



Janek Zitzmann & Pascal Kattler

Ehrenwörtliche Erklärung	1
1. Motivation	2
2. Projektübersicht	2
2.1 Idee	2
2.2. Die Anwendung	2
2.3. Motion Seat	3
3. Bedienungsanleitung	4
4. Projektstruktur	5
4.1. Plane Controller	5
4.2. Flight Physics	6
4.3. Controls	7
4.3.1. Joystick	7
4.3.2. Throttle	8
4.5. SerialPort	9
4.5.1. SerialConnection	10
4.6. MotorController	10
4.7. TerrainGeneration	11
4.8. Audio	14
4.8.1. Sound	14
4.8.2. Audiomanager	15
4.9. Intro	16
5. Arduino	17
5.1. Code	17
5.2. Protokoll	19
6. Quellen und Credits	20
6.1. Tutorials	20
6.2. Assets	20

## Ehrenwörtliche Erklärung

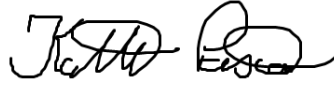
---

Hiermit erklären wir, Janek Zitzmann und Pascal Kattler, ehrenwörtlich,

- dass die abgegebene Projektarbeit selbstständig und ohne fremde Hilfe von uns angefertigt wurde und keine anderen als in der Abhandlung angegebenen Hilfen benutzt wurden
- dass die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Abhandlung gekennzeichnet sind.



29.07.2022, Zitzmann



29.07.2022, Kattler

# 1. Motivation

---

Bei einem Flugsimulator ermöglicht die dreidimensionale, virtuelle Sicht ein natürliches Erlebnis. Die Sicht aus dem Cockpit eines Flugzeugs realistisch nachzuahmen war unsere Motivation.

Motion Sickness ist bei Bewegungsarten ein ausgeprägtes Problem. Auch bei einem Flugsimulator entsteht durch das Ausbleiben eines taktilen Feedbacks bei einigen Flugmanövern Motion Sickness. Teil unseres Projektes ist deshalb die Einbindung eines Motion Seats für den Piloten, der ein taktilen Feedback ermöglicht um die Gefahr von Motion Sickness zu reduzieren.

Die Hochschulumgebung ist perfekt geeignet für solch ein Projekt. Da der Bau eines Motion Seats für Studenten ohne Platz und finanzielle Mittel sehr teuer werden würde, wäre diese Idee nicht umsetzbar. Durch die Kombination aus Vorlesungsinhalten und Projektarbeiten ist die Umsetzung möglich. Auch die Anbindung an das VR-Labor und an die Elektrotechnik-Werkstatt sind ein wichtiger logistischer Aspekt.

Jeder der sich für VR begeistert und ein ähnliches Projekt durchführen möchte, hat an der Hochschule die besten Möglichkeiten.

## 2. Projektübersicht

---

### 2.1 Idee

Flugsimulatoren sind in der Spieleszene verbreitet und werden stetig weiterentwickelt. Ein sich bewegender Pilotensitz erhöht das Realitätsgefühl und bringt den Anwender tiefer in das Erlebnis ein.

Daraus entstand die Idee nach dem Bau eines solchen Simulators. Die Soft- sowie Hardware, darunter auch der Pilotensitz, sollten selbst entwickelt und getestet werden um zu verstehen, wie sich digitale Komponenten auf ein physikalisches System auswirken.

Als Projektstart wurde der Fokus auf den elektrotechnisch und softwarebasierten Projektteilen und der Einbindung von virtueller Realität gelegt.

Als nächsten Schritt würde der Stuhl gebaut und angepasst werden.

### 2.2. Die Anwendung

Die Anwendung besteht aus drei Teilen. Die VR-Einbindung, der eigentliche Programmcode und die Ansteuerung eines Arduinos.

Für die virtuelle Darstellung wurde die HTC Vive Pro genutzt.

Der eigentliche Programmcode umfasst die grafische Oberflächengestaltung, die Flugphysik und die Schnittstellenprogrammierung zum C-basierten Arduino.

Für die Flugphysik wurde eine physikalisch hinreichend genaue Berechnung der einwirkenden Kräfte vorgenommen.

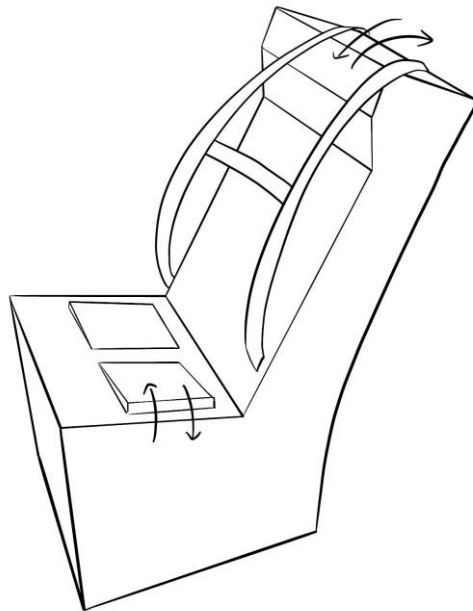
Die Generierung des Untergrunds erfolgt durch einen separaten Map Generator.

