

Bearbeitungsbeginn: 01.09.2022

Vorgelegt am: 28.02.2023

# **Thesis**

zur Erlangung des Grades

**Master of Science**

im Studiengang Medieninformatik

an der Fakultät Digitale Medien

**Luis Keck**

**Matrikelnummer: 269379**

**Entwicklung einer raumfüllenden VR-Installation für ein  
kooperatives Piraten-Game**

Erstbetreuer: Prof. Jirka Dell’Oro-Friedl

Zweitbetreuer: Prof. Dr. Norbert Schnell

# Abstract

Diese Arbeit behandelt die Konzeption und Entwicklung von „Plunder Hunt VR“, eines kooperativen VR-Games für drei Spieler. In diesem Spiel übernehmen die Spieler die Rollen einer Piratencrew und steuern zusammen ein Segelschiff. In der Spielwelt befinden sich außerdem computergesteuerte gegnerische Schiffe, die sich in Kanonengefechten versenken lassen.

Gespielt wird in einer für das Projekt entwickelten raumfüllenden VR-Installation mit verschiedenen, anfassbaren Steuerungselementen, deren Inputs über verschiedene Sensorik an das Spiel übermittelt werden. Der Raum simuliert hierbei das Deck des Piratenschiffs. Die Spieler können sich frei bewegen und die Steuerungselemente benutzen, um das Schiff zu steuern, die Segel zu manipulieren und um die Kanonen zu feuern.

Neben der Verwendung einer VR-Installation, hebt sich das Spiel auch dadurch ab, dass die Funktionsweise von Segeln zumindest in den Grundzügen realistisch abgebildet wird. Dadurch müssen Spieler die Segel passend zum Winkelverhältnis von Schiff und Windrichtung einstellen, um optimale Geschwindigkeiten zu erreichen.

Die Spieler verwenden in der Installation keine klassischen VR-Controller, um die Trennung von Virtualität und Realität möglichst gering zu halten. Stattdessen sind alle Inputmethoden anfassbare Steuerungselemente in der VR-Installation. Dazu zählen neben einem Steuerrad zwei Kanonen und ein Luntenstab, um diese zu zünden. Zusätzlich gibt es eine Kurbel um die Länge der Schot und eine Seilrolle um die Länge des Falls einzustellen.

# Inhaltsverzeichnis

1 Einleitung.....	7
2 Motivation und Inspiration .....	8
3 Grundlagen des Segelns .....	10
4 Technische Basis.....	14
4.1 Hardware .....	14
4.1.1 VR-Headset.....	14
4.1.2 Vive Tracker.....	15
4.1.3 Mikrocontroller-Boards .....	15
4.1.4 Drehgeber .....	15
4.2 Software.....	16
4.2.1 Unity.....	16
4.2.2 Blender .....	16
4.2.3 Wavosaur .....	16
4.2.4 Arduino IDE .....	16
5 Gameplay.....	17
6 Graphische Gestaltung .....	18
6.1 Low-Poly-Style .....	18
6.2 Wasser .....	19
6.2.1 Wasser-Shader .....	19
6.2.2 Geschwindigkeitswahrnehmung.....	20
6.3 Inseln und Felsen .....	21
6.4 Schiffe .....	23
6.4.1 Spielerschiff.....	23
6.4.2 Gegnerschiff.....	27
6.5 Charaktere .....	27
6.5.1 Spielercharaktere .....	28
6.5.2 Gegnercharaktere .....	30
6.5.3 Anpassung der Charaktere an die VR-Anwendung.....	31
7 Spielwelt .....	34
7.1 Aufbau.....	34

7.2 Rand der Welt.....	35
7.3 Gestalterische Erweiterungen .....	36
8 VR-Rauminstallation .....	37
8.1 Verfügbarer Raum für die Installation.....	37
8.2 Steuerungselemente .....	37
8.2.1 Steuerrad.....	38
8.2.2 Schot.....	39
8.2.3 Fall .....	41
8.2.4 Kanonen .....	42
9 Mikrocontroller-Input .....	45
9.1 Mikrocontroller-Hardware .....	45
9.2 Mikrocontroller-Software.....	46
9.2.1 Arduino-Code .....	46
9.2.2 Unity-Code .....	46
10 Segelmechanik.....	49
10.1 Wind.....	49
10.2 Schratsegel.....	50
10.3 Rahsegel.....	51
10.4 Geschwindigkeitsberechnung.....	52
10.5 Lenken des Schiffs.....	54
10.6 Darstellung der Schoten .....	54
11 Kanonenmechanik.....	57
11.1 Ausrichten der Kanonen .....	57
11.2 Feuern der Kanonen .....	58
12 Gegnerverhalten.....	59
12.1 Folgen der Route .....	59
12.2 Kampfverhalten .....	60
12.3 Versenken von Gegnern .....	61
13 Sound-Design.....	63
13.1 Ambient-Sounds .....	63
13.2 Aktionsgebundene Sounds .....	64
14 Multiplayer .....	65

14.1 Host.....	66
14.2 Clients .....	67
15 Fazit und Ausblick.....	68
15.1 Aktueller Zustand von VR-Installation und Game .....	68
15.2 Mögliche Erweiterungen und Verbesserungen.....	69
Literaturverzeichnis.....	70
Anhang.....	72
Raumplan.....	72
Konzeptskizze Steuerrad.....	72
Konzeptskizze Schot.....	73
Konzeptskizze Fall.....	73
Konzeptskizze Kanone .....	73
Arduino-Code.....	74
Poster für Vermittlung der Segelgrundlagen .....	75
Übersicht der Spielwelt mit Routen der Gegnerschiffe .....	76
Eidesstattliche Erklärung.....	77

# Abbildungsverzeichnis

Abb. 1: Rahsegel auf der "Astrid" (Quelle: <a href="https://www.esys.org/segel/Rahsegel.html">https://www.esys.org/segel/Rahsegel.html</a> ) .....	11
Abb. 2: Gaffelsegel (viereckig) auf der 'Regina Maris' (Quelle: <a href="https://www.esys.org/segel/Gaffelsegel_Regina_Maris-hq.JPG">https://www.esys.org/segel/Gaffelsegel_Regina_Maris-hq.JPG</a> ) .....	12
Abb. 3: Umsetzung des Wassers-Shaders, noch nicht angepasst an die geplanten Wetterverhältnisse (Quelle: Eigene Aufnahme) .....	20
Abb. 4: Zwei Varianten von Inseln mit Palmen (Quelle: Eigene Aufnahme) .....	22
Abb. 5: Felsen in verschiedenen Formen und Größen (Quelle: Eigene Aufnahme) .....	23
Abb. 6: Drei Varianten von Sandbänken. Zugehörigkeit durch Trennlinien markiert (Quelle: Eigene Aufnahme) .....	23
Abb. 7: Außenansichten des Spielerschiffs (Quelle: Eigene Aufnahme) .....	26
Abb. 8: Spielerschiff aus Sicht eines Spielers am Steuerrad (Quelle: Eigene Aufnahme) .....	26
Abb. 9: Außenansicht eines gegnerischen Schiffs (Quelle: Eigene Aufnahme) .....	27
Abb. 10: Konzeptskizze für "Captain" und fertiges Modell in Unity (Quelle: Eigene Aufnahme) .....	28
Abb. 11: Konzeptskizze für "Sailor1" und fertiges Modell in Unity (Quelle: Eigene Aufnahme) .....	29
Abb. 12: Konzeptskizze für "Sailor2" und fertiges Modell in Unity (Quelle: Eigene Aufnahme) .....	29
Abb. 13: Offiziersuniformen in Fluch der Karibik (Quelle: <a href="https://fluch-der-karibik.fandom.com/wiki/Royal_Navy">https://fluch-der- karibik.fandom.com/wiki/Royal_Navy</a> ) .....	30
Abb. 14: Modell für Crewmitglieder gegnerischer Schiffe in Unity (Quelle: Eigene Aufnahme) .....	31

<i>Abb. 15: Die drei Spielermodelle mit Anpassungen für die VR-Umgebung (Quelle: Eigene Aufnahme).....</i>	<i>33</i>
<i>Abb. 16: Variante eines Schiffswracks auf Insel platziert (Quelle: Eigene Aufnahme) .....</i>	<i>36</i>
<i>Abb. 17: Steuerungselement "Steuerrad" in der Installation und als virtuelles Gegenstück in der Unity-Anwendung (Quelle: Eigene Aufnahme) .....</i>	<i>39</i>
<i>Abb. 18: Steuerungselement "Schot" in der Installation und als virtuelles Gegenstück in der Unity-Anwendung (Quelle: Eigene Aufnahme) .....</i>	<i>40</i>
<i>Abb. 19: Steuerungselement "Fall" in der Installation und als virtuelles Gegenstück in der Unity-Anwendung (Quelle: Eigene Aufnahme) .....</i>	<i>41</i>
<i>Abb. 20: Steuerungselement "Kanone" in der Installation und als virtuelles Gegenstück in der Unity-Anwendung (Quelle: Eigene Aufnahme) .....</i>	<i>43</i>
<i>Abb. 21: Modell des Luntentabs in der Unity-Anwendung (Quelle: Eigene Aufnahme).....</i>	<i>43</i>
<i>Abb. 22: Verkabelung eines Drehgebers mit einem Mikrocontroller-Board am Beispiel des Steuerungselements "Fall" (Quelle: Eigene Aufnahme) .....</i>	<i>46</i>
<i>Abb. 23: Durchhängende (links) und gespannte (rechts) Schot im Vergleich (Quelle: Eigene Aufnahme).....</i>	<i>56</i>
<i>Abb. 24: Brennendes Gegnerschiff (Quelle: Eigene Aufnahme).....</i>	<i>61</i>
<i>Abb. 25: Hinterlassene Ladung eines versenkten Gegnerschiffs und mit Gold befüllte Kisten auf Spielerschiff (Quelle: Eigene Aufnahme) .....</i>	<i>62</i>
<i>Abb. 26: Übersicht über Struktur des Multiplayers (Quelle: Eigene Aufnahme).....</i>	<i>65</i>

# 1 Einleitung

Das Zeitalter der Piraten bot schon seit jeher eine faszinierende Thematik für Medien aller Art. Auch in der Welt der Computerspiele erschien schon im Jahr 1987 „Sid Meier’s Pirates!“ (vgl. Smithsonian American Art Museum, 2015) und das Steuern eines Piratenschiffs wurde immer wieder ein zentraler Bestandteil von Games, wie zum Beispiel im Jahr 2013 erschienenen „Assassin’s Creed IV Black Flag“ des Entwicklungsstudios Ubisoft Montreal (vgl. Ubisoft, o. D.).

Die meisten dieser Spiele verzichteten hierbei jedoch auf eine realistische Abbildung der Segelphysik, da der durchschnittliche Spieler damit nicht vertraut ist und dadurch überfordert wäre. Außerdem wird in vielen Bereichen auf Realismus verzichtet, um Spieler besser zu erreichen. So sind zum Beispiel die Geschwindigkeiten von Segelschiffen oftmals höher als in der Realität, da sie Spielern sonst zu langsam vorkommen würden oder Langeweile auslösen. Zusätzlich wird der Einfluss der Windrichtung auf das Segeln vollständig ignoriert. Der Wind kommt im Normalfall immer von hinten, egal in welche Richtung das Schiff gerade fährt. Das hat zur Folge, dass Segelschiffe sich eher wie ein Auto oder Motorboot verhalten und somit eher den Erwartungen von Segelunerfahren Spielern entsprechen. (vgl. Schell, 2020)

Außerdem fordern Segelschiffe dieser Größe die Zusammenarbeit mehrerer Crewmitglieder, um alle Einstellungsmöglichkeiten des Schiffs zu bedienen. Dafür bräuchte es entweder eine komplexe Steuerung in Singleplayer-Games oder Multiplayer-Games, in die mehrere Spieler zusammen ein Schiff steuern.

