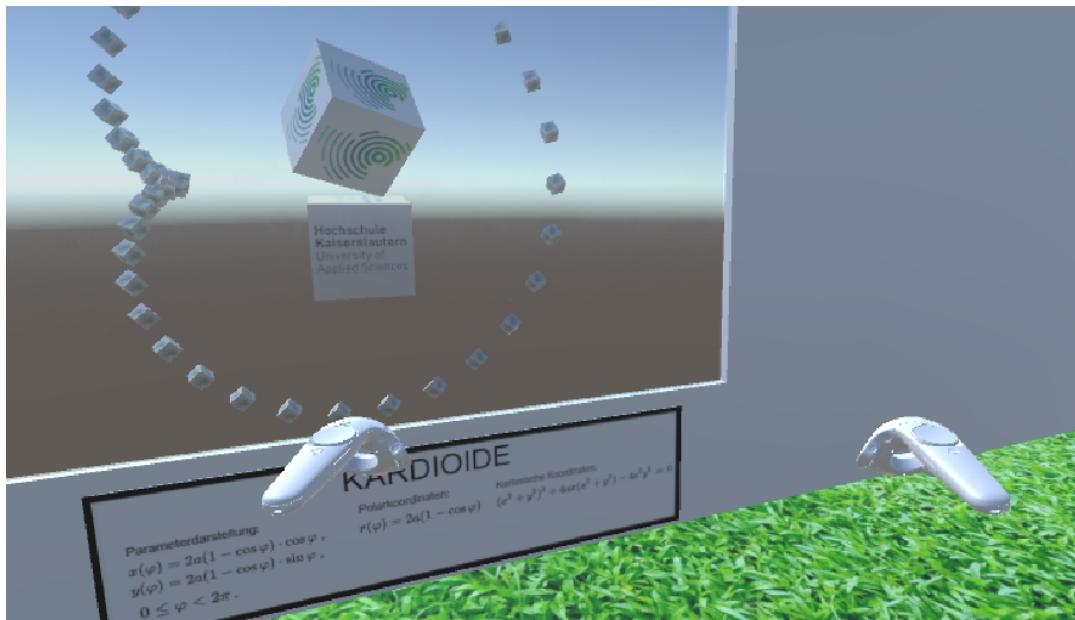


Abbildung 4.11: Karidoide am Fenster im Raum



Abbildung 4.12: Steine im Raum als toter Briefkasten



Abbildung 4.13: Grundriss Informatiklab

4.3 Informatik

Das Informatiklabor bietet zwei Versuche, bei denen das angesprochene Problem der schwierigen Visualisierung am stärksten aufgetreten ist. Diese sind der Suchalgorithmus Bubble Sort und der Dijkstra-Algorithmus, der den kürzesten Weg innerhalb eines Graphen aufzeigt.

Die Wahl ist schließlich so ausgefallen, dass der Bubble Sort in einer Box auf einem Tisch dargestellt wird, welche mit Cubes mit zugehörigen Zahlenwerten gefüllt ist. Dabei werden immer zwei Fächer der Box gleichzeitig geöffnet, sodass man gegebenenfalls die beiden Elemente tauschen muss, falls ein kleinerer Wert rechts eines größeren liegt. Dies wird solange durchgeführt, bis alle Cubes an der richtigen Stelle platziert sind. Visuelles Feedback bekommt der User durch Färben der Blöcke zu grün, sobald sie an der richtigen Position liegen.

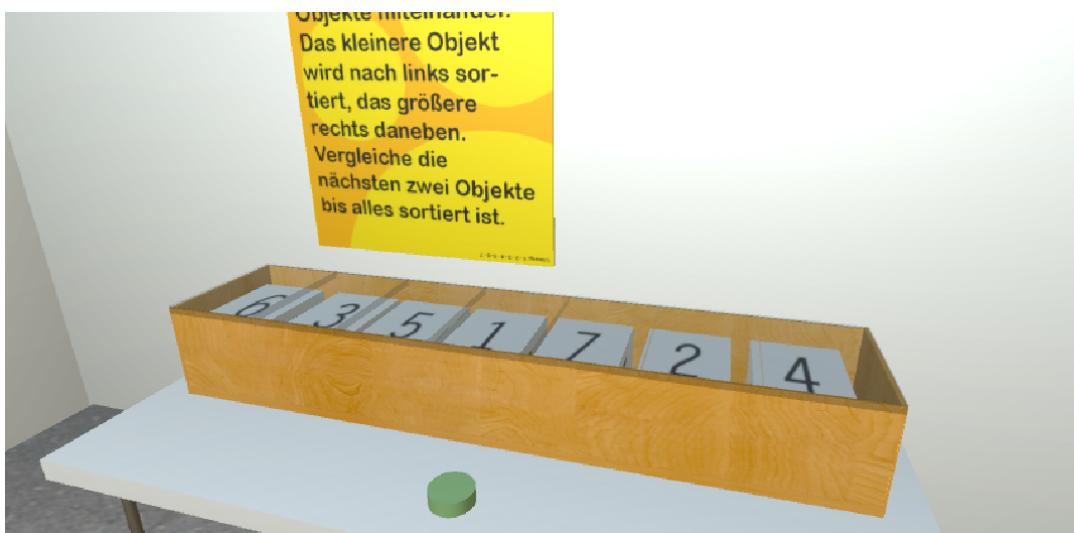


Abbildung 4.14: Bubble Sort mit Button um nächstes Fach zu öffnen

Der Dijkstra-Algorithmus ist an der Wand platziert und die einzelnen Knotenpunkte sind durch Berührung des Controllers abzugehen. Dabei erscheint ebenso ein virtuelles Feedback. Knoten, die bereits besucht wurden, sind grün, mögliche Knoten im nächsten Schritt werden gelb hinterlegt. Ein Sieben-Segment Zähler rechnet die Werte der abgelaufenen Pfade zusammen. Wird die Zahl rot ist man über den Wert des kürzesten Weges gekommen. Erscheint sie grün hat man erfolgreich den kürzesten Weg absolviert. Will man den Zähler zurücksetzen wählt man den Reset-Knopf und man kann von vorne beginnen.

Hier sind einige seltsame Probleme aufgetreten, die sich auch im Nachhinein nicht erklären lassen. Bei Tests im Simulator und mit der HTC Vive Pro kommen teils verschiedene Ergebnisse hervor. Der Bubble Sort wirft mit der Brille nach einer bestimmten Zeit einen Stack-Overflow Fehler auf, bei dem nicht klar zu erkennen ist, wieso dieser entsteht. Der Counter hat auch in beiden Ausführungen verschiedene Probleme gehabt, teils ging er nur im Simulator, teils nur mit Brille, teils weder noch. Diese Probleme werden, um es leicht zu machen einfach auf kleine Bugs von der Seite von Unity oder des HTC Vive Plugins geschoben. Anders sind diese (für uns) nicht erklärbar.