Die dritte Komponente der Anwendung ist der in C geschriebene Quellcode des Arduinos und die dazugehörige Hardware. Hierfür wurde ein eigenes Protokoll zur Übertragung von Befehlen und Werten zwischen Unity und dem Arduino entwickelt. Um das Ansteuern der Motoren zu simulieren, wurden pro Motor zwei LEDs verwendet.

### 2.3. Motion Seat

Der Motionseat soll während des Fliegens die auf den Piloten einwirkenden Kräfte simulieren.

Der Stuhl besteht aus zwei Komponenten: Einem Sitzgurt, der den Nutzer bei Beschleunigung in den Sitz drückt und zwei Klappen im Boden des Stuhls. Diese passen sich der Rotation des Flugzeuges an und drücken den Nutzer nach rechts, links und oben im Falle eines Überkopfflugs (siehe Abb. 1).



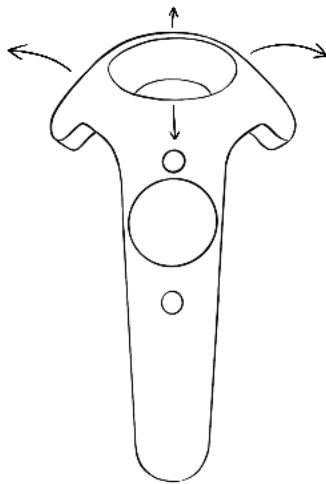
### 3. Bedienungsanleitung

---

Für das Steuern des Flugzeuges werden die Trigger ( 7 ) der Controller benötigt.

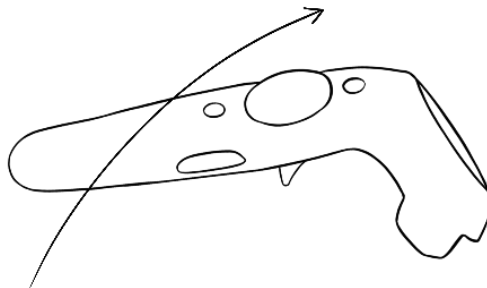
#### Rechter Controller (Flightstick):

- Halten Sie den Controller vertikal
- Betätigen Sie den Trigger, lassen ihn gedrückt und verändern dabei den Winkel des Controllers
- Im Flug verändert das Flugzeug nun die Richtung



#### Linker Controller (Throttle):

- Halten Sie den Controller horizontal
- Betätigen Sie den Trigger, lassen ihn gedrückt und bewegen ihn nach vorne. Sie geben nun Schub.
- Betätigen Sie den Trigger, lassen ihn gedrückt und bewegen ihn nach hinten. Sie nehmen nun Schub weg.



## 4. Projektstruktur

---

- Packages und lose Komponenten
- Packages können in andere Projekte übernommen werden

Packages:

- TerrainGeneration
- Controls
- SerialPort
- Audio

Lose Komponenten:

- Plane Controller
- Flight Physics
- G Kraft
- Timeline

### 4.1. Plane Controller

Der Plane Controller ist die Steuerungsklasse unseres Projektes. Bis auf die Generierung des Terrains werden alle Komponenten in dieser Klasse zusammengeführt. Während der Entwicklung lag bei der Planung ein großer Fokus darauf, die einzelnen Packages möglichst voneinander zu trennen. Dementsprechend importiert der Plane Controller aus jedem Package mindestens eine Klasse. Die Klasse ist sehr speziell auf unser Projekt zugeschnitten und besitzt auch keine öffentlichen Schnittstellen. Damit ist sie für die Weiterverwendung vermutlich eher ungeeignet. Deswegen haben wir uns entschieden die Beschreibung kurz zu fassen und lediglich eine kurze Übersicht über die `FixedUpdate()` Methode zu geben.

Jedes Update werden folgende Funktionen ausgeführt:

```
updatePlane();
```

Liest die Werte von Joystick und Throttle ein und gibt diese an die Flightphysics weiter.

```
updateHotasModel();
```

Rotiert Flightstick und Throttle des Cockpitmodels im Bezug auf die Werte von Throttle (Klasse) und Joystick.

```
updateSound();
```

Passt den Sound der Spielsituation an.

```
updateMotionSeat();
```

Berechnet aus den Daten des RigidBodyes des Flugzeugs die auf den Piloten wirkenden Kräfte. Auf der Basis dieser Daten werden die "Motoren" an- bzw. ausgeschaltet.

## 4.2. Flight Physics

### Implementierte Kräfte

#### Drag

Drag beschreibt die Auswirkungen der Geschwindigkeit und der AirBrakes auf das Flugzeug. Je schneller das Flugzeug fliegt, desto schwerer ist es die Ausrichtung zu ändern. Die Lenkung versteift sich. Dadurch wird die Krafteinwirkung der Geschwindigkeit dargestellt.

#### Aerodynamik

Die Aerodynamik berechnet sich aus der aktuellen Position und der aktuellen Geschwindigkeit mit der angesteuerten Veränderung dieser Parameter. Durch lineares Interpolieren wird die neue Richtung des Flugzeugs festgelegt. Das Flugzeug fliegt eine realistische Kurve in Richtung der neuen Orientierung.