Während der Verzicht auf diese Mechaniken in Mainstream-Games nachvollziehbar begründet ist, bietet eine darauf spezialisierte VR-Installation die Möglichkeit realistischeres Segelverhalten mit einem Piratensetting zu kombinieren und ungewöhnliche Inputmethoden zu nutzen, um eine immersive Erfahrung zu erschaffen. Zusätzlich fördert und fordert eine komplexe Steuerung des Schiffs die Kooperation mehrerer Spieler.



## 2 Motivation und Inspiration

Das Ziel dieses Projekts ist also ein kooperatives Piratenspiel zu entwickeln, in dem mehrere Spieler als Crew zusammenarbeiten, um ein Schiff zu manövrieren. Damit gute Kooperation zwischen den Spielern tatsächlich nötig ist, um das Schiff erfolgreich zu steuern, wird bewusst auf die vereinfachte Darstellungsweise des Segelns, wie man sie aus Mainstream-Computerspielen kennt verzichtet. Stattdessen wird die Segelmechanik näher an Realität angelehnt, wobei jedoch keine richtige Simulation entstehen soll, um Segellaien nicht zu überfordern und das Spiel auch für Personen ohne Vorerfahrung nach kurzer Einweisung zugänglich zu machen. Die Umsetzung des Spiels in einer raumfüllenden VR-Installation bietet die Möglichkeit die Immersion für Spieler zu erhöhen. Inputmethoden können als anfassbare Steuerungselemente umgesetzt werden, so dass haptisches Gefühl und Bewegungen in der Realität weitgehend mit der Virtualität in der VR-Brille übereinstimmen. Außerdem stimmen Größen und Distanzen mit der Realität überein. Spieler können sich in gleicher Weise durch den realen Raum und über das Deck des Schiffs bewegen. Zusätzlich ist es in der Installation möglich, die Inputmethoden räumlich so zu trennen, dass immer nur ein Element für eine Person erreichbar ist. Dadurch wird es nötig, dass die Spieler Steuerungsaufgaben unter sich aufteilen und miteinander kommunizieren.

Eines der Spiele, dass den Zielen dieses Projekts am nächsten kommt, ist das 2018 erschienene Action-Adventure „Sea of Thieves“ des Entwicklers Rare (vgl. Microsoft Store, 2018). Zentrales Gameplay-Element dieses Spiels ist unter anderem, dass mehrere Spieler zusammen im Multiplayer ein Piratenschiff steuern können. Das Besondere in diesem Spiel ist es, dass es hier tatsächlich die Steuerung des Schiffs an verschiedenen Stellen über das Deck verteilt wird. Neben dem Steuerrad hat jedes Segel zwei verschiedene Arten von Leinen, die es möglich machen dieses zu manipulieren. Zum einen ist es möglich die Größe des Segels zu verändern, und damit die Geschwindigkeit anzupassen. Je mehr Segelfläche vorhanden ist, desto schneller wird das Schiff. Zieht man die Segel vollständig ein gibt der Wind dem Schiff keine Antriebskraft mehr und es kommt zum Stehen. Die größte Besonderheit ist jedoch, dass die Windrichtung in diesem Spiel tatsächlich einen Einfluss hat. Die zweite Art der Leinen bietet nämlich die Möglichkeit, den Winkel des Segels anzupassen. Ist die Segelstellung optimal im Verhältnis zur Windrichtung wird das Schiff schneller. Jedoch werden Spieler nicht bestraft, sollte die Segelstellung nicht optimal sein. Die Geschwindigkeit wird eher durch die Segelgröße

verändert. Der richtige Umgang mit der Segelstellung dient eher als Bonus. Ein letztes Steuerungselement von Schiffen in „Sea of Thieves“ ist Möglichkeit einen Anker zu lichten. Dies erlaubt es Spielern das Schiff an einer Stelle zu befestigen, ohne dass es durch Wind oder Wellengang abdriftet. Da es in dem Spiel möglich ist das Schiff zu verlassen, um zum Beispiel Inseln zu betreten, ist dies eine notwendige Mechanik, um zu verhindern, dass Spieler ihr Schiff in ihrer Abwesenheit verlieren. (vgl. Sea of Thieves Wiki, 2022)

Da in diesem Projekt der gesamte Raum, in dem die VR-Installation aufgebaut wird als Schiffsdeck dienen soll, wird es Spielern nicht möglich sein das Schiff zu verlassen. Das hat zur Folge, dass eine Ankermechanik nicht von Nöten ist. Allerdings werden ähnliche Steuerungselemente wie in „Sea of Thieves“ umgesetzt, da es sich hier um die grundlegenden Bedienmöglichkeiten eines Segelschiffs handelt. Vorgesehen sind also Inputmethoden, um das Ruder des Schiffs zu steuern, die Größe von Segeln anzupassen und die Stellung von Segeln zum Wind zu verändern. Ein Unterschied wird jedoch sein, dass dieses Projekt mehr Wert darauflegen wird, dass Spieler die Segel optimal bedienen. Falsche Segelstellungen werden also mehr Einfluss auf die Geschwindigkeit des Schiffs haben. Dieses Vorhaben wird auch Einfluss auf die Art der Segel auf dem Spielerschiff haben und erfordern, dass Spieler ohne Segelerfahrung eine kurze Einweisung in die Grundlagen erhalten, um das Verhalten des Schiffs zu verstehen.

### 3 Grundlagen des Segelns

Um die folgenden Spielmechaniken nachvollziehen zu können ist es nötig einige Grundlagen und Begriffe aus der Welt des Segelsports zu erläutern. Da es sich bei dem Spiel nicht um eine Simulation mit Anspruch auf Realismus handelt, sondern lediglich die Grundzüge der Funktionsweise von Segeln abbilden soll, beschränken sich diese Erklärungen auch nur auf die absoluten Grundlagen. Außerdem fokussieren sich die Erläuterungen auf Bereiche, die für dieses Projekt relevant sind. Begriffe werden um die englischen Übersetzungen ergänzt, da diese im Code des Games verwendet werden. Teilweise wurden jedoch auch andere englische Begriffe im Code verwendet, die das Verständnis für Nicht-Segler erleichtern.

Grundsätzlich sind im Segelsport zwei Arten der Richtungsangabe relevant. Zum einen sind da Backbord (engl. Port) und Steuerbord (engl. Starboard). Backbord beschreibt dabei die in Fahrtrichtung linke Seite des Schiffs, Steuerbord die rechte. Diese Richtungen sind also mit dem Schiff fest verbunden. Die Begriffe Luv und Lee werden genutzt, um zu beschreiben wie das Schiff oder die Segel im Verhältnis zur Windrichtung stehen. Luv beschreibt hierbei die dem Wind zugewandte Seite, also die Richtung, aus der der Wind kommt. Lee ist demnach die dem Wind weggewandte Seite, die Richtung, in die der Wind weht. Die Begriffe Luv und Lee kommen im Spielcode nicht vor, werden aber in späteren Erklärungen zu Spielmechaniken relevant.

Mit Piratenschiffen aus dem 16. und 17. Jahrhundert wird zumeist das klassische Rahsegel assoziiert, wie es in der folgenden Abbildung (Abb. 1). zu sehen ist. Dabei handelt es sich um „[...] ein zumeist rechteckiges oder trapezförmiges Segel, das an einem Rah genannten Rundholz geführt wird“ (Europäisches Segel-Informationssystem [1], 2021). Diese Segel haben in der Realität den großen Nachteil, dass man mit ihnen nicht sehr hoch am Wind fahren kann. Gemeint ist damit, dass der Winkel zwischen Windrichtung und Fahrtrichtung des Schiffs im Normalfall mindestens 60-80° betragen muss (vgl. Europäisches Segel-Informationssystem [1], 2021).



Abb. 1: Rahsegel auf der "Astrid" (Quelle: <https://www.esys.org/segel/Rahsegel.html>)

In Computerspielen, die keine Segelsimulationen sind, wird dieser Fakt zumeist ignoriert, indem die Windrichtung immer so angepasst wird, dass der Wind von hinten auf die Segel trifft. In diesem Projekt sollen jedoch die Grundmechaniken des Segelns abgebildet werden und dazu gehört auch, dass der Wind nur aus einer Richtung kommt und die Segel an diese angepasst werden müssen. Eine weitere Konsequenz daraus ist es, dass Segelschiffe nicht direkt entgegengesetzt der Windrichtung fahren können. Ein Rahsegel würde dafür sorgen, dass die Windkraft entgegengesetzt der gewünschten Fahrtrichtung auf das Schiff wirkt. Bei moderneren Schratsegeln flattert das Segel im Wind, ohne dass es dem Schiff Antriebskraft erzeugt. „Schratsegel ist ein Sammelbegriff für alle Segel, die in Ruhestellung in Richtung der Schiffs-Längsachse gesetzt werden“ (Europäisches Segel-Informationssystem [2], 2021). Dieser Sammelbegriff umfasst alle, meist dreieckigen Segel, die man auf modernen Segelschiffen sieht, aber auch ältere Segeltypen, die in das Zeitalter der Piraten passen. Der große Vorteil von Schratsegeln ist, dass sie im Vergleich zu Rahsegeln Kurse sehr viel höher am Wind erlauben. Schratsegel befinden sich immer auf der Lee-Seite des Schiffs und werden durch eine Schot (siehe Begriffserklärung weiter unten) in ihrem Winkel zum Schiff begrenzt. Auf eine Beschreibung der physikalischen Funktionsweise von Schratsegeln wird verzichtet, da sie nicht für das Projekt relevant wird.

Um entgegengesetzt zu Windrichtung zu fahren, müssen Segelschiffe kreuzen. Dabei wird ein Kurs sehr hoch am Wind, also mit einem geringen Winkel zwischen Windrichtung und Fahrtrichtung, gesegelt. Um sich nicht zu weit von Ziel zu entfernen, wird immer wieder eine

Wende durchgeführt. Dabei dreht sich der Bug des Schiffs einmal durch den Wind, sodass der Winkel zum Wind wieder der gleiche ist, nur in die andere Richtung. Schratsegel wechseln hierbei die Seite, da der Wind nach der Wende auf die andere Seite des Segels trifft. Es ist also nicht möglich direkt entgegengesetzt der Windrichtung zu fahren, weshalb Segelschiffe eine Art Zick-Zack-Route wählen. Schratsegel sind hier Rahsegeln überlegen, da sie in einem geringeren Winkel zur Windrichtung gesegelt werden können und somit effizienteres kreuzen erlauben. In alle anderen Richtungen zu Wind lassen sich Segelschiffe geradeaus fahren.