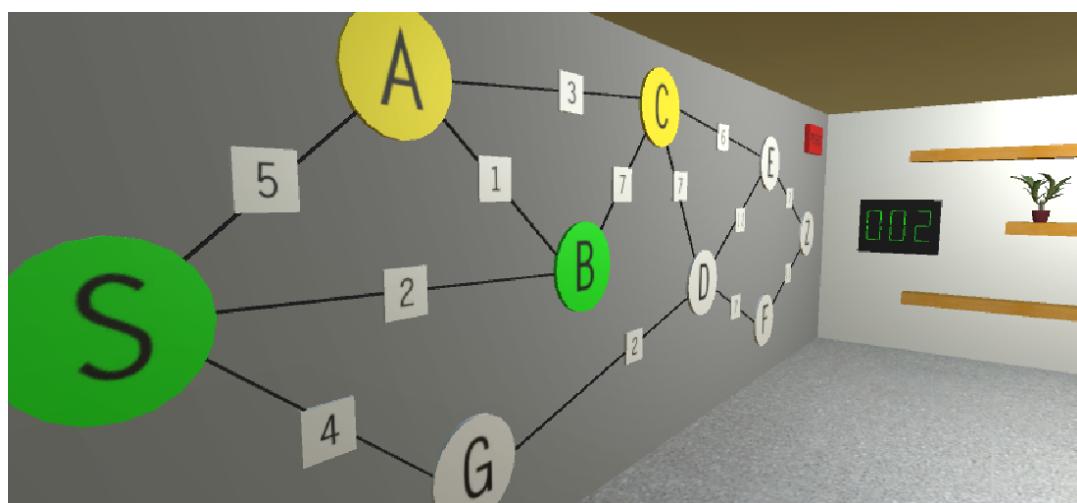


Abbildung 4.15: Dijkstra-Algorithmus, grüne Knoten besucht, gelbe Knoten besuchbar, Zähler und Reset-Button

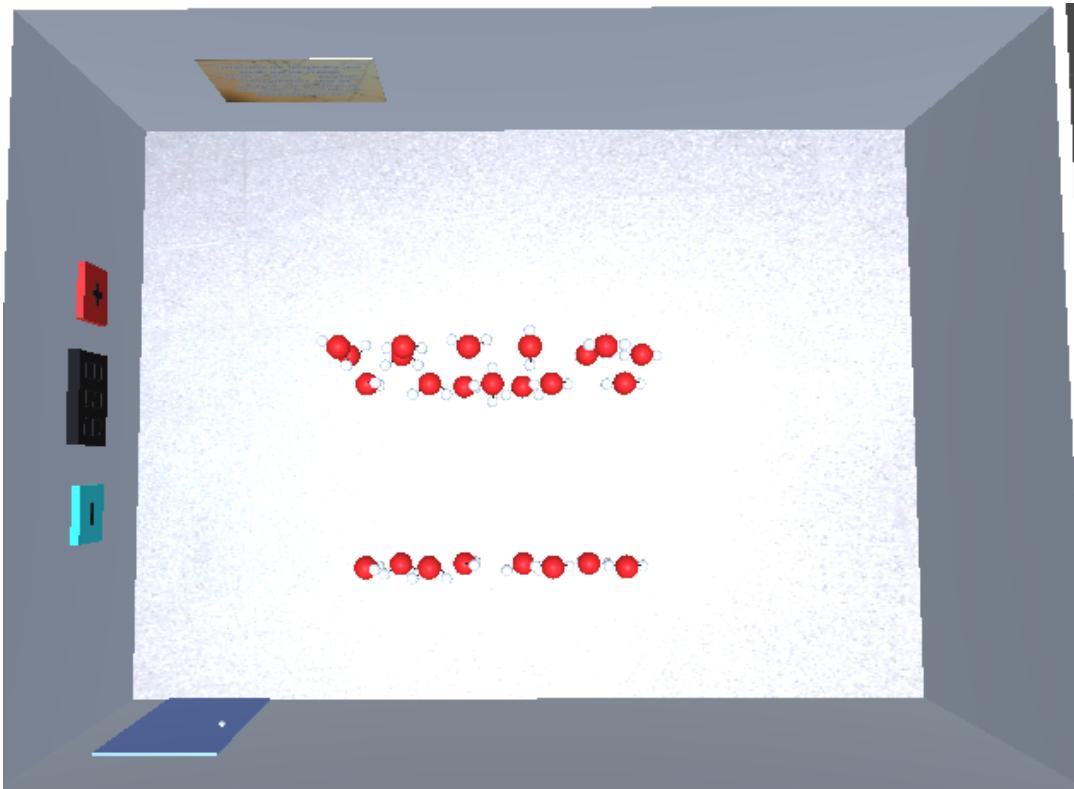


Abbildung 4.16: Grundriss Teilchenlabor

4.4 Teilchenlabor

Das Teilchenlabor hat auf einer relativ spontanen Idee basiert und soll zeigen, wie das Verhalten von einzelnen Wassermolekülen (H_2O) bei sich verändernder Raumtemperatur ist. Dazu fliegen bei Betreten des Raums die Moleküle von Wand zu Wand von denen sie sich abstoßen. Auch Kollisionen miteinander führen zu einem Richtungswechsel.

Dabei hat sich als Problem herausgestellt, dass Mesh-Collider in den aktuellen Unity-Versionen nicht mehr unterstützt werden, wodurch die Moleküle zu Beginn immer durch die Wände geflogen sind und langsam, aber sicher verschwanden. Ein möglicher Lösungsansatz war es, um das gesamte Molekülgebilde einen Cube-Collider zu legen, sodass die Moleküle sich gegenseitig und auf die Cube-Collider des Raumes reagieren können.