Das Flugzeug hat sich auch dieser Orientierung angepasst.

#### Beschleunigung und Auftriebskraft

Die Beschleunigung berechnet sich aus der aktuellen Geschwindigkeit und der veränderten Kraft des Triebwerks.

Der Auftrieb berechnet sich aus der Neigung des Flugzeugs in Kombination mit der `Rigidbody.velocity`.

Diese Kräfte werden als ein `Vector3` dem `Rigidbody` in der Methode `AddForce(Vector3 a)` übergeben.

#### Drehmoment

Das Drehmoment (Eng.: Torque) wird für die Flugeigenschaften benötigt. Es wird das Drehmoment einberechnet, welches aus den Richtungswinkeln des Flugzeugs besteht. Fliegt man beispielsweise eine Kurve, so wird das Drehmoment für `Roll` neu berechnet und auf den `Rigidbody` angewandt.

Das Drehmoment wird mit der `AddTorque(Vector3 torque)` Methode angewandt.

### Öffentliche Schnittstellen

```
public void Move(float rollInput, float pitchInput, float yawInput, float throttleInput, bool airBrakes);
```

Die `Move`-Funktion wandelt die Input-Parameter in die einzelnen Variablen um, wie sie für obige Berechnungen benötigt werden.

Diese Funktion berechnet alle Kräfte und wendet sie auf den `Rigidbody` an.



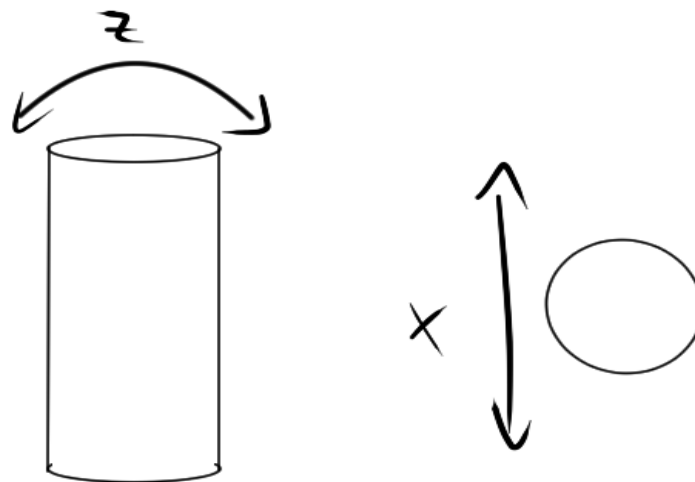
### 4.3. Controls

Das Package Controls besteht aus zwei Skripten, die unabhängig voneinander eingebunden werden können: "JoyStick.cs" und "Throttle.cs". Die Idee ist es, mit den Vive Controllern die Bedienung eines handelsüblichen Hotas-Systems("Hands on Throttle and Stick" ) nachzubilden.

#### 4.3.1. Joystick

Um das Tracking zu starten muss der Nutzer eine definierte Taste drücken. Beim Drücken dieser Taste wird die Rotation des Controllers zum Zeitpunkt des Drückens als Null-Rotation gespeichert. Solange die Taste gedrückt bleibt, liefert ein Abfragen der Eingabewerte die Differenz zwischen der derzeitigen Rotation und der Null-Rotation zurück.

Der Eingabewert eines Joysticks besteht aus zwei Komponenten: Die Rotation um die X-Achse und die Rotation um die Z-Achse. Wird der Stick nach vorne bzw. Hinten geneigt, rotiert er um die X-Achse. Wird er nach rechts oder links geneigt, rotiert er um die Z-Achse.



Dieser zweidimensionale Vektor wird in der Joystick Klasse wie folgt berechnet:

1. Beim Drücken des Triggers(bzw. Des definierten Buttons) wird die Null-Rotation gespeichert.
2. Jedes Update wird die Differenz zwischen der Null-Rotation und der jetzigen Rotation berechnet.
3. Da Quaternion.euler Werte zwischen 0 und 360 zurückliefert, werden von allen Werten > 180 360 abgezogen. Der Differenzvektor Delta hat damit einen Wertebereich von -180 bis 180.
4. Der Eingabewert wird jetzt in eine Prozentzahl umgerechnet. Diese Prozentzahl bezieht sich auf eine im Editor festgelegte Reichweite.

## Editorfelder



### Role

Diese Hand soll den Joystick bedienen.

### Tracking Activation Button

Mit dieser Taste wird das Tracking gestartet.

### Range

Gibt an, wie weit der Controller rotiert werden muss um 100% zu liefern. Bei einer Reichweite von 90 liefert delta.x eine 1 für wenn der Controller 90 Grad um die X-Achse rotiert wurde und analog eine -1 für eine Rotation um -90 Grad.

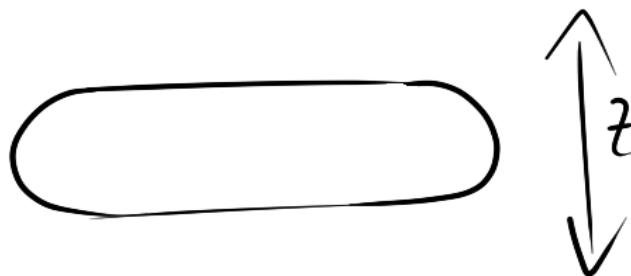
## Öffentliche Schnittstellen

### getRelativePosition()

Gibt einen zweidimensionalen Vektor mit Werten im Bereich [-1,1] zurück.