In einem Spiel, bei dem das Segelschiff der Spieler von der Windrichtung abhängig sein soll, ist es also sinnvoll das Schiff mit einem Schratsegel auszustatten, um das Segeln in alle Richtungen möglichst angenehm zu gestalten. Gleichzeitig sind Rahsegel jedoch die Segel, die mit Piratenschiffen am meisten verbunden werden. Das Spielerschiff für dieses Projekt wurde deshalb mit zwei Masten und zwei Segeln ausgestattet. Am vorderen Mast befindet sich ein Rahsegel und am hinteren ein Gaffelsegel, wie es in der folgenden Abbildung (Abb. 2) zu sehen ist. Bei dem Gaffelsegel handelt es sich um eine Unterart der Schratsegel, bei der ein unregelmäßig viereckiges Segel zwischen Gaffel und Baum aufgespannt wird. Der Baum (engl. Boom) ist hierbei ein Rundholz das rechtwinklig am Mast nach hinten angebracht wird und sich um den Mast drehen kann. Es bildet beim Gaffelsegel und vielen anderen Unterarten des Schratsegels die Unterkante der Segelfläche. Die namensgebende Gaffel ist ein weiteres Rundholz, das die obere Kante des Segels bildet und in einem Winkel am Mast angebracht wird, sodass das Segel nach hinten höher wird. (vgl. Europäisches Segel-Informationssystem [3], 2021)



Abb. 2: Gaffelsegel (viereckig) auf der 'Regina Maris' (Quelle: [https://www.esys.org/segel/Gaffelsegel\\_Regina\\_Maris-hq.JPG](https://www.esys.org/segel/Gaffelsegel_Regina_Maris-hq.JPG))

Im Projektcode werden die beiden Segel als „FrontSail“ und „BackSail“ bezeichnet, um sie leichter zuordnen zu können.

Das Gaffelsegel lässt sich durch ein Seilzugsystem, das zwischen Baum und Schiffsdeck aufgespannt wird, bedienen. Dieses wird Schot (engl. Sheet) genannt und erlaubt es den Winkel zwischen Segel und Schiff zu verändern, um diesen an das Verhältnis zwischen Wind und Fahrtrichtung anzupassen. Vereinfacht gesagt ist es dabei nötig das Segel durch Verlängern der Schot weiter zu öffnen, je größer der Winkel zwischen Windrichtung und Schiff wird, um die höchstmögliche Geschwindigkeit zu erreichen. Die genauen Winkel sind hierbei von Schiffstyp und Segeln abhängig. Da es sich in diesem Projekt jedoch um ein fiktives Schiff handelt, zu dem die dafür benötigten Daten nicht vorliegen wurden, die Winkel frei gewählt. Sie orientieren sich jedoch an realistischen Verhältnissen.

Ein weiterer Bestandteil von Segelschiffen der für dieses Projekt noch relevant wird ist das Fall (engl. Halyard). Hierbei handelt es sich um eine Leine, die es erlaubt Segel zu setzen. Im Normalfall ist das Seil oben am Segel befestigt, sodass es hochgezogen werden kann.

Zu guter Letzt wird der Verklicker, beziehungsweise Windanzeiger relevant. Dabei handelt es sich um kleine Fähnchen oder pfeilförmige Anzeiger, die an der Spitze des Masts angebracht sind. Ein Verklicker dient dazu den Seglern die Windrichtung anzuzeigen, damit sie wissen, wie sie ihre Segel einstellen müssen. Die angezeigte Richtung ist hierbei eigentlich nicht die tatsächliche Windrichtung, sondern der sogenannte „scheinbare Wind“, eine Mischung aus der tatsächlichen Windrichtung und dem Fahrtwind des Schiffs (vgl. Tom, o. D.). Für dieses Projekt wird der Verklicker allerdings die tatsächliche Windrichtung anzeigen, um den Umfang an benötigten Informationen für neue Spieler zu reduzieren.

## 4 Technische Basis

Das folgende Kapitel beschreibt die Hard- und Software, die für dieses Projekt verwendet wurde, beziehungsweise zur Verfügung stand.

### 4.1 Hardware

#### 4.1.1 VR-Headset

Zentraler Bestandteil der Hardware für das Projekt sind die VR-Headsets die Spieler tragen werden. Zur Verfügung steht zum einen eine HTC Vive Pro. Dieses Headset ist mit zwei AMOLED-Bildschirmen ausgestattet, die eine örtliche Auflösung von 1440 x 1600 Pixel pro Auge haben. Das Gesichtsfeld beträgt laut Hersteller 110° und die Bildschirme haben eine Bildwiederholrate von 90 Hz. Vorteil dieses Headsets ist zum einen, dass es mit Kopfhörern ausgestattet ist, und somit keine weiteren Lautsprecher nötig sind. Außerdem ist es mit einem Wireless Adapter ausgestattet, sodass sich das Headset kabellos mit einem Computer verbinden lässt. Da sich die Spieler in der raumfüllenden Installation frei bewegen können sollen, ist eine Unabhängigkeit von Kabeln von Vorteil. (vgl. HTC Corporation [1], o. D.)

Bei den weiteren VR-Headsets, die für das Projekt verfügbar waren, handelt es sich ebenfalls um HTC Vives, jedoch um Vorgängermodelle der HTC Vive Pro. Abgesehen von schlechteren technischen Daten bieten sie aber alle Funktionalitäten, die für das Projekt nötig sind. Der größte Nachteil dieser Headsets ist es, dass sie durch Kabel mit dem Computer verbunden werden müssen und dadurch Spieler in ihrer Bewegungsfreiheit einschränken. Der Raum in der die VR-Installation aufgebaut wurde ist mit zwei SteamVR Basisstationen der ersten Generation ausgerüstet, die Outside-In-Tracking erlauben. Outside-In-Tracking ist eine Trackingmethode bei den Kameras oder wie in diesem Fall andere Sensoren genutzt werden, „die an einem stationären Ort platziert und auf das getrackte Objekt (z. B. ein VR-Headset) ausgerichtet sind“ (World of VR, o. D.). Das Verfügbare Equipment verwendet hierbei stationäre Basiseinheiten, die Lighthouses genannt werden, und verschiedene Lichtmuster in den Raum werfen. Die HTC-Headsets sind mit Fotowiderständen ausgerüstet und können mit diesen ihre Lage und Position im Raum bestimmen. Diese Daten werden mit Hilfe von SteamVR berechnet, das auch als Schnittstelle zu Unity genutzt wird. Vorteil dieses Konzepts ist es, dass die Genauigkeit durch zusätzliche Lighthouses erhöht werden kann. Nachteil ist es, dass es bei Verdeckungen zwischen Headset und Lighthouse zu Tracking-Ausfällen kommen

kann. In einer Installation, in der sich mehrere Aufbauten und Personen befinden, kann dies zu Problemen führen. Außerdem ist dieses Setup in seiner Größe beschränkt. (vgl. World of VR, o. D.).

#### 4.1.2 Vive Tracker

Als weiteres VR-Equipment werden in diesem Projekt drei Vive-Tracker der dritten Generation genutzt. Dabei handelt es sich um kleine controllerartige Sensoren, die jedoch keine Tasten als Inputmethode bieten. Jedoch lassen sich ihre Rotation und Position im Raum tracken, zusammen mit dem Headset über die Lighthouses und SteamVR. Ein typischer Anwendungsbereich der Vive-Tracker ist die Ganzkörperverfolgung. Dabei werden mehrere Tracker an verschiedenen Körperstellen befestigt, um alle Bewegung der Gliedmaßen erfassen zu können. (vgl. HTC Corporation [2], o. D.)

Allerdings lassen sich einzelne Tracker auch wie in diesem Projekt beliebig an Objekten befestigen um diese als VR-Controller zu nutzen. Hilfreich ist hierbei das Gewinde der Tracker, mit dem sie sich an normale Stativschrauben anbringen lassen.

#### 4.1.3 Mikrocontroller-Boards

Um die Sensordaten einiger Inputmethoden in der VR-Installation zu verarbeiten, wurden Mikrocontroller eingesetzt. Verwendet wurden hierbei drei Mikrocontroller-Boards die Baugleich zum Arduino UNO sind. Auf ihnen laufen kleine C++-Programme, die einkommende analoge oder digitale Daten von angeschlossenen Sensoren verarbeiten und an einen Computer in Form von verwertbaren Variablen weitergeben. Die Variablen lassen sich wiederum mit Hilfe einer COM-Port-Schnittstelle in Unity abgreifen.

#### 4.1.4 Drehgeber

Bei den an die Mikrocontrollern angeschlossenen Sensoren handelt es sich um Drehgeber. Verwendet wurden drei optische Drehgeber, auch optische Encoder genannt, um Drehbewegungen der Inputmethoden zu erfassen. Die Drehbewegung wird über eine Welle an den Sensor übertragen und rotiert dadurch eine mit Löchern versehene Scheibe. Auf der einen Seite der Scheibe befinden sich zwei Lichtquellen. Gegenüber auf der anderen Seite sind zwei optische Detektoren. Je nach Rotation der Scheibe werden beide, keine oder nur eine der beiden Lichtquellen durchgelassen. Durch eine Versetzung der Löcher für die beiden



Lichtquellen entstehen zwei Phasen (A und B). Durch diese Phasenverschiebung ist es möglich die Drehrichtung des Sensors zu erfassen. (vgl. Electric Diy Lab, 2020)

Die verwendeten Sensoren haben eine Auflösung von 1.200 Werten pro volle Umdrehung. Sie sind nicht in der Lage ihre Rotation nach Trennung vom Strom zu speichern, sodass sie immer mit dem Wert 0 starten und je nach Drehrichtung diesen Wert um 1.200 pro 360° erhöhen oder verringern.

## 4.2 Software

### 4.2.1 Unity

Unity ist eine Entwicklungsumgebung und Game-Engine mit der sich Computerspiele und ähnliche Anwendungen entwickeln lassen. Außerdem ist es mit Unity möglich ein Projekt für verschiedene Plattformen zu exportieren. Durch verschiedene Plugins kann Unity um Funktionen erweitert werden. So wird in diesem Projekt zum Beispiel „OpenXR“ genutzt, um die benötigten VR-Funktionalitäten zu bieten. In Unity wurde durch eingebaute Funktionen und C#-Skripte die Spiellogik und der Szenenaufbau festgelegt. Außerdem wurden Shader für dieses Projekt mit dem Unity *Shader Graph* erstellt.

### 4.2.2 Blender

Blender ist eine 3D-Grafiksuite, die es erlaubt 3D-Modelle zu modellieren, texturieren und animieren. Mit diesem Tool wurden sämtliche Modelle für dieses Projekt erstellt. Aufgrund des gewählten Grafikstils wurde auf Texturen weitgehend verzichtet.

### 4.2.3 Wavosaur

Wavosaur ist ein Audio-Editor, mit dem sich die Sounddateien, die für dieses Projekt genutzt wurden, anpassen ließen. Dabei ging es meist darum Lautstärken anzugleichen, Clip-Längen anzupassen oder ungewünschte Geräusche zu entfernen.

### 4.2.4 Arduino IDE

Arduino IDE ist eine Entwicklungsumgebung, in der sich kleine C++-Skripte schreiben lassen, die man anschließen per USB-Kabel auf Mikrocontroller-Boards übertragen kann. Diese Programme haben zumindest in diesem Projekt den Zweck, die Sensordaten der Drehgeber in nutzbare Variablen umzuwandeln.