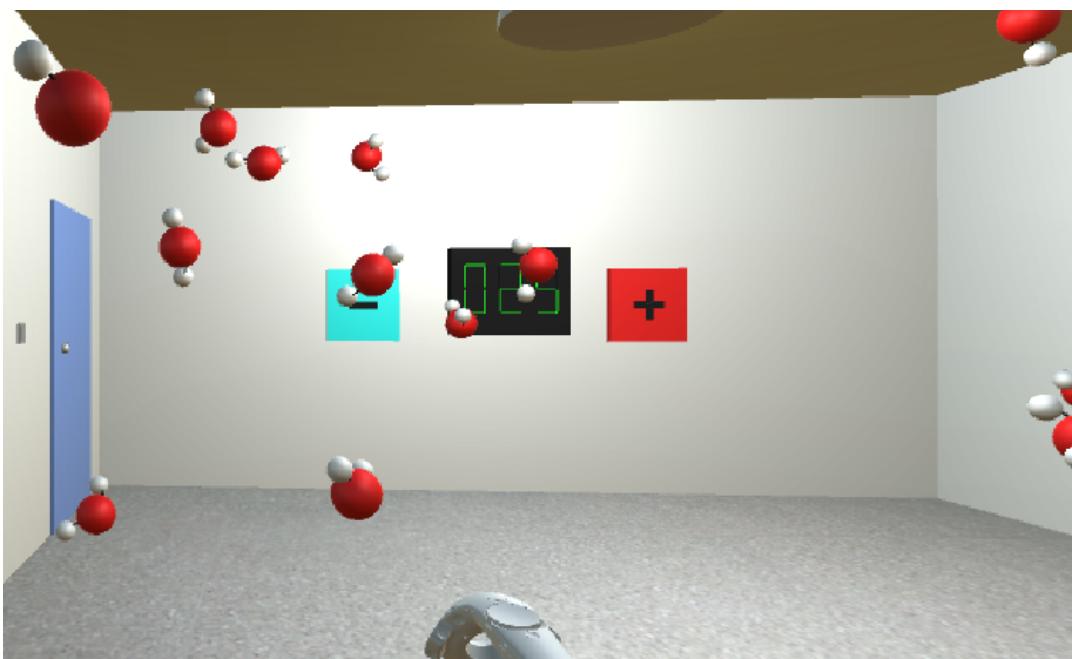


Abbildung 4.17: Teilchenlabor mit Display und Temperatursteuerung

Ein kleineres weiteres Problem ist es, dass es ein sehr unangenehmes Gefühl ist, wenn ein Molekül direkt auf den User zufliegt und einen am Kopf treffen würde. Dafür wurde jedoch keine passable Lösung gefunden.

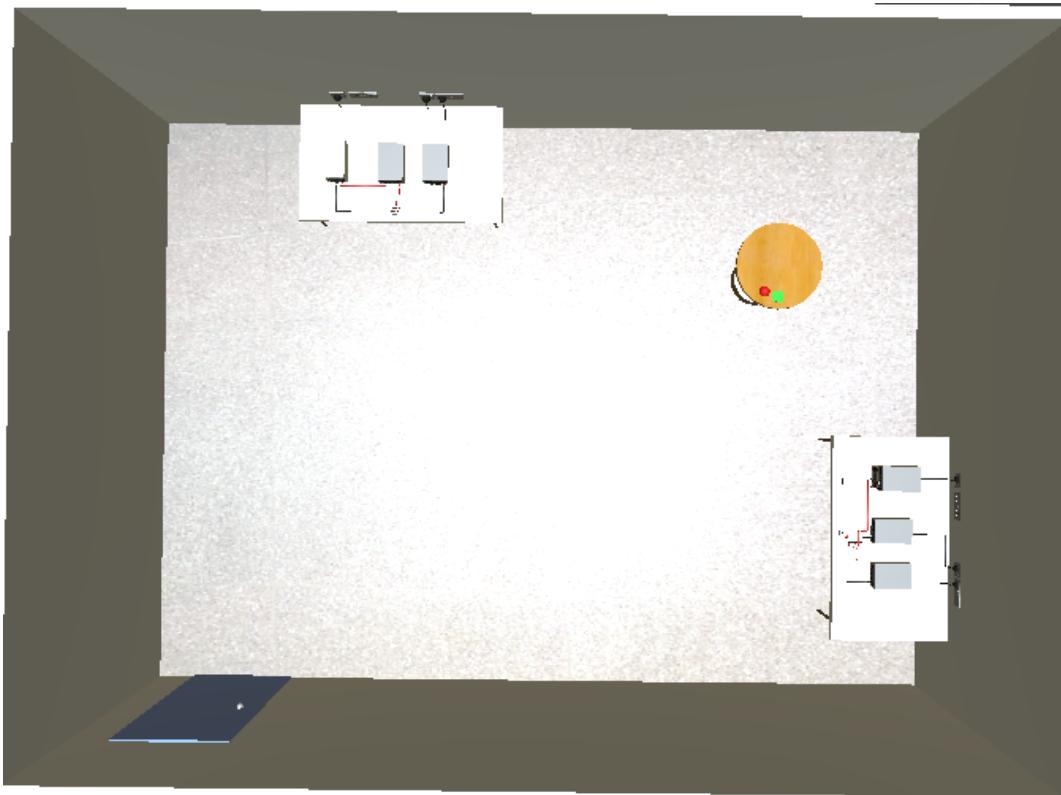


Abbildung 4.18: Grundriss Elektrotechniklabor

4.5 Elektrotechnik

Die beiden Versuche der Elektrotechnik sind raumtechnisch im Hinterzimmer des Physiklabors. Das liegt ganz einfach daher, dass ansonsten eine gerade Anzahl an Räumen im Flur entstanden wäre, was nicht aufteilbar gewesen wäre und daher wurde dies fachlich der Physik ?untergeordnet?.

Zu diesen Versuchen wurde auf tatkräftige Unterstützung von Herr Dr.-Ing. Hubert Zitt gebaut, der für uns die Versuche im Elektrotechnik-Labor der Hochschule Kaiserslautern in Realität aufgebaut hat und uns genau beschrieb, welche Vorgänge umgesetzt werden sollen und was veranschaulicht werden soll.

Das größte Problem hierbei waren vor allem die Kabel in VR umzusetzen. Es wäre prinzipiell schöner gewesen, wenn ein User die Buchsen der Geräte

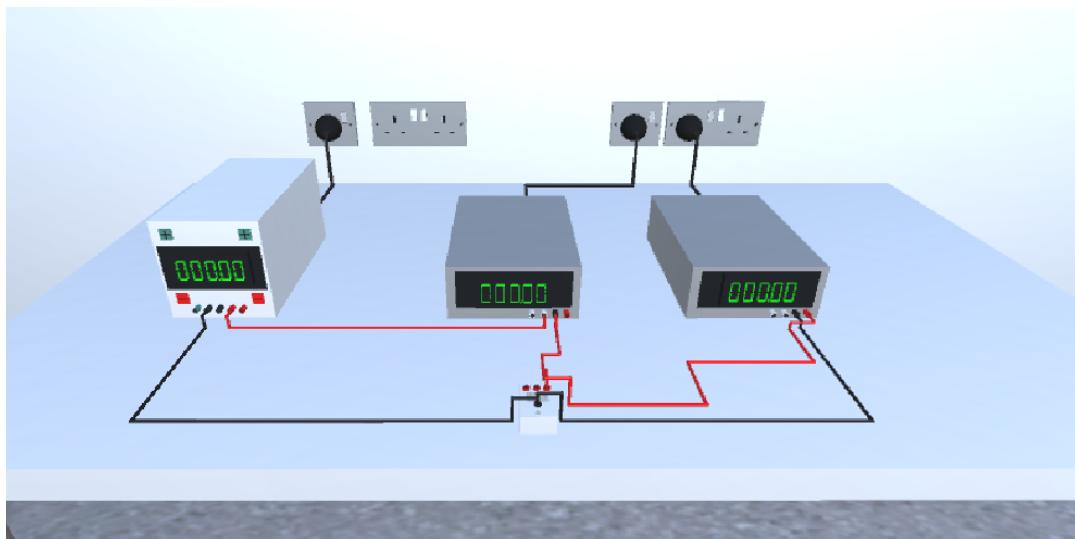


Abbildung 4.19: Versuch: Messen von Strom und Spannung bei einem nicht-linearen Verbraucher

selbst verkabeln könnte, dies gab die Zeit und auch unsere Erfahrung mit Unity nicht her, solch ein Vorhaben würde den vorgegebenen Rahmen um ein Vielfaches sprengen vermutlich. Daher wurde die Wahl auf statische Kabel getroffen, welche aus etlichen Zylindern einzeln zusammengebaut wurden. Ein weiteres Problem war es, an die benötigten Formeln heranzukommen, da es hierbei hin und wieder zu Missverständnissen kam. Gelöst wurde dieses Problem durch zur Hilfenahme von Interpolation.