## 4.3.2. Throttle

Ein Throttle (z.d. Schubregler) ist ein Schieberegler, der durch eine Verschiebung in Richtung der Z-Achse ("nach vorne/hinten") die Geschwindigkeit des Flugzeuges(oder eines generellen Fahrzeuges) regelt.

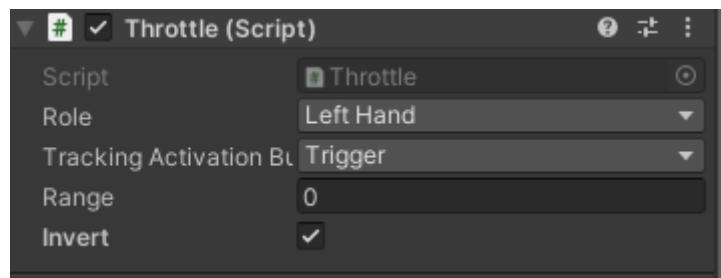


Ein Throttle hat ein Minimum (0%) und ein Maximum(100%). Anders als der Flightstick richtet sich der Throttle nach dem Loslassen nicht wieder auf seine Default-Position aus sondern behält den Wert den der Pilot eingestellt hat.

Die Throttleklasse funktioniert sehr ähnlich zum Flightstick. Wird der Trigger gedrückt, wird die Ausgangsposition abgespeichert und die Veränderung in Z-Richtung berechnet. Die Veränderung wird

in den Bereich  $[-1,1]$  gekürzt. Anstatt diesen Wert direkt zurückzugeben, wird er auf eine Klassenvariable Throttlevalue addiert. Dieser ThrottleValue wird bei Abfrage zurückgegeben.

## Editorfelder



### Role

Welche Hand soll den Joystick bedienen.

### Tracking Activation Button

Mit welcher Taste wird das Tracking gestartet.

### Range

Gibt an wie weit der Controller bewegt werden muss um 100% Schub zu erreichen. Dieser Wert ist abhängig von der Skalierung der Spielwelt und muss daher durch Testen eingestellt werden.

### Invert

Gibt an ob die Steuerung invertiert werden soll. Bei Invertierung muss der Nutzer den Controller von sich weg bewegen um Schub zu geben. Ohne Invertierung muss der Nutzer den Controller zu sich ziehen um Schub zu geben.

## Öffentliche Schnittstellen

### getThrottleValue()

Gibt die Position des Throttles in Prozent zurück. Dieser Wert liegt im Bereich  $[0,1]$ .

## 4.5. SerialPort

Das Package SerialConnection besteht aus zwei Klassen. Die Klasse "SerialConnection.cs" regelt die serielle Kommunikation über den Serialport. Die Klasse "MotorController.cs" nutzt die Funktionen der SerialConnection um das Verhalten der Motoren des MotionSeats zu simulieren.

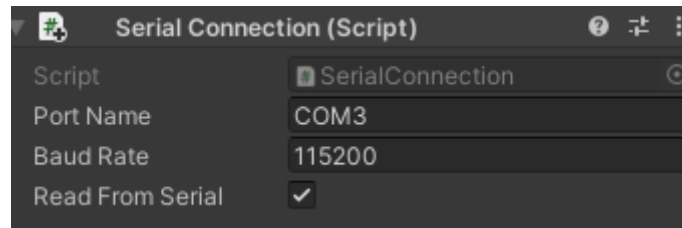
**Wichtig:** Um die C# Klassen SerialPort nutzen zu können, muss das Kompatibilitätslevel in den Player Settings auf 4.x gesetzt werden.



#### 4.5.1. SerialConnection

Diese Klasse nutzt die C#-Klasse SerialPort um eine Verbindung zu einem definierten Port aufzubauen.

##### Editorfelder



##### Port Name

Der Name des Ports zu dem die Verbindung hergestellt werden soll. Nach dem Anschluss des Arduinos kann im Gerätemanager der Name des entsprechenden Ports ausgelesen werden.

##### Baudrate

Baudrate definiert die Übertragungsgeschwindigkeit in Bit pro Sekunde. **Die Übertragungsrate muss in Unity und auf dem Mikrocontroller übereinstimmen!**

##### ReadFromSerial

Die Serialconnectionklasse startet einen Thread der bei gesetzter Flag auf eingehende Daten aus dem Serialport lauscht. Diese werden anschließend auf die Konsole geloggt.

#### Öffentliche Schnittstellen

```
static void setOutputOnPort(int port, State value)
```

Schickt einen Command String der Form ">port:value.toInt()" an den Arduino.

'>' ist das Operations Symbol für das Setzen eines digitalen Output Ports.

```
static void resetAll()
```

Setzt alle digitalen Output Pins auf 0.