## 5 Gameplay

Da der Fokus dieses Projekts eher auf der Umsetzung der VR-Installation liegt ist das Gameplay für die erste Umsetzung dieses Projekts recht einfach gestaltet. Drei Spieler befinden sich zusammen auf einem Piratenschiff und bilden somit dessen Crew. Man startet auf dem Wasser und beginnt damit das Schiff zu segeln. Dabei gibt es verschiedene Inputmethoden um Ruder, Segelflächen und Segelstellungen zu beeinflussen, die in einem späteren Kapitel genauer erläutert werden. In der Spielwelt bewegen sich außerdem weitere computergesteuerte Schiffe, die zunächst zirkuläre Routen fahren. Ziel der Spieler ist es, diese Schiffe zu versenken und danach die auf dem Wasser treibende Ladung aufzusammeln. Nähert sich das Spielerschiff einem Computergegner, hört dieser auf seine Route zu verfolgen und beginnt zunächst geradeaus zu fahren. Außerdem nutzt das Gegnerschiff seine Kanonen, um das Spielerschiff zu beschießen, sollte es in Reichweite sein. Ist das Spielerschiff in Sichtweite, aber nicht mit den Kanonen erreichbar, passt das Gegnerschiff seinen Kurs an, um weiter schießen zu können. Entkommt das Gegnerschiff, kehrt es zu seiner ursprünglichen Route zurück. Das Spielerschiff ist auf der Backbord- und Steuerbordseite jeweils mit einer Kanone ausgerüstet. Spieler haben die Möglichkeit die Ausrichtung der Kanonen auf horizontaler und vertikaler Achse zu verändern, um mit dieser zu zielen und können durch Zünden der Lunte einen Schuss abfeuern. Treffen die Spieler ein Gegnerschiff nimmt es Schaden und beginnt zu brennen. Bei einem erneuten Treffer sinkt das Gegnerschiff und hinterlässt einen Teil seiner Ladung auf dem Wasser schwimmend. Fahren die Spieler mit ihrem Schiff durch die Ladung verschwindet diese und der Plunder wird in Form von Gold auf das Spielerschiff geladen. In der aktuellen Umsetzung des Projekts ist es nicht möglich, dass das Schiff der Spieler von Gegnern versenkt wird. Die Installation soll dazu genutzt werden das Konzept zu zeigen und auszuprobieren. Ein Abbruch des Spiels und der benötigte Neustart ist hierbei eher hinderlich, da eher einmalige Tests der Anwendung vorkommen werden und somit die Spieler nicht wirklich ihre Fähigkeiten durch Erfahrung ausbauen können.

In der aktuellen Umsetzung befinden sich fünf gegnerische Schiffe, die versenkt werden können in der Spielwelt. Wird dies von den Spielern erreicht, sind ihre Plunderkisten voll und sie können weiter in der Spielwelt herumsegeln.

## 6 Graphische Gestaltung

Grundsätzlich soll das Spiel bei eher düsterem Wetter auf dem Meer stattfinden. Deswegen wurde für die virtuelle Lichtquelle die als Sonne dient ein eher bläulicher Farbton gewählt. Auch das Wasser ist eher dunkelblau-gräulich. Am Himmel ist die Sonne nicht zu sehen. Auch einzelne Wolken lassen sich nicht erkennen, da eine graue Wolkendecke dargestellt werden soll. Die Sichtweite der Spieler wird durch einen Nebel eingeschränkt, der auch weiter zur Stimmung beiträgt und durch Unity-eigene Rendereinstellungen erzeugt wurde. Abgerundet werden die Wetterverhältnisse durch einen leichten Regen, der mit dem Unity-Partikelsystem erstellt wurde und dessen Tropfen auch ringförmige Partikel erzeugen, wenn sie auf das Wasser treffen, um die Kollision zu visualisieren.

### 6.1 Low-Poly-Style

Bei der Wahl des Grafikstils für dieses Projekt gab es drei Faktoren, die beachtet werden mussten. Ein erster war es, dass die 3D-Modelle der Segelschiffe, vor allem für das Spielerschiff, sehr genaue Bedingungen erfüllen mussten. Die Größe des Schiffs ist abhängig von der nutzbaren Fläche des Raums in der die VR-Installation aufgebaut wurde. Anfassbare Elemente der Installation brauchen virtuelle Gegenstücke die ausreichend ähnlich in ihrer Form sind, sodass Spieler die echten Objekte greifen können, auch wenn sie nur die virtuelle Welt sehen. Die Art und Verteilung von Segeln wurde auch mit Rücksicht auf geplante Spielmechaniken gewählt. Eine weitere Besonderheit, die das Spielerschiff erfüllen muss, ist dass das begehbare Deck eine ebene Fläche sein muss, um den realen Boden widerzuspiegeln. Modelle für Piratenschiffe haben typischerweise eine erhöhte Ebene für das Steuerrad oder haben je nach Stil auch ein gebogenes Deck. Diese möglichen Diskrepanzen könnten Spieler irritieren und sollten deshalb nicht entstehen. Die beste Lösung ist es also selbst erstellte 3D-Modelle zu verwenden. Auf diese Weise wird die vollständige Kontrolle über die Gestaltung des Spielerschiffs und weiterer Modelle gewährleistet. Um den Grafikstil einheitlich zu halten, ist es außerdem sinnvoll alle Modelle selbst zu erstellen. Das hat allerdings auch zur Folge dass der zeitliche Aufwand für das Beschaffen von 3D-Modellen steigt. Es ist also nötig einen Grafikstil zu wählen der zeitsparende Modelliermethoden erlaubt. Aus diesem Grund wurde für das Projekt ein Low-Poly-Style gewählt. Objekte bestehen aus großen dreieckigen Flächen und werden nicht unbedingt texturiert. Stattdessen erhält jede Fläche eines Modells eine

einzigste Farbe. Durch einen *Flat*-Shader wird das typische Aussehen des Low-Poly-Style vervollständigt. Dadurch dass Formen nur grob geometrisch imitiert werden, wird Modellierzeit gespart und es ist somit möglich in gleicher Zeit mehr Vielfalt in die 3D-Gestaltung des Spiels zu bringen. Ein weiterer Vorteil dieser Modellierteknik, ist es das die resultierenden Modelle durch ihre geringere Anzahl an Polygonen weniger Renderleistung von Computern verlangen. Da VR-Anwendungen rechenintensiv sind und in der Projektzeit kein Fokus auf Optimierung gelegt werden kann, verringert sich so das Risiko, dass die verfügbaren Computer nicht genügen Leistung liefern.

Die erstellten Materialien für 3D-Modelle sind alle mit einem simplen *Flat*-Shader ausgestattet, der eine Farbe für eine Fläche auf Grund des Lichteinfalls und der angegebenen Basisfarbe des Materials berechnet, ohne die Kanten des Modells zu glätten. Außerdem sind die Materialien für die meisten Objekte matt gehalten. Ausnahmen bilden der Wasser-Shader und der *Foliage*-Shader, da hier weitere Berechnung wie zum Beispiel Verschiebung von *Vertices* stattfinden. Außerdem erhält das Gold, dass die Kisten der Spieler als Plunder füllt einen anderen Shader um es glänzend darzustellen.

## 6.2 Wasser

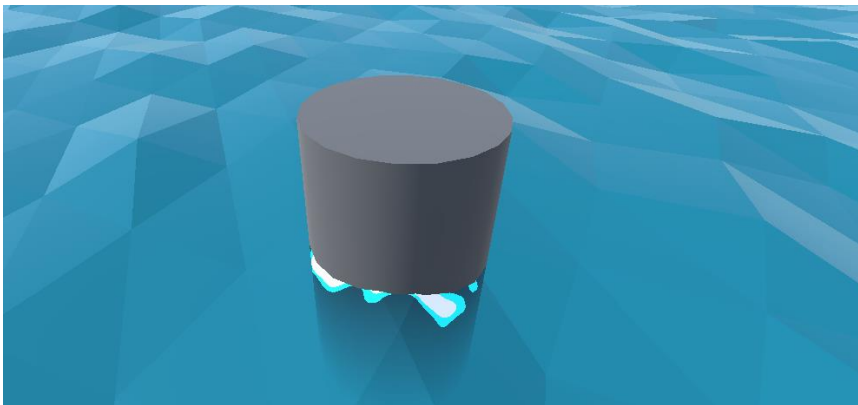
Ein erster Schritt in der grafischen Gestaltung des Spiels war die Erstellung einer Wasserfläche. Sie dient als Grundlage der Spielwelt. Als 3D-Modell dient hier eine einfache *Plane*, die in kleinere dreieckige Flächen unterteilt wurde. Ein Shader soll dazu dienen anhand *Vertices* der *Plane* Wellen zu generieren und passend einzufärben.

### 6.2.1 Wasser-Shader

Der für dieses Projekt verwendete Wasser-Shader basiert auf einem Tutorial des YouTube-Kanals "Flaroon" (vgl. Flaroon, 2021) und wurde im Unity-eigenen *Shader Graph*, einem Node-basiertem Tool, erstellt. Da dieser Shader schon mehrere Einstellungsmöglichkeiten, um Farben und Verhalten des Wassers anzupassen enthält war es möglich die geplanten Wetterverhältnisse darzustellen, ohne den Shader an sich zu verändern. Er wurde jedoch, wie im folgenden Unterkapitel beschrieben, noch erweitert.

Grundsätzlich besteht der Shader aus drei Komponenten. Der erste Teil des Shaders verändert die Position der einzelnen *Vertices* der verwendeten *Plane*. Hierzu wird eine *Gradient Noise*-

Node verwendet, um die Höhe der einzelnen *Vertices* zufällig anzupassen. Ein Parameter „Noise Scale“ erlaubt es die Ausprägung dieses Effekts zu kontrollieren. Durch Abhängigkeit von der Zeit bewegt sich der Noise Diagonal über die *Plane*, um Wellen zu erzeugen. Die Farbe des Wassers wird durch simples Flat-Shading bestimmt. Anders als bei den Materialien für die meisten Objekte im Spiel wurde das Wasser nicht matt gestaltet, sondern reflektiert Lichtquellen, um eine glaubhaftere Wasseroberfläche darzustellen. Die Farbgestaltung des Wasser-Shaders beinhaltet außerdem eine Komponente, die es erlaubt Teile des Objekts unter der Wasseroberfläche in einer frei wählbaren Farbe als Silhouette darzustellen. Sinnvoll ist es hier eine dunklere Abstufung der verwendeten Wasserfarbe zu benutzen. Die letzte Komponente des Shaders sorgt dafür, dass bei Objekten die sich teilweise im Wasser befinden, eine Art Schaum um das Objekt bildet. Dieser Effekt ist auch auf der folgenden Abbildung (Abb. 3) sichtbar.



*Abb. 3: Umsetzung des Wassers-Shaders, noch nicht angepasst an die geplanten Wetterverhältnisse (Quelle: Eigene Aufnahme)*

### 6.2.2 Geschwindigkeitswahrnehmung

Ein Problem, dass bei Tests der Anwendung auffiel war, dass es Spielern schwierig fällt die Geschwindigkeit des Schiffes einzuschätzen. Grundsätzlich muss gesagt werden, dass die tatsächliche Geschwindigkeit des Schiffs in keiner Weise auf realistischen Verhältnissen basiert, sondern durch das Testen der Anwendung angepasst wurde. Dadurch wurden Geschwindigkeiten festgelegt, die für die menschliche Wahrnehmung und das Gameplay angenehm sind. Nicht so langsam, dass Langeweile entsteht und nicht so schnell, dass die Steuerung zu schwer wird oder Motion-Sickness entsteht. Da die VR-Installation die Beschleunigung des Schiffs nicht imitieren kann, können Spieler Geschwindigkeit nur visuell in

der virtuellen Welt wahrnehmen. Hier kommen jedoch die Schwächen des gewählten Low-Poly-Styles der Spielwelt zu Vorschein. Da Objekte aus großen einfarbigen Flächen ohne Textur bestehen, fehlen Spielern visuelle Referenzpunkte, um die relative Geschwindigkeit zwischen Schiff und Wasser wahrzunehmen. Die Wasseroberfläche besteht zwar aus verschiedenen Blaustufen, diese sind jedoch durch eine Rauschtextur generiert worden und bieten somit auch nur bedingt visuelle Ankerpunkte. Die Welt ist auch mit verschiedenen Landflächen wie Inseln, Felsen und Sandbänken gefüllt, die diesem Problem entgegenwirken. Sie können jedoch nur als Referenz genutzt werden, wenn sich das Schiff in der Nähe befindet. Fahren die Spieler auf offenem Wasser ohne Land in der Nähe, wird die Geschwindigkeitswahrnehmung weiterhin eingeschränkt.