Bei dem ersten Versuch wird Strom und Spannung bei einem nichtlinearen Verbraucher gemessen. [Zita] Der zweite Versuch veranschaulicht das Messen von Spannungen bei einer RC-Reihenschaltung. [Zitb]

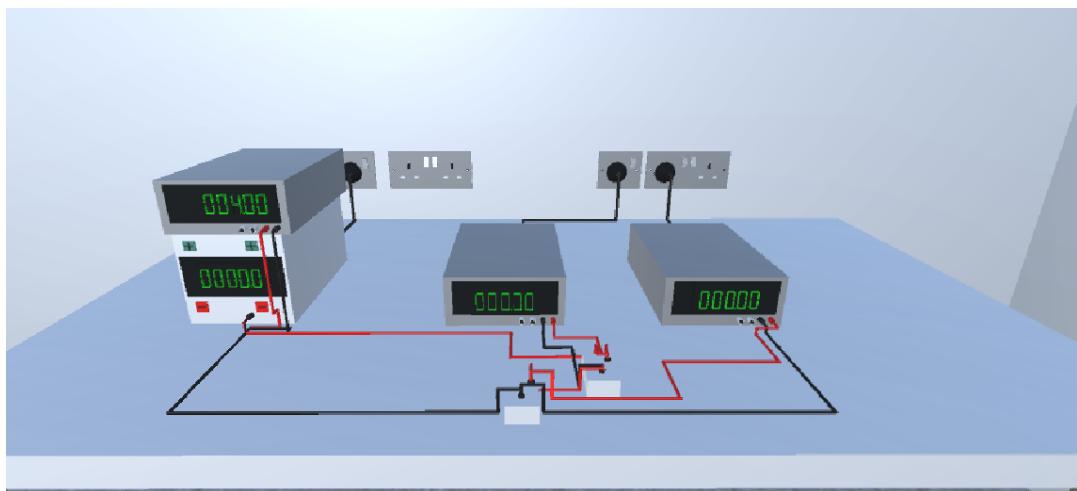


Abbildung 4.20: Versuch: Messen von Spannungen bei einer RC-Reihenschaltung

Kapitel 5

Code

5.1 Allgemeine Scripte

Viele Funktionen sind in Unity standardmäßig implementiert. So zum Beispiel die Möglichkeit physikalische Eigenschaften auf ein Objekt zu definieren. Dies macht es anfassbar und ermöglicht die Interaktion mit dem Objekt.

Eigene bzw. weitere und schwierigere Funktionen müssen hingegen selbst implementiert werden. Alle von uns implementierten Funktionen und Methoden befinden sich im Projekt unter Assets -> Scripts. Alle Scripte wurden in C# programmiert.

Es gibt Scripte, die sich durchs ganze Projekt ziehen und andere, die sich auf bestimmte Szenen beziehen. Allgemein können jedoch alle Scripte überall verwendet werden.

5.1.1 Scene_Management.cs

Das Scene Management Script dient zur richtigen Positionierung des Spielers, wenn er den Flur betritt. Je nachdem aus welchem Raum er kommt ändert sich die Startposition. Das Script muss in jeder Szene eingebaut sein, da nur so die zuletzt verwendete Szene ausgelesen werden kann.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class Scene_Management : MonoBehaviour {
    GameObject camera_obj;

    // Use this for initialization
    void Start () {
        Scene scene = SceneManager.GetActiveScene();
        camera_obj = GameObject.Find("ViveRig");
        Vector3 vec = new Vector3();

        if (scene.name == "NeuerFlur" && Globals.last_scene != "") {
            Debug.Log("Switching Cam from: " + Globals.last_scene);
            switch (Globals.last_scene) {
                case "BioLab":
                    vec = new Vector3(12.0f, 0f, -1.4f);
                    break;
                case "grosserRaum": // Mathe
                    vec = new Vector3(9.2f, 0f, 0f);
                    break;
                case "Lab1": // Chemie
                    vec = new Vector3(22.0f, 0f, -1.4f);
                    break;
                case "Lab2": // Physik
                    vec = new Vector3(22.0f, 0f, 1.4f);
                    break;
                case "Lab3": // Informatik
                    vec = new Vector3(16.9f, 0f, 1.4f);
                    break;
                case "Teilchenlabor":

```

```
        vec = new Vector3(12.0f, 0f, 1.4f);
        break;
    case "VRLab":
        vec = new Vector3(16.9f, 0f, 1.4f);
        break;
    }
    Debug.Log("Changing Position: " + vec);
    camera_obj.transform.position = vec;
} else
{
    Globals.last_scene = scene.name;
}
}

// Update is called once per frame
void Update () {

}

}
```

Das Script wird in der *start()* Methode ausgeführt - also beim Laden der Szene. Am Anfang wird der Szenenname rausgefunden und in einer Variable gespeichert. Anschließend erfolgt eine Abfrage, ob die aktuelle Szene der Flur ist oder nicht. Falls nicht, wird der aktuelle Scenenname als *last_scene* in einer Globalen Variable gespeichert. Wenn es sich um den Flur handelt wird die Position des Players entsprechend der letzten Szene im Flur geändert.

5.1.2 Globals.cs

In der Globals.cs werden globale Variablen gespeichert, die Szenenübergreifend benötigt werden. Aktuell ist dies nur der letzte Szenenname.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public static class Globals {

    public static string last_scene = "";
}
```

5.1.3 Load_Publics.cs

In diesem Script werden alle Variablen gespeichert, die über ein Script hinaus aber nur innerhalb einer Szene benötigt werden. Das Script wird in jeder Szene eingebunden. Das Objekt auf dem es positioniert ist, spielt dabei keine Rolle.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Load_Publics : MonoBehaviour {

    //Allgemein
    public static bool light_on = false;
    public static bool light_col = false;
    public static string[] lightnames = {"Point_Light_l", "Point_Light_m", "P

    // Informatiklab
    // Dijkstra
    internal static readonly bool reset_clicked;
    public static string Dijkstra_Word;
    public static bool s_active = true;
    public static bool a_active = false;
    public static bool b_active = false;
    public static bool g_active = false;
    public static bool c_active = false;
    public static bool d_active = false;
    public static bool e_active = false;
    public static bool f_active = false;
    public static bool z_active = false;
    public static bool r_active = false;

    public static bool s_clicked = false;
    public static bool a_clicked = false;
    public static bool b_clicked = false;
    public static bool g_clicked = false;
    public static bool c_clicked = false;
    public static bool d_clicked = false;
    public static bool e_clicked = false;
    public static bool f_clicked = false;
```