#### 4.6. MotorController

Der Motionseat soll später drei Motoren haben. Der MotorController bietet Methoden um diese gezielt zu setzen bzw. Zurückzusetzen. Die Motoren sind von 1-3 durchnummeriert. Jeweils zwei Pins (Set und Reset) sind einem Motor zugeordnet:

- Motor 1
  - Pin 12 SET
  - Pin 11 RESET
- Motor 2
  - Pin 10 SET

- Pin 9 RESET
- Motor 3
  - Pin 8 SET
  - Pin 7 RESET

## Öffentliche Schnittstellen

`setMotorX()`

Setzt den SET Pin von Motor x auf 1 und den RESET Pin auf 0.

`resetMotorX()`

Setzt den RESET Pin von Motor x auf 1 und den SET Pin auf 0.

`disableMotorX()`

Setzt beide Pins von Motor x auf 0.

`resetAll()`

Setzt alle Pins auf 0.

## 4.7. TerrainGeneration

Dieses Package ist für die Generierung von TerrainChunks zur Runtime zuständig. Eine ausführliche Beschreibung der Funktionsweise würde den Rahmen an dieser Stelle sprengen. Deswegen wird nur ein kurzer Überblick über die Generierung und eine Anleitung zur Benutzung im Editor gegeben.

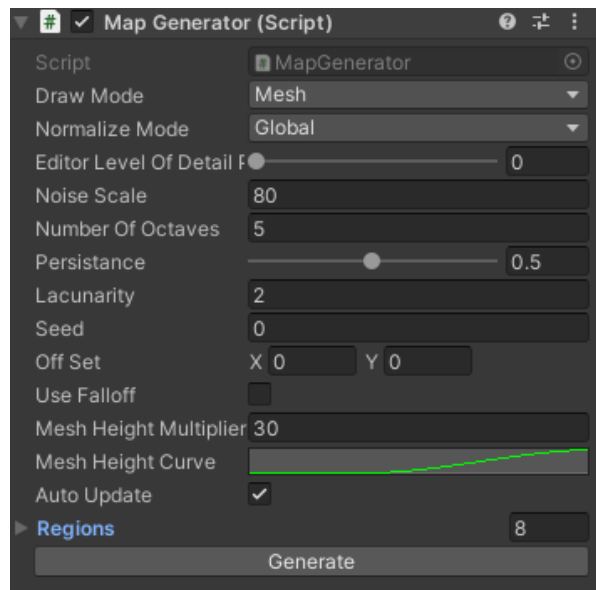
Die Generierung eines einzelnen Chunks läuft in fünf Schritten ab:

1. Erzeugen einer HeightMap mithilfe von PerlinNoise
2. Erzeugen einer ColorMap auf Basis der Heightmap
3. Erzeugen eines Meshes auf Basis der Heightmap
4. Umwandeln der ColorMap in eine 2D-Textur die an das Mesh angefügt wird
5. Erzeugen eines Colliders und Anfügen an das Mesh

Die tatsächliche Generierung ist komplizierter, die generelle Vorgehensweise ändert sich jedoch nicht.

## Editorfelder MapGenerator

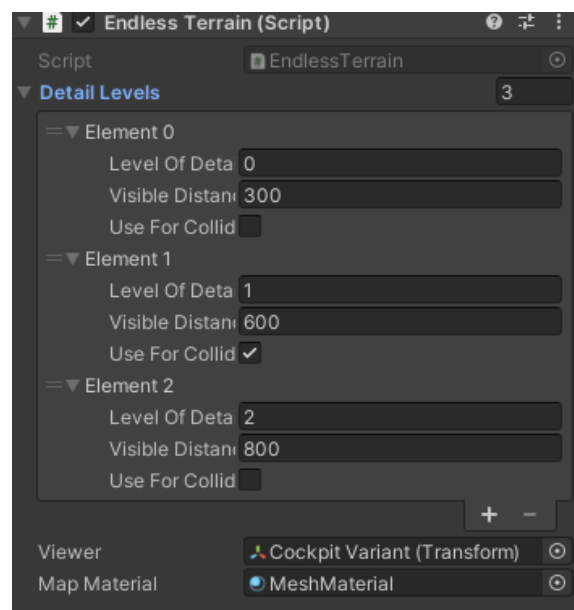
Um die Auswirkungen der einzelnen Felder kennen zu lernen, kann die Szene TerrainGenerationTest aufgerufen werden.



- Drawmode, Editor LoD und AutoUpdate
  - Werden für Preview im Editor genutzt und nicht für die eigentliche Anwendung
- Normalize Mode
  - Wenn ein endloses Terrain generiert wird, werden mehrere Meshes aneinander gereiht. Um Umbrüche zwischen den Maps anzugleichen, muss ein globaler Höchstwert approximiert werden.
  - Global für Endless Terrain
  - Lokal für eine einzelne Map
- Noise Scale
  - Skaliert die Heightmap nach oben.
- Number of Octaves
  - Octaves sind mehrere Noisemaps die aufeinander addiert werden.
  - Hierbei haben nachfolgende Oktaven höhere Frequenzen, jedoch eine geringere Auswirkung auf das Terrain.
  - Sorgt für mehr Detail im Terrain
- Persistance
  - Einfluss der Oktaven nimmt um diesen Faktor pro Level ab
  - Bei 0.5 halbiert sich der Einfluss den eine Oktave hat pro Level
- Lacunarity
  - Die Frequenz des Noises erhöht sich pro Oktave um diesen Wert
  - Sorgt für viele kleine Unregelmäßigkeiten im Terrain
- Seed