Um diesem Problem entgegenzuwirken, wurde der Wasser-Shader erweitert. Zunächst wurden zwei Texturen erstellt, die jeweils eine kleine „Schaumfläche“ darstellen, grafisch angelehnt an den Schaum, der durch den ursprünglichen Shader an Rändern von Objekten generiert wird. Diese Texturen werden in zufälliger Verteilung auf der Wasseroberfläche platziert, wobei dabei auch Leerstellen entstehen können, um eine zu große Dichte von Schaumtexturen zu verhindern. Um weiter zu verhindern, dass sichtbare Muster entstehen, werden die Texturen außerdem in 90°-Intervallen rotiert, sodass ihre Ausrichtung auch zufällig ist. Damit sich die Texturen der *Vertex*-Bewegungen des Wasser-Shaders anpassen werden diese in die Farbgenerierung eingespeist, sodass die Texturen sich auch den Kantenknicken der *Plane*-Polygone anpassen.

Die Schaumflächen, die nun zufällig über die Wasseroberfläche verteilt sind, bieten Spielern visuelle Referenzpunkte, um die Geschwindigkeit des Schiffs einzuschätzen, auch wenn es sich nicht in der Nähe von Landmassen befindet.

## 6.3 Inseln und Felsen

Es wurde sich dagegen entschieden das Spielgeschehen auf offenem Meer stattfinden zu lassen. Stattdessen sollte die Spielwelt mit verschiedenen Landmassen gefüllt werden. Neben der graphischen Erweiterung, die Spielern geboten wird haben diese Landmassen auch Einfluss auf das Gameplay. Zum einen bilden sie Hindernisse die von Schiffen umfahren werden müssen. Außerdem können sie als visuelle Blockade zu Gegnern dienen, vor allem jedoch als physische Barrieren die Kanonenschüsse abfangen können. Außerdem fördern sie

wie erwähnt die Geschwindigkeitswahrnehmung von Spielern, in dem sie einen visuellen Referenzpunkt bieten, der genutzt werden kann, um die relative Geschwindigkeit des Schiffs einzuschätzen. Damit Schiffe und Kanonenkugeln nicht durch die Landobjekte fahren, beziehungsweise fliegen können, wurden sie mit *Collidern* versehen, die auf Kollisionen mit verschiedenen Objekten reagieren können. Das Spielerschiff wird somit von Landobjekten gestoppt. Ein C#-Skript sorgt zusätzlich dafür, dass das Schiff rotiert wird, um zu verhindern, dass es in einer Insel stecken bleibt. Kanonenkugeln erzeugen einen passenden Partikeleffekt, der den Einschlag darstellen soll. Die Interaktion mit gegnerischen Schiffen wird in Kapitel 12 erklärt.

Für die aktuelle Umsetzung des Projekts wurden drei Arten von Landmassen modelliert, die genutzt werden können, um die Spielwelt zu füllen. Um mehr Vielfalt zu erzeugen, gibt es außerdem mehrere Varianten jeder Landart. Der erste Typ ist eine kleine Sandinsel. Auf ihr befinden sich mehrere Palmen, die teilweise Kokosnüsse in verschiedenen Anzahlen tragen. Auf der zweiten Variante der Insel sind außerdem zusätzlich kleine Felsen platziert, wie auf der folgenden Abbildung (Abb. 4) zu sehen.

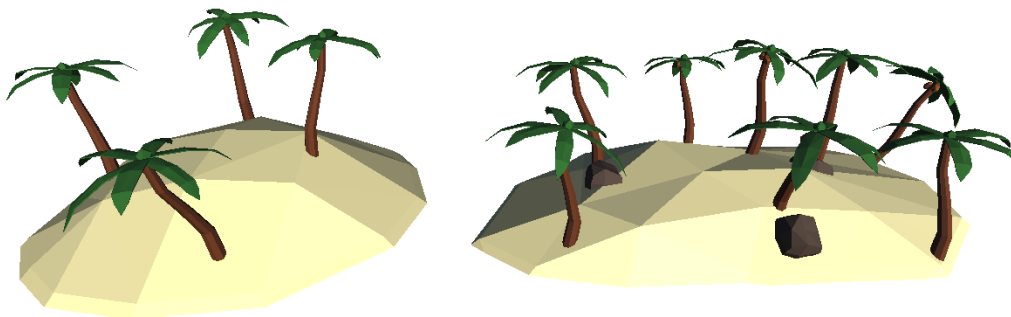


Abb. 4: Zwei Varianten von Inseln mit Palmen (Quelle: Eigene Aufnahme)

Für die Blätter der Palmen wurde eine Erweiterung des verwendeten Flat-Shaders erstellt. Diese sorgt dafür, dass sich die Blätter im Wind bewegen, wobei Richtung und Stärke hier aktuell nicht von den für die Segelmechanik verwendeten Windbedingungen abhängig sind. Bei der zweiten Art von Landmasse handelt es sich um Felsen, die in verschiedenen Formen und Größen gestaltet wurden (Abb. 5).

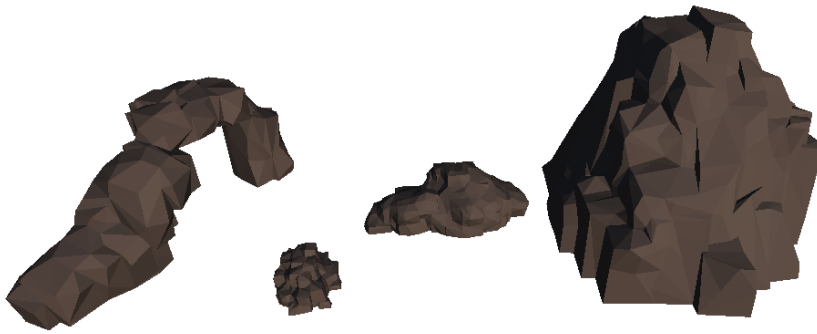


Abb. 5: Felsen in verschiedenen Formen und Größen (Quelle: Eigene Aufnahme)

Die dritte und letzte Art der Landmasse ist die Sandbank. Auch hier wurden verschiedene Formen und Größen erstellt, wobei bei einer Variante drei einzelne Sandbänke ein Objekt bilden, wie in der folgenden Abbildung (Abb. 6) markiert wurde.

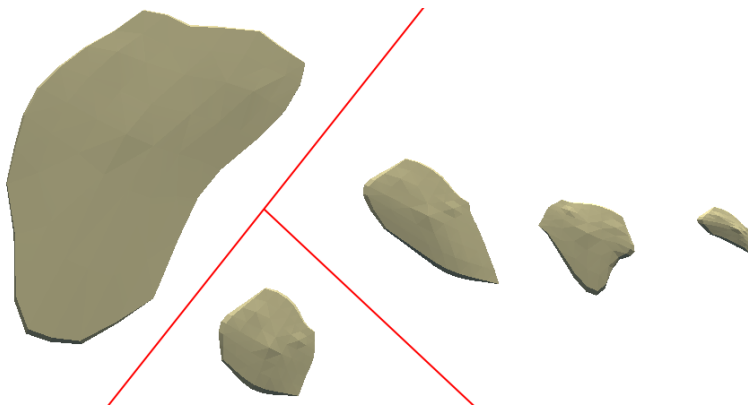


Abb. 6: Drei Varianten von Sandbänken. Zugehörigkeit durch Trennlinien markiert (Quelle: Eigene Aufnahme)

## 6.4 Schiffe

### 6.4.1 Spielerschiff

Für die Entwicklung des Spielerschiffs wurde darauf verzichtet einen real existierenden Schiffstyp zu imitieren. Die Anforderungen an das Schiff sind sehr limitierend, sodass es besser war das Schiff auf Basis seiner Anforderungen zu erstellen, anstatt ein existierendes Modell an die Anforderungen anzupassen.

Ein erster Faktor, der die Bauweise des Schiffs bestimmt ist, der Raum in der die VR-Installation aufgebaut wurde. Die Größe des begehbaren Decks wird durch die tatsächlichen Raummaße (~6,10 m \* 4,10 m) bestimmt. Um die gesamte Breite des Raums nutzen zu können und dem



Schiff dennoch ein passendes Länge-zu-Breite-Verhältnis zu geben, ist nur der hintere Teil des virtuellen Decks begehbar, während es nach vorne verlängert wird, um das Schiff im Verhältnis schmaler zu machen. Der vordere Teil des Decks befindet sich außerhalb des realen Raums, aber kann in der virtuellen Welt gesehen werden. Zusätzlich wird die begehbare Fläche des Decks eingeschränkt, da sich in einer Ecke des Raums zwei Trennwände befinden, die einen kleinen Raum (~2,00 m \* 1,50 m) bilden in dem sich ein Tisch mit dem Computer befindet, der für die kabellose Vive Pro verwendet wird. Der PC-Raum befindet sich im Spiel in der hinteren linken Ecke des Spielerdecks. Um zu verhindern, dass Spieler gegen die Wände laufen, werden auf dem Deck in der Virtualität Kisten und Fässer platziert, die eine visuelle Abgrenzung darstellen. Ein Teil dieser Behältnisse dient außerdem dazu den gesammelten Plunder der Spieler darzustellen, weshalb sie nach oben offen sind und sich im Spielverlauf mit Gold füllen. Der Aufbau des Raums hatte auch Einfluss auf die Positionen der einzelnen Steuerungselemente der Installation (genauer in Kapitel 8).

Ein zweiter Faktor, der das Aussehen des Schiffs beeinflusst war, die Wahl der Segeltypen und Steuerungselemente, die den Spielern zur Verfügung stehen sollen, um das geplante Gameplay umzusetzen. Die Form des Rumpfs und der Segel sind zwar angelehnt an Schiffe des 16. und 17. Jahrhundert, aber wurden frei Hand ohne konkrete Vorlage modelliert. Durch die geringe Raumgröße ist das Schiff deutlich kleiner als man aus Piratenfilmen oder -spielen gewohnt ist, weshalb es auch nur zwei Segel hat. Man könnte es eventuell als Schoner bezeichnen, einem Schiff mit zwei oder mehr Masten bei dem der hintere Mast höher ist und das grundsätzlich mit Gaffelsegeln ausgestattet ist. Auch Rahsegel sind bei Schonern möglich, sie werden typischerweise aber anders als hier angebracht. Schoner sind auch mit anderen Segeltypen zusätzlich geriggt, auf die hier jedoch verzichtet wurde.