```
public static bool z_clicked = false;
public static bool r_clicked = false;

public static int counter = 0;
public static int maximum = 19;
public static string last_clicked = "";

// Bubblesort
public static int b_state = 0;
public static bool bubble_active = true;
public static bool s_1_act = true;
public static bool s_2_act = true;
public static bool s_3_act = true;
public static bool s_4_act = true;
public static bool s_5_act = true;
public static bool s_6_act = true;
public static bool s_7_act = true;

// BioLab
public static string scene_change = "";
public static bool bio_collision_happened = false;

// Teilchenlabor
// Molekuele
public static int Temperatur = 25;
public static bool min_act = true;
public static bool plus_act = true;
public static float Temp_Max = 200f;
public static float Temp_Min = 0f;
public static float Map_Temp_Max = 0.25f;
public static float Map_Temp_Min = 0f;
public static float move_speed_multi = 15;
public static float RemapTemp(float from, float fromMin, float fromMax,
{
    var fromAbs = from - fromMin;
    var fromMaxAbs = fromMax - fromMin;
    var normal = fromAbs / fromMaxAbs;
    var toMaxAbs = toMax - toMin;
    var toAbs = toMaxAbs * normal;
    var to = toAbs + toMin;
```

```
        return to;
    }

    // Mathelab
    public static int sev_bridges_counter = 0;
    public static bool bridges_active = true;

    // Elektrolab
    public static double lampe_netzteil_count = 0;
    public static float RemapLight(float from, float fromMin, float fromMax,
    {
        var fromAbs = from - fromMin;
        var fromMaxAbs = fromMax - fromMin;
        var normal = fromAbs / fromMaxAbs;
        var toMaxAbs = toMax - toMin;
        var toAbs = toMaxAbs * normal;
        var to = toAbs + toMin;

        return to;
    }

    // Versuch 2
    public static float Uq = 4.0f;
    public static float R = 2000f;
    public static float C = 0.5f;
    public static float frequency_Netzteil = 0.0f;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

Wichtig ist, dass in dem Script eine Klasse definiert ist, die die Variablen und für die Variablen wichtige Funktionen z. B. zum Mapping von manchen Variablen enthält.

5.1.4 Sev_Seg_Counter.cs

Die 7-Segment-Anzeige wird in verschiedenen Szenen benutzt. Dieses Script dient dazu eine eingegebenen Zahl auf das 3-Ziffern-Display zu übertragen. Der Funktion `setSevSegCount()` wird eine Zahl als int und der Name des zugehörigen Parent-Objekts übergeben. Die Zahl wird als String formatiert, so dass sie besser in 3 Teile geteilt werden kann. Das Parent-Objekt wird per `Find` Befehl gesucht und in einer Variable gespeichert. Anschließend wird für jede der 3 Zeichen des Zahl-Strings die Funktion `set_n()` aufgerufen, welche die entsprechende Position in der 7-Segment-Anzeige in die Zahl umwandelt.

Innerhalb der `set_n()` Methode werden die Segment-Objekte innerhalb des Parent-Objekts gesucht. Da die Benennung immer gleich ist, kann dies programmatisch anhand eines zusammengesetzten Strings geschehen. Anschließend wird für jedes Segment der Anzeige die Farbe entsprechend einer Vorgabe, die per If-Abfrage anhand der eingegebenen Zahl festgestellt wird, entweder Schwarz oder Rot bzw. Grün gesetzt. Dadurch entsteht die Zahl auf der Anzeige.

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Sev_Seg_Counter : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }

    public void setSevSegCount(int seconds, String parentname)
    {
        if (seconds < 1000)
        {
```

```
        string s_number = String.Format("{0:000}", seconds);
        GameObject parent = GameObject.Find(parentname);

        Debug.Log(s_number);

        set_n(int.Parse(Char.ToString(s_number[0])), 1, parent);
        set_n(int.Parse(Char.ToString(s_number[1])), 2, parent);
        set_n(int.Parse(Char.ToString(s_number[2])), 3, parent);
    }
}

private void set_n(int number, int disp_num, GameObject parent_item)
{
    string s_lt = "n_" + disp_num + "_lt";
    string s_rt = "n_" + disp_num + "_rt";
    string s_lb = "n_" + disp_num + "_lb";
    string s_rb = "n_" + disp_num + "_rb";
    string s_m = "n_" + disp_num + "_m";
    string s_b = "n_" + disp_num + "_b";
    string s_t = "n_" + disp_num + "_t";

    Debug.Log(parent_item);
    Debug.Log(number);

    switch (number)
    {
        case 0:
            setColorP(parent_item.transform.Find(s_lt).gameObject);
            setColorP(parent_item.transform.Find(s_rt).gameObject);
            setColorP(parent_item.transform.Find(s_lb).gameObject);
            setColorP(parent_item.transform.Find(s_rb).gameObject);
            setColorN(parent_item.transform.Find(s_m).gameObject);
            setColorP(parent_item.transform.Find(s_b).gameObject);
            setColorP(parent_item.transform.Find(s_t).gameObject);
            break;
        case 1:
            setColorN(parent_item.transform.Find(s_lt).gameObject);
            setColorP(parent_item.transform.Find(s_rt).gameObject);
            setColorN(parent_item.transform.Find(s_lb).gameObject);
            setColorP(parent_item.transform.Find(s_rb).gameObject);
            setColorN(parent_item.transform.Find(s_m).gameObject);
            setColorN(parent_item.transform.Find(s_b).gameObject);
    }
}
```

```
    setColorN(parent_item.transform.Find(s_t).gameObject);
    break;
case 2:
    setColorN(parent_item.transform.Find(s_lt).gameObject);
    setColorP(parent_item.transform.Find(s_rt).gameObject);
    setColorP(parent_item.transform.Find(s_lb).gameObject);
    setColorN(parent_item.transform.Find(s_rb).gameObject);
    setColorP(parent_item.transform.Find(s_m).gameObject);
    setColorP(parent_item.transform.Find(s_b).gameObject);
    setColorP(parent_item.transform.Find(s_t).gameObject);
    break;
case 3:
    setColorN(parent_item.transform.Find(s_lt).gameObject);
    setColorP(parent_item.transform.Find(s_rt).gameObject);
    setColorN(parent_item.transform.Find(s_lb).gameObject);
    setColorP(parent_item.transform.Find(s_rb).gameObject);
    setColorP(parent_item.transform.Find(s_m).gameObject);
    setColorP(parent_item.transform.Find(s_b).gameObject);
    setColorP(parent_item.transform.Find(s_t).gameObject);
    break;
case 4:
    setColorP(parent_item.transform.Find(s_lt).gameObject);
    setColorP(parent_item.transform.Find(s_rt).gameObject);
    setColorN(parent_item.transform.Find(s_lb).gameObject);
    setColorP(parent_item.transform.Find(s_rb).gameObject);
    setColorP(parent_item.transform.Find(s_m).gameObject);
    setColorN(parent_item.transform.Find(s_b).gameObject);
    setColorN(parent_item.transform.Find(s_t).gameObject);
    break;
case 5:
    setColorP(parent_item.transform.Find(s_lt).gameObject);
    setColorN(parent_item.transform.Find(s_rt).gameObject);
    setColorN(parent_item.transform.Find(s_lb).gameObject);
    setColorP(parent_item.transform.Find(s_rb).gameObject);
    setColorP(parent_item.transform.Find(s_m).gameObject);
    setColorP(parent_item.transform.Find(s_b).gameObject);
    setColorP(parent_item.transform.Find(s_t).gameObject);
    break;
case 6:
    setColorP(parent_item.transform.Find(s_lt).gameObject);
    setColorN(parent_item.transform.Find(s_rt).gameObject);
    setColorP(parent_item.transform.Find(s_lb).gameObject);
```