- Derselbe Seed führt zu derselben Noisemap
- Offset
  - Führt zu einer Verschiebung der Noisemap in die jeweilige Richtung
- MeshHeightMultiplier
  - Skaliert die Höhe des Meshes
- MeshheightCurve
  - Sorgt für das Setzen der Werte auf Null unter einem bestimmten Schwellenwert
  - Verhindert Hügel im Wasser und sorgt für eine glatte Wasseroberfläche
- Regions
  - Regionen kann eine Farbe und ein Schwellenwert zugewiesen werden.
  - Auf Basis dieser Regionen wird die ColorMap zugewiesen.

### Editorfelder EndlessTerrain



Im Inspektor des EndlessTerrain Skripts können lediglich die Level of Details der verwendeten Meshes eingestellt werden.

### **LoD**

- Level of Detail
  - Die Anzahl der Vertices im LoD-Mesh wird um  $1/(LoD + 1)$  reduziert.
- Visible Distance
  - Der Abstand zum Viewer ab dem das LoD-Mesh verwendet werden soll.
- Use For Collider
  - Welches Detail Level für die Generierung des Colliders verwendet werden soll.

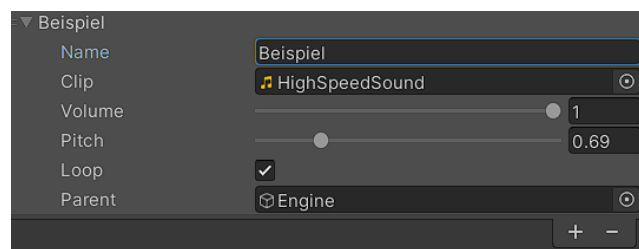
## 4.8. Audio

Die Audioausgabe wird mit Hilfe eines Audiomanagers abgearbeitet. Bei Schlüsselereignissen, wie dem Beschleunigen oder Abbremsen, wird der abgespielte Sound verändert. Diese werden von der Sound-Klasse zur Verfügung gestellt.

### 4.8.1. Sound

Die Soundklasse ist eine reine Datenklasse. Sie enthält einen AudioClip sowie mehrere Attribute für das Abspielen des Sounds. Der Sound ist dreidimensional, dementsprechend kann jedem Sound ein Parent GameObject zugeordnet werden. Dieses sorgt dafür, dass der Sound an der richtigen Stelle abgespielt wird. Zur Runtime wird durch den Audiomanager für jeden Sound eine eigene AudioSource erstellt und an das jeweilige Parentobject angehängt.

Editorfeld



#### Name

Der Name dient als ID. Soll der zugehörige Clip durch den Audiomanager abgespielt werden muss dieser Name als Parameter mitgegeben werden.

#### Clip

Die Audioquellen können per Drag'n'Drop abgelegt werden. Unity unterstützt verschiedene Dateitypen (Siehe auch: <https://docs.unity3d.com/Manual/AudioFiles.html>). Die im Projekt genutzten Files sind mp3-Files und wav-Files.

#### Volume

Die Lautstärke kann von 0 bis 1 angepasst werden. Hierbei handelt es sich um die Lautstärke der erstellten AudioSource und nicht des Clips an sich.

#### Pitch

Der Pitch gibt die Tonhöhe an. Je größer der eingestellte Wert, desto höher ist die Frequenz des der erstellten Audioquelle.

#### Loop

Ist diese Flag gesetzt, wird der Clip in einer Endlosschleife wiederholt.

#### Parent

Container für die zur Runtime erstellten Audioquelle.



### 4.8.2. Audiomanager

Der Audiomanager verwaltet die angegebenen Sounds. Für jeden Sound wird beim Szenenstart eine AudioSource erstellt und an das hinterlegte Parent Objekt angehängt. Gleichzeitig verwaltet er über eine Map die Zuordnung zwischen Soundname und AudioSource.

#### Öffentliche Schnittstellen

```
public void Play(string name)
```

Startet das Abspielen durch die dem Namen zugordnete AudioSource.

```
public bool isPlaying(string name)
```

Returns audiosource.isPlaying der zugehörigen AudioSource.

```
public void Stop(string name)
```

Evokes audiosource.stop() der zugehörigen AudioSource.

```
public void setVolume(string name, float volume)
```