Ein letzter Faktor der Einfluss auf die Gestaltung des Schiffs hatte war, dass der Boden des VR-Raums eine ebene Fläche bildet. Das hat zur Folge das auch im 3D-Modell des Spielerschiffs das Deck flach ist. Eine erhöhte Ebene für das Steuerrad die über ein paar Stufen betretbar ist konnte hier nicht umgesetzt werden. Zu einem wegen dem Bauaufwand, der hier entstehen würde, aber vor allem weil sich die Tür des Raums nach innen öffnet und sich auf der Hinterseite des virtuellen Schiffs befindet. Ein erhöhter Aufbau würde die Tür blockieren und müsste deswegen bewegbar sein.

Mit Rücksicht auf diese Faktoren entstand ein Schiff, dessen Rumpf in der Breite den realen Raummaßen entspricht. Es ist jedoch länger, um ein passenderes Verhältnis zwischen Länge und Breite des Schiffs zu schaffen. Das Schiff wirkt dennoch ein wenig in der Länge gestaucht, was jedoch durch den gewählten Grafikstil nicht störend wirkt. Abgesehen davon können Spieler das Schiff nicht von außen betrachten, sodass die Stauchung nicht auffällt. Auf dem Schiff befinden sich zwei Masten, an denen jeweils ein Segel befestigt ist. Der hintere Mast ist ein wenig höher und an ihm ist ein Gaffelsegel befestigt, das zwischen einem Baum und einer Gaffel aufgespannt wird. An der Spitze des Masts befindet sich ein Verklicker, der den Spielern bei der Bestimmung der Windrichtung helfen wird. Im Gaffelsegel befinden sich außerdem Windfäden den Spielern anzeigen, wie das Segel eingestellt werden muss (Erklärung in Kapitel 10). Am vorderen Mast befindet sich ein Rahsegel, das an einer Rah aufgehängt ist. Die Segel sind durch Ringe mit dem Mast und weiteren Rundhölzern verbunden die später als Rotationspunkte dienen. Auch die Hierarchie der einzelnen Objekte wurde anhand der geplanten Segelmechanik bestimmt. Zwischen den Masten und weiteren Teilen des Schiffs sind Seile aufgespannt. Dabei handelt es sich um die Seile, die sich auch in der laufenden Anwendung nicht bewegen werden und somit im Modell fest eingebaut werden können. Bewegliche Seile werden hingegen wie später erklärt in Unity generiert.

Auf dem Deck befinden sich die virtuellen Gegenstücke zu den Steuerelementen der VR-Installation. Die Teile der Objekte, die von Spielern berührt werden, sind in ihrer Form möglichst identisch zu den realen Gegenständen, wohingegen der Rest visuell an das Segelschiff angepasst wurde. Konkret handelt es sich bei diesen Objekten um das Steuerrad, Seilrollen für Schot und Fall und die zwei Kanonen. Genauere Erklärungen zum Aufbau dieser Steuerungselemente folgen in Kapitel 8. Das fertige Modell des Spielerschiffs in Unity ist auf der folgenden Abbildung (Abb. 7) zu sehen.



*Abb. 7: Außenansichten des Spielerschiffs (Quelle: Eigene Aufnahme)*

Abgerundet wird das Schiff durch visuelle Elemente, die nicht für das Gameplay relevant sind. Dazu gehören die Bugspriet, ein Rundholz das am Bug nach vorne gerichtet angebracht ist, kleine Stapel mit Kanonenkugeln und ein Holzgitter, das einen Laderaum im Rumpf andeutet. Außerdem befinden sich Fässer und Kisten auf dem Schiff, teilweise dekorativ und teilweise mit Funktionen wie Raumbegrenzung und Anzeige des Spielfortschritts. Die tatsächliche Sichtweise auf das Schiff von Spielern wird in der folgenden Abbildung (Abb. 8) dargestellt, wobei das Sichtfeld in der VR-Brille abweichen kann.



*Abb. 8: Spielerschiff aus Sicht eines Spielers am Steuerrad (Quelle: Eigene Aufnahme)*

### 6.4.2 Gegnerschiff

Um einen stilistischen Zusammenhang zwischen Spieler- und Gegnerschiff zu schaffen, basiert das Gegnerschiff auf dem Modell des Spielerschiffs. Es ist ein wenig größer als das Spielerschiff und durch die geringere Anzahl an Anforderungen näher am ursprünglichen Design, das für das Spielerschiff vorgesehen war. So ist das Steuerrad mittig platziert und steht auf einer erhobenen Plattform. Es ist außerdem größer, da es nicht wie beim Spielerschiff an die Maße des realen Steuerrads gebunden ist. Auf Elemente die Spieler beim Segeln unterstützen wie Verklicker und Windfäden wurde verzichtet. Das Gegnerschiff ist auch ein Zweimaster, allerdings ist hier an beiden Masten ein Rahsegel befestigt. Zusätzlich ist es mit einem dreieckigen Vorsegel (engl. Jib) ausgestattet, dass an einem Seil zwischen dem vorderen Mast und der Bugspriet aufgehängt ist. Außerdem befindet sich deutlich mehr Ladung in Form von Kisten und Fässern auf dem Deck, um es attraktiver für Piraten zu machen. In der aktuellen Umsetzung sind alle Gegnerschiffe identisch und werden dargestellt wie es in der folgenden Abbildung (Abb. 9) zu sehen ist.



*Abb. 9: Außenansicht eines gegnerischen Schiffs (Quelle: Eigene Aufnahme)*

### 6.5 Charaktere

Damit die Spieler ihre gegenseitigen Positionen in der VR-Installation kennen und nicht kollidieren, wurden Charaktermodelle für die Spieler erstellt. Parallel dazu wurden auch Modelle erstellt die als Crew auf Gegnerschiffen platziert werden können. Grundlage für alle Charaktere bildet ein Low-Poly-Charaktermodell, dass mit Hilfe eines Tutorials des YouTube-

Kanals „CG Geek“ erstellt wurde (vgl. CG Geek, 2019). Das Modell wurde selbst in Blender modelliert und entspricht nicht genau dem Charakter, der im Video zu sehen ist, aber wurde mit der gleichen Technik erstellt. Um die verschiedenen Charaktere zu gestalten, wurde zunächst eine Kopie des Basismodells verwendet, die in ihrer Körperform angepasst wurde. Danach wurden zusätzliche Elemente wie Kleidung, Kopfbedeckungen und andere Gegenstände hinzugefügt, um den Charakteren verschiedenes Aussehen zu verleihen.

### 6.5.1 Spielercharaktere

Die Anwendung wurde für drei Spieler entwickelt, weshalb auch drei verschiedene Charaktermodelle für Spieler erstellt wurden. Da Spieler die Rollen einer Piratencrew übernehmen sind die Charaktere an stereotypische Darstellungen von Piraten wie man sie aus Medien kennt angelehnt.

Der erste Spielercharakter, der die Bezeichnung „Captain“ erhalten hat, soll den Kapitän der Piratencrew darstellen. Er hat einen fülligen Körper und buschigen Bart. Er trägt einen langen Mantel und einen großen Hut, der mit einer Feder geschmückt ist. Zusätzlich hat er ein Holzbein und es befindet sich ein Papagei auf seiner Schulter. Die Konzeptskizze und das finale Modell sind auf der folgenden Abbildung (Abb. 10) zusehen.



*Abb. 10: Konzeptskizze für "Captain" und fertiges Modell in Unity (Quelle: Eigene Aufnahme)*

Der zweite Spielercharakter mit der Bezeichnung „Sailor1“ stellt einen der beiden Matrosen der Spielercrew dar. Seine Körperform ist nah am ursprünglichen Charaktermodell und erinnert somit an einen schlanken Mann. Er trägt eine Weste ohne Hemd darunter und einen dreieckigen Hut. Um die Weste ist ein breites Band gebunden. An seiner Hüfte befindet sich

ein Säbel und eine seiner Hände wurde durch einen Haken ersetzt, wie man es auch in der folgenden Abbildung (Abb. 11) sieht.



Abb. 11: Konzeptskizze für "Sailor1" und fertiges Modell in Unity (Quelle: Eigene Aufnahme)

Der dritte und letzte Charakter trägt die Bezeichnung „Sailor2“ und hat eine eher weibliche Körperform. Sie trägt ein Hemd mit eingerollten Ärmeln, einen Gürtel mit Schnalle, sowie ein Kopftuch und eine Augenklappe. Die folgende Abbildung (Abb. 12) zeigt das fertige Modell.



Abb. 12: Konzeptskizze für "Sailor2" und fertiges Modell in Unity (Quelle: Eigene Aufnahme)

Alle Charaktere tragen lange Hosen und Stiefel.

### 6.5.2 Gegnercharaktere

Das Charaktermodell für gegnerische Crews ist Uniformen der britischen Royal Navy wie man sie zum Beispiel aus der bekannten Filmreihe „Fluch der Karibik“ kennt nachempfunden und auf der folgenden Abbildung (Abb. 13) sieht.



Abb. 13: Offiziersuniformen in Fluch der Karibik (Quelle: [https://fluch-der-karibik.fandom.com/wiki/Royal\\_Navy](https://fluch-der-karibik.fandom.com/wiki/Royal_Navy))

Die Kleidung der Gegner ist an diese Uniformen angelehnt, soll sie aber nicht perfekt imitieren. Zum einen wegen der Limitierung durch den Low-Poly-Style, aber auch weil die Gegnerschiffe fiktiv ohne Verbindung zu einem Staat oder einer Organisation sind, weshalb sich auch nur einfarbige Flaggen auf den Schiffen befinden. Da sich die Gegnercharaktere durch die gleichen Uniformen ohnehin sehr ähneln würden und Spieler sie meist nur aus der Entfernung sehen werden, wurde darauf verzichtet verschiedene Modelle für Gegner zu erstellen. Jedoch bekamen einzelne Gegnercharaktere verschiedene Haarfarben, um ein wenig Varianz zu erzeugen. Die Kleidung der gegnerischen Crews besteht aus einem langen blauen Mantel, der über einem weißen Hemd getragen wird. Anders als bei den Piraten sind die weißen Hemden in einem sauberen Weiß gehalten. Um den Mantel ist ein rotes Band gebunden. Der dreieckige Hut und die Stiefel sind schwarz. Die Hose ist grau. An der Hüfte tragen die Gegnercharaktere einen Säbel. Die folgende Abbildung (Abb. 14) zeigt eine Version des Gegnermodells mit braunen Haaren.