```
        setColorP(parent_item.transform.Find(s_rb).gameObject);
        setColorP(parent_item.transform.Find(s_m).gameObject);
        setColorP(parent_item.transform.Find(s_b).gameObject);
        setColorP(parent_item.transform.Find(s_t).gameObject);
        break;
    case 7:
        setColorN(parent_item.transform.Find(s_lt).gameObject);
        setColorP(parent_item.transform.Find(s_rt).gameObject);
        setColorN(parent_item.transform.Find(s_lb).gameObject);
        setColorP(parent_item.transform.Find(s_rb).gameObject);
        setColorN(parent_item.transform.Find(s_m).gameObject);
        setColorN(parent_item.transform.Find(s_b).gameObject);
        setColorP(parent_item.transform.Find(s_t).gameObject);
        break;
    case 8:
        setColorP(parent_item.transform.Find(s_lt).gameObject);
        setColorP(parent_item.transform.Find(s_rt).gameObject);
        setColorP(parent_item.transform.Find(s_lb).gameObject);
        setColorP(parent_item.transform.Find(s_rb).gameObject);
        setColorP(parent_item.transform.Find(s_m).gameObject);
        setColorP(parent_item.transform.Find(s_b).gameObject);
        setColorP(parent_item.transform.Find(s_t).gameObject);
        break;
    case 9:
        setColorP(parent_item.transform.Find(s_lt).gameObject);
        setColorP(parent_item.transform.Find(s_rt).gameObject);
        setColorN(parent_item.transform.Find(s_lb).gameObject);
        setColorP(parent_item.transform.Find(s_rb).gameObject);
        setColorP(parent_item.transform.Find(s_m).gameObject);
        setColorP(parent_item.transform.Find(s_b).gameObject);
        setColorP(parent_item.transform.Find(s_t).gameObject);
        break;
    }
}

private void setColorP(GameObject gameObject)
{
    if(Load_Publics.counter <= Load_Publics.maximum)
    {
        gameObject.GetComponent<Renderer>().material.color = Color.green;
    } else
    {
```

```
        gameObject.GetComponent<Renderer>().material.color = Color.red;
    }

}

private void setColorN(GameObject gameObject)
{
    gameObject.GetComponent<Renderer>().material.color = Color.black;
}

private int[] GetIntArray(int num)
{
    List<int> listOfInts = new List<int>();
    while (num > 0)
    {
        listOfInts.Add(num % 10);
        num = num / 10;
    }
    listOfInts.Reverse();
    return listOfInts.ToArray();
}
```

5.1.5 Display_Meter_5_D.cs

Ähnlich des Scripts *Sev_Seg_counter.cs* ist auch dieses Script zum Anzeigen von Zahlen auf einer 7-Segment-Anzeige gedacht. Die Besonderheit hier ist, dass 5 Ziffern dargestellt werden und die letzten beiden Ziffern nach einem Komma stehen. Also eine Kommazahl.

Der Unterschied der beiden Scripte besteht rein in der Formatierung der Zahl und der Eingabe einer Double-Variable, anstatt einer Integer-Variable. Der Rest ist identisch. Einzig die *set_n()*-Methode wird fünf mal anstatt nur drei mal aufgerufen.

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Display_Meter_5_D : MonoBehaviour {

    // Use this for initialization
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    public void setDisplay(double do_number, string parent_name)
    {
        if(do_number < 1000)
        {
            string s_number = String.Format("{0:000.00}", do_number);
            GameObject parent = GameObject.Find(parent_name);

            Debug.Log(s_number);

            set_n(int.Parse(Char.ToString(s_number[0])), 1, parent);
        }
    }
}
```

```
        set_n(int.Parse(Char.ToString(s_number[1])), 2, parent);
        set_n(int.Parse(Char.ToString(s_number[2])), 3, parent);
        set_n(int.Parse(Char.ToString(s_number[4])), 4, parent);
        set_n(int.Parse(Char.ToString(s_number[5])), 5, parent);

        setColorP(parent.transform.Find("decimal_point").gameObject);
    }
}

private void set_n(int number, int disp_num, GameObject parent_item)
{
    string s_lt = "n_" + disp_num + "_lt";
    string s_rt = "n_" + disp_num + "_rt";
    string s_lb = "n_" + disp_num + "_lb";
    string s_rb = "n_" + disp_num + "_rb";
    string s_m = "n_" + disp_num + "_m";
    string s_b = "n_" + disp_num + "_b";
    string s_t = "n_" + disp_num + "_t";

    switch (number)
    {
        case 0:
            setColorP(parent_item.transform.Find(s_lt).gameObject);
            setColorP(parent_item.transform.Find(s_rt).gameObject);
            setColorP(parent_item.transform.Find(s_lb).gameObject);
            setColorP(parent_item.transform.Find(s_rb).gameObject);
            setColorN(parent_item.transform.Find(s_m).gameObject);
            setColorP(parent_item.transform.Find(s_b).gameObject);
            setColorP(parent_item.transform.Find(s_t).gameObject);
            break;
        case 1:
            setColorN(parent_item.transform.Find(s_lt).gameObject);
            setColorP(parent_item.transform.Find(s_rt).gameObject);
            setColorN(parent_item.transform.Find(s_lb).gameObject);
            setColorP(parent_item.transform.Find(s_rb).gameObject);
            setColorN(parent_item.transform.Find(s_m).gameObject);
            setColorN(parent_item.transform.Find(s_b).gameObject);
            setColorN(parent_item.transform.Find(s_t).gameObject);
            break;
        case 2:
            setColorN(parent_item.transform.Find(s_lt).gameObject);
            setColorP(parent_item.transform.Find(s_rt).gameObject);
```

```
    setColorP(parent_item.transform.Find(s_lb).gameObject);
    setColorN(parent_item.transform.Find(s_rb).gameObject);
    setColorP(parent_item.transform.Find(s_m).gameObject);
    setColorP(parent_item.transform.Find(s_b).gameObject);
    setColorP(parent_item.transform.Find(s_t).gameObject);
    break;
case 3:
    setColorN(parent_item.transform.Find(s_lt).gameObject);
    setColorP(parent_item.transform.Find(s_rt).gameObject);
    setColorN(parent_item.transform.Find(s_lb).gameObject);
    setColorP(parent_item.transform.Find(s_rb).gameObject);
    setColorP(parent_item.transform.Find(s_m).gameObject);
    setColorP(parent_item.transform.Find(s_b).gameObject);
    setColorP(parent_item.transform.Find(s_t).gameObject);
    break;
case 4:
    setColorP(parent_item.transform.Find(s_lt).gameObject);
    setColorP(parent_item.transform.Find(s_rt).gameObject);
    setColorN(parent_item.transform.Find(s_lb).gameObject);
    setColorP(parent_item.transform.Find(s_rb).gameObject);
    setColorP(parent_item.transform.Find(s_m).gameObject);
    setColorN(parent_item.transform.Find(s_b).gameObject);
    setColorN(parent_item.transform.Find(s_t).gameObject);
    break;
case 5:
    setColorP(parent_item.transform.Find(s_lt).gameObject);
    setColorN(parent_item.transform.Find(s_rt).gameObject);
    setColorN(parent_item.transform.Find(s_lb).gameObject);
    setColorP(parent_item.transform.Find(s_rb).gameObject);
    setColorP(parent_item.transform.Find(s_m).gameObject);
    setColorP(parent_item.transform.Find(s_b).gameObject);
    setColorP(parent_item.transform.Find(s_t).gameObject);
    break;
case 6:
    setColorP(parent_item.transform.Find(s_lt).gameObject);
    setColorN(parent_item.transform.Find(s_rt).gameObject);
    setColorP(parent_item.transform.Find(s_lb).gameObject);
    setColorP(parent_item.transform.Find(s_rb).gameObject);
    setColorP(parent_item.transform.Find(s_m).gameObject);
    setColorP(parent_item.transform.Find(s_b).gameObject);
    setColorP(parent_item.transform.Find(s_t).gameObject);
    break;
```

```
case 7:
    setColorN(parent_item.transform.Find(s_lt).gameObject);
    setColorP(parent_item.transform.Find(s_rt).gameObject);
    setColorN(parent_item.transform.Find(s_lb).gameObject);
    setColorP(parent_item.transform.Find(s_rb).gameObject);
    setColorN(parent_item.transform.Find(s_m).gameObject);
    setColorN(parent_item.transform.Find(s_b).gameObject);
    setColorP(parent_item.transform.Find(s_t).gameObject);
    break;
case 8:
    setColorP(parent_item.transform.Find(s_lt).gameObject);
    setColorP(parent_item.transform.Find(s_rt).gameObject);
    setColorP(parent_item.transform.Find(s_lb).gameObject);
    setColorP(parent_item.transform.Find(s_rb).gameObject);
    setColorP(parent_item.transform.Find(s_m).gameObject);
    setColorP(parent_item.transform.Find(s_b).gameObject);
    setColorP(parent_item.transform.Find(s_t).gameObject);
    break;
case 9:
    setColorP(parent_item.transform.Find(s_lt).gameObject);
    setColorP(parent_item.transform.Find(s_rt).gameObject);
    setColorN(parent_item.transform.Find(s_lb).gameObject);
    setColorP(parent_item.transform.Find(s_rb).gameObject);
    setColorP(parent_item.transform.Find(s_m).gameObject);
    setColorP(parent_item.transform.Find(s_b).gameObject);
    setColorP(parent_item.transform.Find(s_t).gameObject);
    break;
}
}

private void setColorP(GameObject gameObject)
{
    if (Load_Publics.counter <= Load_Publics.maximum)
    {
        gameObject.GetComponent<Renderer>().material.color = Color.green;
    }
    else
    {
        gameObject.GetComponent<Renderer>().material.color = Color.red;
    }
}
```

```
private void setColorN(GameObject gameObject)
{
    gameObject.GetComponent<Renderer>().material.color = Color.black;
}

private int[] GetIntArray(int num)
{
    List<int> listOfInts = new List<int>();
    while (num > 0)
    {
        listOfInts.Add(num % 10);
        num = num / 10;
    }
    listOfInts.Reverse();
    return listOfInts.ToArray();
}
```