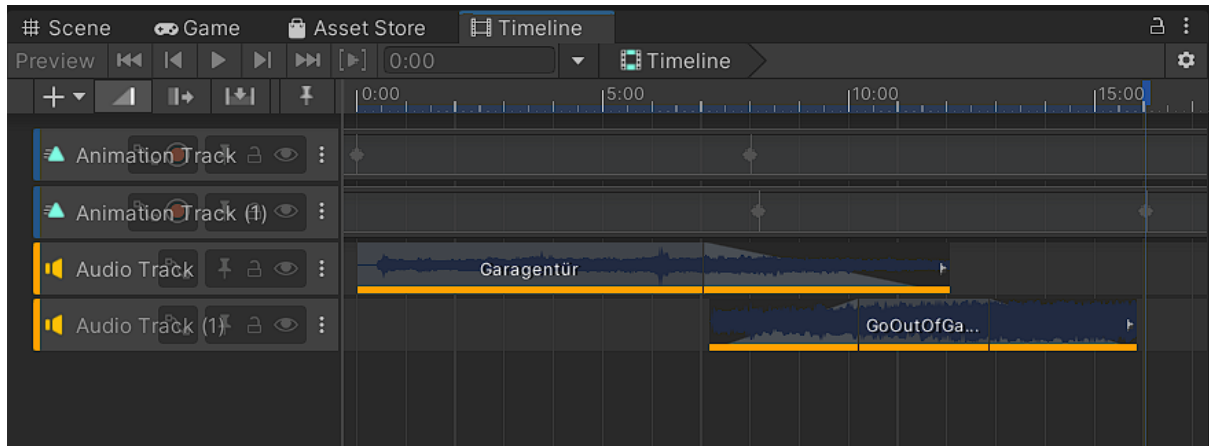
Evokes audiosource.setVolume(volume) der zugehörigen AudioSource.

## 4.9. Intro

Das Intro wurde mit Hilfe der Timeline Funktion erstellt. Man erzeugt im Projekt ein leeres GameObject und kann die Timeline erstellen.

Um die Timeline erstellen zu können, muss diese auf die Oberfläche in Unity angezeigt werden. Hierfür klickt man unter Window -> Sequencing auf Timeline

Übersicht



Die Timeline besteht aus einzelnen Spuren (vgl.: Animation Track, AudioTrack). Diese können beliebig editiert werden.

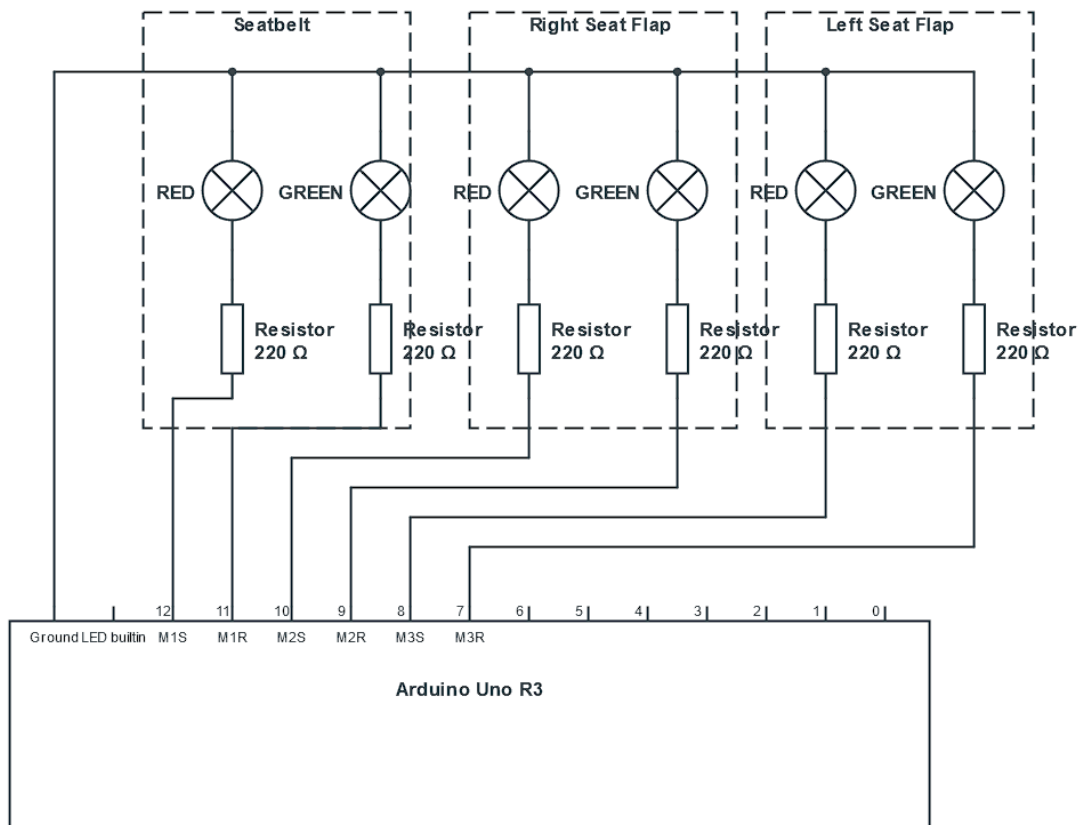
Um Animationen erstellen zu können muss man einen zeitlichen Start- und Endpunkt wählen und die Animation manuell erzeugen. Zum Beispiel wurde im Projekt das Garagentor geöffnet, indem ein Start- und Endpunkt gewählt wurde und Start- sowie Endrotation im Inspector eingestellt wurde. Unity erhält dadurch zwei Rotationswerte und passt diese über den gewählten Zeitraum den entsprechenden Zeiteinheiten an. Dadurch ergibt sich ein dynamisches Öffnen der Tür.

Die Sounds können per Drag and Drop integriert und angepasst werden.