*Abb. 14: Modell für Crewmitglieder gegnerischer Schiffe in Unity (Quelle: Eigene Aufnahme)*

### 6.5.3 Anpassung der Charaktere an die VR-Anwendung

Für diese Anwendung wurde bewusst darauf verzichtet den Spielern wie in VR-Games normalerweise üblich Controller in die Hände zu geben. Grund dafür ist das Inputsystem in der VR-Installation, die für das Projekt entwickelt wurde. Die Spieler müssen in der Lage sein die verschiedenen Steuerungselemente zu greifen, um das Schiff zu kontrollieren. Würden sie Controller in den Händen halten wäre es nötig diese immer wieder abzulegen oder an den Armschlaufen hängen zu lassen. Da die Controller für die Steuerung des Spiels jedoch nicht nötig sind, kommen sie gar nicht zum Einsatz. Eine negative Folge daraus ist aber, dass nun die Positionen der Spielerhände in der Unity-Anwendung nicht bekannt sind. Die erstellten Charaktermodelle wurden mit Inverse-Kinematics-Rigs ausgestattet, die es erlauben, die Hände und Füße des Charakters in Position und Rotation zu verändern und basierend darauf die Gliedmaßen automatisch anpassen. In einer normalen VR-Anwendung könnte man hier in einem C#-Skript in Unity die Position und Rotation der Hände mit der Transformation der Controller gleichsetzen, um die Armbewegungen der Spieler in die virtuelle Welt zu übertragen. Da diese Informationen hier fehlen ist das nicht möglich. Die HTC Vive Pro ist in der Lage mit ihren beiden nach außen gerichteten Kameras Hände zu tracken, und dass sowohl in 2D als auch in 3D. Um diese Funktion zu nutzen, bietet HTC ein Plugin für Unity an (vgl. HTC Corporation [3], 2021). Damit wäre es möglich die Position und Rotation der Spielerhände zu tracken, ohne dass diese einen Controller in der Hand halten müssen. Diese Daten ließen sich dann auf das Rig übertragen, um die Position der Hände in der virtuellen Anwendung gleichzusetzen. Versuche dieses Plugin zu nutzen waren jedoch leider nicht erfolgreich. Selbst in einem neu aufgesetzten Projekt und Folgen der Anweisungen die HTC liefert, wurden Hände



nicht von der Anwendung erkannt. Da es keine Fehlermeldungen gab, und nur wenig Informationen zu dieser Funktion auffindbar waren, wurde darauf verzichtet viel Zeit in die Lösung dieses Problems zu investieren. Die Umsetzung der geplanten Anwendung hatte hier Priorität. Es blieb jedoch das Problem, das reale und virtuelle Hände nicht übereinstimmend dargestellt werden können.

Eine Lösung, die in VR-Anwendungen zum Einsatz kommt und die auch hier genutzt wurde, ist es bei Charaktermodellen vollständig auf Gliedmaßen zu verzichten. So sehen Spieler zwar ihre Arme nicht in der virtuellen Umgebung, aber dieser Umstand ist besser als eine Diskrepanz zwischen realer und virtueller Hand zu erzeugen. Die Charaktermodelle wurden also angepasst in dem die Arme entfernt wurden. Da bei den Beinen das gleiche Problem entsteht wurden diese auch entfernt. Hier könnte man zwar eine Laufanimation verwenden, diese wäre aber nicht synchron mit den tatsächlichen Bewegungen der Spieler. Auch aus ästhetischen Gründen war ein Verzicht auf alle Gliedmaßen besser als nur die Arme zu entfernen. Zusätzlich wurde bei den Charakteren noch der Hals entfernt. Die Modelle bestehen somit aus einem Torso, über dem der Kopf schwebt. Position und Rotation der Köpfe kann nun direkt aus den Translationsdaten des jeweiligen VR-Headsets entnommen werden. Der Torso wird mit Hilfe eines C#-Skripts unter dem Kopf platziert und in seiner vertikalen Rotation dem Kopf angepasst. Damit Spieler nicht das Innere ihres eigenen Charaktermodells sehen wurden bei den virtuellen Unity-Kameras die Culling-Masks angepasst, sodass für Spieler der eigene Kopf nicht gerendert wird. Eventuell wird es auch nötig sein den eigenen Torso auszublenden, sollte dieser von Spielern als störend empfunden werden. Für diese Entscheidung ist Rückmeldung von Dritten hilfreich, jedoch wurde die Anwendung zum Zeitpunkt dieser Arbeit noch nicht ausgiebig von Probanden getestet.

Wie man in der folgenden Abbildung (Abb. 15) sieht gehen durch die nötigen Anpassungen leider auch ursprünglich geplante Eigenschaften der Charaktere wie die Hakenhand und das Holzbein verloren.



*Abb. 15: Die drei Spielermodelle mit Anpassungen für die VR-Umgebung (Quelle: Eigene Aufnahme)*

Bei den Gegnermodellen wurden diese Anpassungen nicht vorgenommen. Da ihre Gliedmaßen nicht an reale Gegenstände angeglichen werden müssen entsteht hier auch in einer VR-Anwendung kein Problem. Es wäre eine Überlegung diese Änderungen dennoch an den Gegnern vorzunehmen, um eine stilistische Symmetrie beizubehalten. Da die Verwendung der ursprünglichen Charaktermodelle mit zusätzlicher technischer Ausstattung, wie zum Beispiel Ganzkörpertracking durch Vive Tracker in der Theorie möglich wäre, wurde davon abgesehen.

## 7 Spielwelt

Da die Grundlage der Spielwelt eine flache Wasseroberfläche bildet und die Spieler zusätzlich ein wenig erhöht auf dem Schiffsdeck stehen, haben sie eine sehr hohe Weitsicht in der Anwendung. Diese könnte zwar durch die Renderdistanz der virtuellen Kameras eingeschränkt werden, jedoch wäre dieser Effekt sehr auffällig. Aus diesem Grund und weil Spieler die Segelmechaniken auch über größere Distanzen nutzen sollen, wurde eine große Fläche für die Spielwelt gewählt. Die für die Wasseroberfläche erstellte *Plane* hat eine Kantenlänge von 2.000 Unity-Einheiten, was auch 2.000 Metern in der VR-Umgebung entspricht. In der aktuellen Umsetzung wurden neun dieser *Planes* in einer 3x3 Konstellation platziert, sodass eine Spielwelt mit einer Fläche von 36 km<sup>2</sup> entsteht. Von einer zufälligen Map-Generierung, um eine unendliche Spielwelt zu erzeugen wurde abgesehen. Da Spieler die Anwendung in den meisten Fällen nur einmalig testen werden können, war es sinnvoll eine abgegrenzte Spielfläche zu haben, die bewusst gestaltet werden konnte, um die Funktionen des Spiels vorzustellen.

### 7.1 Aufbau

Auch wenn es sich bei der Spielwelt um eine endliche Fläche handelt, wurden die verschiedenen Landmassen, die zur Gestaltung erstellt wurden, nicht alle von Hand platziert. Stattdessen wurde ein C#-Skript geschrieben, dass die als *Prefabs* gespeicherten Objekte nimmt und zufällig verteilt auf einem Grid erzeugt. Dafür wird in einer doppelten for-Schleife ein Grid durchlaufen, dass der Größe der Map entspricht und alle 100 Meter ein zufällig gewähltes Landobjekt platziert. Um das dadurch entstehende, unnatürlich wirkende Muster zu brechen, wird jedes Objekt um einen zufälligen Wert von bis zu 25 Metern in beide Richtungen der zur Wasseroberfläche parallelen Achsen verschoben. Zusätzlich wird jedes Objekt um einen zufälligen Wert zwischen 0 und 359° um die eigene vertikale Achse rotiert und zufällig leicht in seiner Skalierung verändert. Außerdem beinhaltet das C#-Skript einen Parameter, der genutzt wird, um zu bestimmen, wie hoch der Anteil an Grid-Positionen sein soll, an dem kein Landobjekt platziert wird, um zu verhindern, dass zu viele Landmassen entstehen.

Um Varianten der Spielwelt schnell zu erzeugen, wurde das C#-Skript mit einer *CustomEditor*-Klasse erweitert, die einen Button im Inspector-Fenster des C#-Skripts in Unity erzeugt, sodass

mit einem Klick die alten Landmassen gelöscht sind und neue wieder zufällig generiert werden. Ein zweiter Button erlaubt es außerdem alle Landmassen zu löschen, ohne neue zu erzeugen. Diese Funktionen werden außerhalb der Laufzeit des Spiels verwendet und verändern somit den bestehenden Szenenaufbau. Das ist wichtig, da alle Veränderungen an der Szene, die während der Laufzeit geschehen nicht gespeichert werden. Dieses C#-Skript sollte nicht dazu dienen ständig neue Varianten der Spielwelt zu erzeugen, sondern mit wenig Aufwand und Zeit eine Map zu erstellen die für das Projekt geeignet ist. Sobald das C#-Skript eine zufriedenstellende Karte erzeugt hatte, wurde es nicht weiterverwendet. Stattdessen wurden nun händisch kleinere Verbesserungen vorgenommen, wie zum Beispiel das Verschieben, löschen oder hinzufügen einzelner Landobjekte, bei denen die zufällige Generierung nicht ganz den Ansprüchen genügt hat. Das Festlegen auf einer Instanz der generierten Welten war außerdem wichtig, da zum einen dekorative Elemente wie Schiffswracks noch von Hand platziert wurden und die Routen der gegnerischen Schiffe so gewählt werden mussten, dass sie nicht durch Landmassen führen.

## 7.2 Rand der Welt

Durch das Verwenden einer endlichen Welt musste entschieden werden was passiert, wenn die Spieler mit ihrem Schiff den Rand der Welt erreichen. Eine Möglichkeit wäre es gewesen den gesamten Rand mit Land zu umschließen. Aufgrund der hohen Sichtweite müsste man aber noch hohe Berge einfügen, die den Blick blockieren, um zu verhindern das Spieler in die Leere sehen. Stattdessen wurde eine mit dem Unity-Partikelsystem erstellte Nebelwand am Rand der Wasserfläche entlang erzeugt. Diese dient als visuelle Blockade am Ende der Spielwelt und dient auch der nächsten Entscheidung, die für den Rand der Welt getroffen wurde. Um zu verhindern, dass das Schiff der Spieler einfach gegen eine unsichtbare Wand fährt, wurde ein C#-Skript erstellt, das registriert, ob das Spielerschiff einen Rand der Karte erreicht. Sobald dieser Fall eintritt, wird das Schiff auf die gegenüberliegende Seite der Spielwelt teleportiert, sodass es bei gleicher Fahrtrichtung wieder Richtung Mitte der Welt fährt. Da das Schiff, während der Teleportation auf beiden Seiten in der Nebelwand ist, und Spieler somit nicht den Rest der Karte sehen, passiert die Verschiebung der Schiffsposition unbemerkt.

### 7.3 Gestalterische Erweiterungen

Um die Spielwelt weiter zu füllen, wurden auf Basis des 3D-Modells für das Spielerschiff weitere Modelle erstellt, die Schiffswracks darstellen. Diese wurden von Hand auf einzelnen Landobjekten platziert. Ein Beispiel ist in der folgenden Abbildung (Abb. 16) zu sehen.



*Abb. 16: Variante eines Schiffswracks auf Insel platziert (Quelle: Eigene Aufnahme)*

Um die Spielwelt außerdem lebendiger wirken zu lassen wurde ein Schwarm Möwen modelliert und animiert, der sich auf einer zufälligen Route über die Map bewegt. Das hierfür erstellte C#-Skript sorgt dafür, dass der Schwarm auf einer Linie zwischen zwei zufälligen Punkten auf sich gegenüberliegenden Kanten der Spielwelt entlang fliegt. Zwei Parameter erlauben es die Flughöhe und Geschwindigkeit des Schwarms anzupassen. Erreicht der Schwarm den Rand der Karte wird eine neue zufällige Route berechnet.