5.2 Szenenspezifische Scripte

5.2.1 Beschleunigung.cs

Dieses Script ist für die Steuerung der Geschwindigkeit der Teilchen im Teilchenlabor zuständig.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Beschleunigung : MonoBehaviour {

    // Use this for initialization
    void Start () {
        Sev_Seg_Counter counti = new Sev_Seg_Counter();
        counti.setSevSegCount(0, "Zähler1");
    }

    // Update is called once per frame
    void Update () {
        Sev_Seg_Counter counti = new Sev_Seg_Counter();
        counti.setSevSegCount(Load_Publics.Temperatur, "Zähler1");
    }

    private void OnTriggerEnter(Collider other)
    {
        switch (gameObject.name)
        {
            case "plus":
                if (Load_Publics.plus_act && Load_Publics.Temperatur < Load_
                {
                    Load_Publics.Temperatur += 25;
                    Debug.Log("wärmer");
                    StartCoroutine(waiter(true));
                }
                break;
            case "minus":
                if (Load_Publics.min_act && Load_Publics.Temperatur > Load_
                {
                    Load_Publics.Temperatur -= 25;
                }
        }
    }
}
```

```
        Debug.Log("kälter");
        StartCoroutine(waiter(false));
    }
    break;
}

IEnumerator waiter(bool isplus)
{
    GameObject button_mol = null;
    if (isplus)
    {
        button_mol = GameObject.Find("plus");
        Load_Publics.plus_act = false;
    } else
    {
        button_mol = GameObject.Find("minus");
        Load_Publics.min_act = false;
    }
    button_mol.GetComponent<Renderer>().material.color = Color.yellow;

    yield return new WaitForSeconds(1);      //Wait 1 Second
    Color color = new Color();