Die Timeline kann in Sekunden oder in Frames eingestellt werden. Während dem Ablaufen der Timeline kann mit den in der Szene enthaltenen Objekten nur sehr begrenzt interagiert werden. Es ist im Flugsimulator zwar möglich, während des Ablaufens der Timeline den Flightstick zu betätigen. Die Änderungen zeigen sich aber erst nach dem Ende der Animation.

## 5. Arduino

Der Arduino wird mit einem USB-C auf USB-B Kabel an den Computer angeschlossen. Anschließend wird ein Breadboard mit folgendem Schaltplan an den Arduino angeschlossen:



Jede LED ist an einen digitalen Pin angeschlossen. Liegt Spannung an dem jeweiligen Pin an, leuchtet die LED. Die Beschriftung der ersten 6 Pins ist wie folgt zu verstehen:

- MxS = Motor x SET
- MxR = Motor x RESET

### 5.1. Code

Der Code des Arduinos ist zwar im GitHub Repository des Projekts mitgeliefert, um aber zu vermeiden das er im Laufe der Zeit verloren geht werden wir hier die wichtigsten Punkt noch einmal einfügen, um das Rekonstruieren des Codes zu ermöglichen.

Arduino bietet ähnlich wie Unity gewisse Lifecycle Funktionen an. Setup() ist vergleichbar mit Unitys onStart() und wird beim Initialen Boot ausgeführt.

**Wichtig:** Setup wird nur beim Powerup des Arduinos ausgeführt. Ein Neustarten der Unity Anwendung ändert nichts daran. Soll der Arduino also auf einen gewissen Ausgangsstate zurückgesetzt werden, muss das manuell geschehen.

```

void setup() {
    Serial.begin(115200); // open serial connection
    initPorts(); // set all digital pins to output and state to LOW
    Serial.setTimeout(500); // read/write timeout for serialport
    inString.reserve(200); // reserve 200 bytes for inputstring
}

```

**Loop ist vergleichbar mit Unitys Update() Funktion.**

```

void loop() {
    if(stringComplete){
        executeCommand(inString);

        //reset command buffer
        inString = "";
        stringComplete = false;
    }
}

```

SerialEvent() ist eine Eventfunktion, die nach jedem Loop ausgeführt wird, sofern neue Daten am Serialport anliegen. Alle neuen Chars werden eingelesen. Sollte eine new Line Char ('\n') eingelesen werden, wird die Methode unterbrochen und die globale Boolean stringComplete auf true gesetzt. Im nächsten Loop wird dann Execute Command aufgerufen. Wie das Protokoll implementiert wurde, wird im nächsten Unterkapitel beschrieben.

```

void serialEvent(){
    while(Serial.available()){
        char inChar = (char) Serial.read();
        inString += inChar;

        //line end = command complete => break out from loop
        if(inChar == '\n'){
            stringComplete = true;
            break;
        }
    }
}

```

## 5.2. Protokoll

Über den Serialport können Daten nur als String gesendet werden. Im Arduino werden die einzelnen Chars ausgelesen und zu einem String zusammengefügt. Arduino bietet dankenswerter Weise grundlegende String-Funktionen an, die in C so nicht vorhanden sind.

Unser Protokoll dient momentan nur den State der digitalen Pins zu setzen. Es wird vorausgesetzt, dass der Pinmode aller digitalen Pins in `setup()` als OUTPUT deklariert wurde.

Der erste Char des Command strings ist das Operationssymbol:

- '>'
  - Set Output on Pin
  - Muss von folgender Struktur gefolgt werden:
  - 'Int portnummer' ':' 'Int state'
  - ">10:0"
    - Setzt Pin 10 auf LOW
- 'r'
  - Setzt den State aller digitalen Pins auf LOW

## 6. Quellen und Credits

---

### 6.1. Tutorials

The Bergison G-Seat:

<https://www.youtube.com/watch?v=PzwfRqSbLzo>

Procedural Landmass Generation von Sebastian Lague (Tutorial Reihe):

<https://www.youtube.com/watch?v=wbpMiKiSKm8>

Grundlagen und Umsetzung der Flugphysik:

In Anlehnung an: <https://github.com/melowntech/VtsJetFighter>

Einbinden von Audio:

Introduction to Audio in Unity – Brackeys

[https://www.youtube.com/watch?v=60T43pvUyFY&t=630s&ab\\_channel=Brackeys](https://www.youtube.com/watch?v=60T43pvUyFY&t=630s&ab_channel=Brackeys)

### 6.2. Assets

Cockpit: <https://sketchfab.com/3d-models/spacefighter-cockpit-wasp-interdictor-db4aa67fe1164ec088083d7dac2d273f>

Startbahn: <https://sketchfab.com/3d-models/runway-and-hangar-cf6a8925f591421a9b82c28c61ad4c47>

Skybox:

<https://assetstore.unity.com/packages/2d/textures-materials/sky/colorskies-91541>

Sounds:

<https://www.audiyou.de/beitrag/start-eines-airbus-a321-9640/>

<https://www.audiyou.de/beitrag/boeing-777-fluggerusch-5978/>

<https://freesound.org/people/qubodup/sounds/162408/>

<https://freesound.org/people/alex36917/sounds/582250/>