    if (isplus)
    {
        Load_Publics.plus_act = true;
        if(ColorUtility.TryParseHtmlString("#FF0000", out color))
        {
            button_mol.GetComponent<Renderer>().material.color = color;
        }
    }
    else
    {
        Load_Publics.min_act = true;
        if (ColorUtility.TryParseHtmlString("#00FFFF", out color))
        {
            button_mol.GetComponent<Renderer>().material.color = color;
        }
    }
}
```

Das Script wurde auf die beiden Buttons in der Szene zum Beschleunigen und Abbremsen der Teilchen gelegt. Bei Aufruf der Szene wird zuerst in der *start()* die 7-Segment-Anzeige auf 0 gesetzt. Das ist der Temperatur Startwert. Anschließend wird die Anzeige in der *Update()* jeweils auf den Wert der Temperatur-Variable in der *Load_Publics.cs* gesetzt. Standardmäßig ist dies 25.

Das Script enthält außerdem Listener, die ausgelöst werden, sobald der Spieler einen der Buttons berührt. Diese Listener unterscheiden dann welcher Button gedrückt wurde und ändern entsprechend die Temperatur-Variable. Nachdem ein Button gedrückt wurde wird es für eine Sekunde unmöglich diesen nochmals zu drücken. Dies wird auch durch eine Farbänderung dargestellt.

Kapitel 6

Verworfene Ideen

Einige Versuche und Ideen, die zu Beginn gemacht wurden, wurden letztendlich verworfen, da stattdessen andere Ideen und Inputs kamen, denen stattdessen nachgegangen wurden. Dies ist aus unserer Sicht bei einem solch flexiblen Projekt unvermeidlich.

Dazu zählt ein Geografie Labor, für das bis zum Schluss keine sinnvollen Ideen gesammelt werden konnten. Zwar war die Überlegung irgendwie eine Karte mit einfließen zu lassen, jedoch konnte keine fertige Durchführung in einem Versuch konstruiert werden.

Des Weiteren wurden viele Versuchsskizzen verworfen, da andere Ideen vorgezogen wurden. So zum Beispiel wäre eine Turing Maschine im Informatiklabor vorstellbar, das Sezieren eines Frosches oder Fisches im Biologielabor, so wie viele Ideen, für das Labor der Mathematik, wie Kugelkoordinaten. Natürlich kennen die Wissenschaft und somit auch das Virtual Science Lab keinen Endzustand. Es ist stets erweiterbar und durch weitere Versuche, Laborräume und Erweiterungen fortführbar. Neben den genannten Versuchen, die es in die konkrete Ideenfindung geschafft haben, zeigt der Ausblick wie es mit dem Projekt weitergehen könnte.

Kapitel 7

Ausblick

Ob und wie es mit dem Projekt Virtual Science Lab weitergehen wird, steht zum Zeitpunkt dieser Abgabe noch in den Sternen, jedoch ist es von unserer Seite mehr als wünschenswert, dass es eine Zukunft für das Projekt geben wird. Die aktuellen Versuche sind immer verbesserbar. Allein der Unterschied zwischen dem Bunsenbrenner im Ausgangsprojekt und der neu modellierte zeigen, dass kleine Details einen immensen Unterschied für das Gesamterlebnis beitragen können. Zwar wurde schon sehr auf Details wert gelegt, doch perfekt ist eben doch zuletzt im Ermessen des Betrachters.

Außerdem ist es sinnvoll, den Android Build erneut anzugehen. Bei diesem besteht das große Problem, verschiedene Geräte mit unterschiedlichsten Steuerungsarten mit einzubeziehen.

Außerdem wäre eine Art Feldtest sinnvoll, in dem getestet wird, ob Schüler oder Studenten, die das Virtual Science Lab absolvieren, in einer passenden Prüfung besser abschneiden könnten, als eine Gruppe, die das Labor nicht gesehen hat.

Der Aufbau der Laborräume glich immer mehr einem Prozess, bei dem dieselben Schritte nacheinander abgespult wurden. Zunächst wird der Raum durch Wände, Boden und Decke begrenzt, eine Tür wird hinzugefügt, vor der die Kamera platziert wird, sodass man an der Position der Tür den Raum betritt. Anschließend werden falls nötig Tische angeordnet, so wie die verschiedenen Utensilien die für die vorgesehenen Versuche nötig sind. Zum Schluss werden die Skripte zur Funktionalität geschrieben, um so die Ver-

suchsdurchführung zu ermöglichen. Anschließend wird der Versuch im Virtual Reality Labor getestet und falls nötig Bugs notiert, die im Anschluss gefixt werden. Dieser Evaluationsvorgang wird iterativ beliebig oft durchgeführt, bis das Ergebnis zufriedenstellend ist.

Kapitel 8

Fazit

Wie schon aus verschiedenen anderen Projekten hat die Zusammenarbeit problemlos funktioniert und Absprachen sowie regelmäßige Treffen im Virtual Reality Labor wurden über den gesamten Projektverlauf hinweg gepflegt. Alle Probleme waren nach und nach gut lösbar und es wurde innerhalb des Projektverlaufes einiges in gegenseitigem Einverständnis geändert, was jedoch im Nachhinein immer eine gute Entscheidung war. Der einzige Wermutstropfen ist der zu kurz gekommene Android Build, für den am Ende die Zeit ausging. Zwar ist eine Ausführung über Android Geräte wie die Samsung VR Gear prinzipiell möglich, allerdings ist die Steuerung der HTC Vive Pro nicht kompatibel mit solchen Low Budget Modellen. Diese Aufgabenstellung, sowie eine Evaluierung im Schul- oder Hochschulumfeld sind Dinge, die für die Zukunft des Projektes denkbar sind und einen nächsten Schritt bilden sollten.

Wir sind sehr froh, dass wir nun über einen Zeitraum von fast einem Jahr (mit dem vorherigen Semester zusammen) an dem Projekt arbeiten durften, was uns einen tiefen Einblick in die Entwicklung von Virtual Reality Anwendungen, vor allem in der Entwicklungsumgebung Unity vermitteln konnte. Auch die Erfahrungen im Arbeiten mit Git konnte deutlich verbessert werden, was mittlerweile kein Problem mehr für einen guten Arbeitsfluss aufgeworfen hat.

Das alleinige Ausarbeiten von Ideen, unterstützt von Herr Prof. Dr. Manfred Brill konnte uns jedoch vor allem tiefe Einblicke bieten, wie es ist, an einem dynamischen Projekt von der Planung bis zur Entwicklung teilzuhaben, was uns für unsere spätere Karriere noch von großem Nutzen sein kann. Dabei

waren auf die aufgetretenen Fehler und Probleme eine große Hilfe, denn die Realität der Projektdurchführung ist (wohl fast) nie wie die Vorstellung und Planung.

Abschließend gilt unser Dank Herrn Prof. Dr. Brill, sowie Fabian Kalweit, die auch kurzfristig stets Zeit gefunden haben, um mit uns den weiteren Ablauf zu besprechen und auch etliche Ideen mit auf den Weg gegeben haben. Außerdem sind wir sehr dankbar für die Modelle des Mikroskops und des Bunsenbrenners unseres Kommilitonen Cedric Schug, der falls der Unity Asset Store nicht ergiebig war unter die Arme gegriffen hat. Zu guter Letzt auch noch danke an Herr Dr.-Ing. Hubert Zitt, der sich ebenfalls viel Zeit nahm, um uns die beiden Elektrotechnik-Versuche genau aufzubauen, zu zeigen und zu erklären.

Literaturverzeichnis

- [Zin] ZINTEL, MARC, LAUER, PHILIPP: *Virtual Science Lab.* <https://github.com/fraygeyst/VirtualScienceLab>. Zuletzt gesehen am 31.12.2019.
- [Zita] ZITT, HUBERT: *Laboraufgaben 03 der Elektrotechnikvorlesung an der Hochschule Kaiserslautern.* http://webhome.hs-kl.de/~zitt/downloads_save/ET/ET_MNT2_Zitt_Labor03.pdf. Zuletzt gesehen am 18.12.2019.
- [Zitb] ZITT, HUBERT: *Laboraufgaben 05 der Elektrotechnikvorlesung an der Hochschule Kaiserslautern.* http://webhome.hs-kl.de/~zitt/downloads_save/ET/ET_MNT2_Zitt_Labor05.pdf. Zuletzt gesehen am 18.12.2019.