## 8 VR-Rauminstallation

### 8.1 Verfügbarer Raum für die Installation

Zentrales Ziel dieses Projekts war es eine raumfüllende VR-Installation zu bauen, die das Deck eines Piratenschiffs imitiert, und Spielern erlaubt mit anfassbaren Input-Elementen das Schiff zu bedienen. Der dafür verfügbare Raum, in den die Installation integriert wurde, hat eine Länge von ungefähr 6,10 Metern und eine Breite von 4,10 Metern. Eine 2,00 \* 1,50 Meter große Ecke des Raumes kann nicht für die Installation verwendet werden, da sich dort ein Tisch mit Computer hinter einer Trennwand befindet. In zwei sich diagonal gegenüberliegenden Ecken sind Lighthouses der ersten Generation montiert, die Outside-In-Tracking mit SteamVR erlauben. Die Wände des Raums sind mit Holzplatten verkleidet an denen Objekte montiert werden können. Auf dem Boden ist ein Teppich verlegt. Entlang der Decke sind außerdem mehrere Holzlatten verschraubt, die auch zur Montage genutzt werden können (vgl. Anhang „Raumplan“)

### 8.2 Steuerungselemente

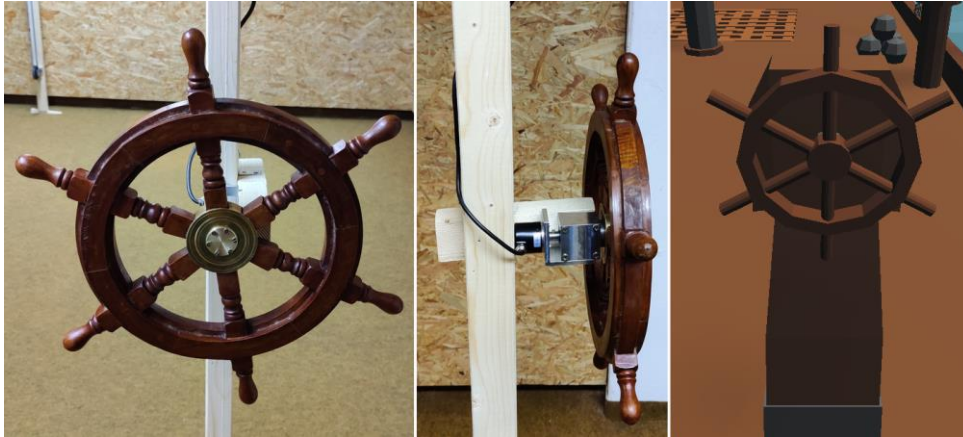
Für die VR-Installation wurden fünf Stationen entworfen, die es Spielern erlauben mit dem virtuellen Schiff zu interagieren. Ein Steuerrad kann genutzt werden das Schiff zu Steuern. Mit der Schot kann das hintere Segel bedient werden und mit dem Fall lässt sich die Größe der vorderen Segelfläche anpassen. Auf Backbord und Steuerbord steht jeweils eine Kanone, mit der gegnerische Schiffe versenkt werden können.

Der genaue Aufbau der einzelnen Elemente wird in den folgenden Unterkapiteln genauer beschrieben. Allgemein war ursprünglich geplant jede Input-Station an einem Gestell, das auf einem möglichst kleinen Fuß steht zu befestigen. Kleine Standfüße hätten jedoch zur Folge, dass die Steuerelemente leicht umkippen können. Die Position der Stationen sollte sich jedoch nicht verändern, um eine Übereinstimmung mit den virtuellen Gegenständen zu gewährleisten. Größere Standplatten würden diesem Problem zwar entgegenwirken, müssten aber von Spielern betreten werden, wobei Stolperfallen entstehen. Ein weiteres Problem von auf Standfüßen platzierten Elementen war zusätzlich, dass nicht klar war, wo die Kabel für die verwendete Sensorik zum Computer entlanggeführt werden soll.

Die endgültige Lösung war es, auf Standfüße ganz zu verzichten und stattdessen die vorhandenen Deckenlatten zu nutzen. Die Gestelle wurden somit raumhoch geplant und zwischen Boden und Decke eingespannt. Für mehr Stabilität wurden sie oben in die Deckenlatten geschraubt. Auf diese Weise wird sichergestellt, dass die Steuerungselemente auch bei Interaktion ihre Position beibehalten und der Boden frei bleibt, um Stolpern zu verhindern. In den Entwurfszeichnungen wurden vierkantige Metallstangen genutzt, die in ihrer Länge mit einer Feststellschraube angepasst werden können. Das würde erlauben die Installation mit geringem Aufwand auf- und -abzubauen und die Elemente platzsparend zu verstauen. Aus Zeit- und Kostengründen wurden stattdessen jedoch Holzlatten verwendet, die oben mit den Deckenlatten verschraubt und unten mit Holzkeilen eingespannt wurden. Dadurch verringerte sich zwar die Mobilität der Installation, die Holzlatten machten es jedoch einfacher die Input-Elemente zu montieren. Außerdem war es jetzt möglich die Mikrocontroller-Boards in Deckennähe zu montieren und die Kabel entlang der Decke zu führen.

### 8.2.1 Steuerrad

Als Steuerrad wurde ein richtiges Schiffssteuerrad aus Holz mit einem Durchmesser von 47 cm verwendet. Es wurde auf einer Welle aufgehängt, die in einer Halterung aus Aluminium mit der Welle eines Drehgebers verbunden wurde, sodass die Drehung des Steuerrads auf den Sensor übertragen wird. Diese wurde an die vertikale Holzlatte montiert, wobei eine weitere kleine Holzlatte im rechten Winkel genutzt werden musste, um den Abstand des Steuerrads zum Gestell zu erhöhen. Grund hierfür war zum einen um Reibung mit dem Gestell zu verhindern, weil bei Fertigung der Halterung nicht bedacht wurde, dass das Steuerrad Abstand braucht. Außerdem sollte es möglich sein die einzelnen Griffe des Steuerrads mit der ganzen Hand zu umgreifen. Die Konzeptskizze für das Steuerrad ist im Anhang zu finden. Die fertige Umsetzung des realen Steuerrads, sowie das virtuelle Gegenstück sind in der folgenden Abbildung (Abb. 17) zu sehen.



*Abb. 17: Steuerungselement "Steuerrad" in der Installation und als virtuelles Gegenstück in der Unity-Anwendung (Quelle: Eigene Aufnahme)*

Bei der Modellierung des virtuellen Steuerrads war es wichtig, dass es die gleiche Größe wie das echte Gegenstück hat. Das heißt der Durchmesser des Holzrings und die Länge der abstehenden Griffe musste in der Unity-Anwendung mit der Realität übereinstimmen. Außerdem musste die Anzahl der Griffe und der Winkel zwischen den Griffen passen, sodass Spieler die einzelnen realen Griffe zielsicher greifen können, obwohl sie nur das virtuelle Steuerrad in der VR-Brille sehen.

Das Steuerrad sollte ursprünglich im hinteren Bereich des Decks mittig platziert werden. Durch den PC-Raum in der Ecke des Raumes war das jedoch nicht möglich, weshalb es weiter auf Steuerbordseite aufgebaut wurde. Die finale Position des Steuerrads lässt sich aus dem Raumplan im Anhang entnehmen.

### 8.2.2 Schot

Die Schot ist ein Seilzugsystem zwischen Rumpf und Baum, dass sich in seiner Länge verstellen lässt, um zu bestimmen, wie weit ein Segel vom Wind nach Lee bewegt werden kann. Einige verworfene Entwürfe für die Umsetzung dieses Steuerungselements sahen vor, dass Spieler eine Leine spannen und lockern können, die auf eine Rolle gewickelt ist. Das Ende der Leine könnte man in einer Klemme befestigen, damit man sie loslassen kann, ohne dass sich die Länge der Schot ändert. Ein Elektromotor an der Rolle würde die Kraft des Windes imitieren, sodass die Schot bei Druck auf dem Segel angezogen wird. Dieses Konzept hat jedoch mehrere Schwächen. Zum einen wäre dies eine komplexe mechanische Vorrichtung und dadurch fehleranfällig. Der Motor müsste verschiedene Zustände imitieren können. Ist Druck auf dem



Segel muss die Schot gespannt werden. Ist die Schot zu locker, flattert das Segel im Wind und die Leine ist nicht gespannt. Ziehen Spieler an der Schot muss sie kürzer werden und gegebenenfalls wieder gespannt werden sobald Druck auf dem Segel ist. Ein zusätzliches Problem wäre, dass es nicht möglich ist, den anfassbaren, losen Teil des Seils in die virtuelle Welt zu übertragen, sodass Spieler nicht wüssten, wohin sie greifen müssen.

Stattdessen wird die Schot als Kurbel dargestellt, die man drehen kann, um die Schot zu lockern oder dichter zu ziehen, in dem die Leine auf einer Rolle auf- und abgewickelt wird. Als anfassbares Objekt in der VR-Installation wurde hier eine Drehkurbel verwendet die eigentlich bei Maschinen zum Einsatz kommt. Der Griff ist frei drehbar, sodass sich die Kurbel angenehm bedienen lässt. Sie ist ähnlich zum Steuerrad auf eine Welle montiert und in einer Halterung mit einem Drehgeber verbunden. Die Kurbel besteht aus Kunststoff, weshalb sich das Material nicht sehr passend zum Piratenschiff anfühlt. Alternativen aus Holz oder Metall die für den Installationsaufbau geeignet sind waren jedoch nur schwer beschaffbar. Die Konzeptskizze für die Schot ist im Anhang zu finden. Die Schotkurbel als anfassbarer Teil der Installation und als Modell in der Virtualität ist in der folgenden Abbildung (Abb. 18) zu sehen.



*Abb. 18: Steuerungselement "Schot" in der Installation und als virtuelles Gegenstück in der Unity-Anwendung (Quelle: Eigene Aufnahme)*

Die Kurbel hat einen Durchmesser von 20 cm, der auch für das virtuelle Gegenstück übernommen wurde. Auch der Griff hat in der virtuellen Welt die gleiche Größe, sodass die Rotation der echten Kurbel direkt übertragen werden kann.

Die Schot sollte ursprünglich in der Mitte des Decks platziert werden, ein Stück vor dem Steuerrad. Da jedoch keine der Deckenlatten direkt in der Mitte des Raums entlangführt, da

sich dort Lampen befinden, musste die Schot um ein kleines Stück nach Backbord verschoben werden. Die finale Position der Schot lässt sich aus dem Raumplan im Anhang entnehmen.

### 8.2.3 Fall

Auch wenn sich bei Schot dagegen entschieden wurde, war es dennoch ein Ziel, dass die Spieler in der Installation mit einem echten Seil interagieren können, um das Gefühl, dass sie ein Schiff steuern zu verstärken. Das Fall, das Spieler nutzen können, um die Segelfläche des vorderen Segels in ihrer Größe zu verändern bot hierbei eine Möglichkeit, die genutzt wurde. Anders als bei der Schot war es beim Fall nicht notwendig, dass das Seil je nach Situation verschieden stark gespannt ist. Stattdessen wurde ein Konzept entwickelt bei der ein Seil an seinen beiden Enden verbunden wurde und auf zwei Seilrollen, die eigentlich für Trainingsgeräte genutzt werden, aufgehängt wird. Die beiden Rollen wurden mit Halterungen an einer vertikalen Holzlatte montiert, wobei sich eine in Boden- und die andere in Deckennähe befindet. Das Resultat ist eine senkrecht hängende Leine, die Spieler greifen und unendlich weit in beide Richtungen ziehen können. Dabei soll ein Herunterziehen des Seils das Segel hochziehen und somit die Segelfläche verkleinern. Die andere Richtung lässt das Segel herunter und vergrößert somit die Segelfläche. An der oberen Rolle ist eine Welle befestigt, die mit Hilfe einer Halterung mit einem Drehgeber verbunden ist. Das Ziehen am Seil resultiert in einer Rotation der Rollen, die über den Drehgeber an ein Mikrocontroller-Board weitergegeben wird. Die Konzeptskizze für das Fall ist im Anhang zu finden. In der folgenden Abbildung (Abb. 19) sind mehrere Abbildungen des Falls in der Installation zu sehen, zusammen mit einem Bild des virtuellen Falls.

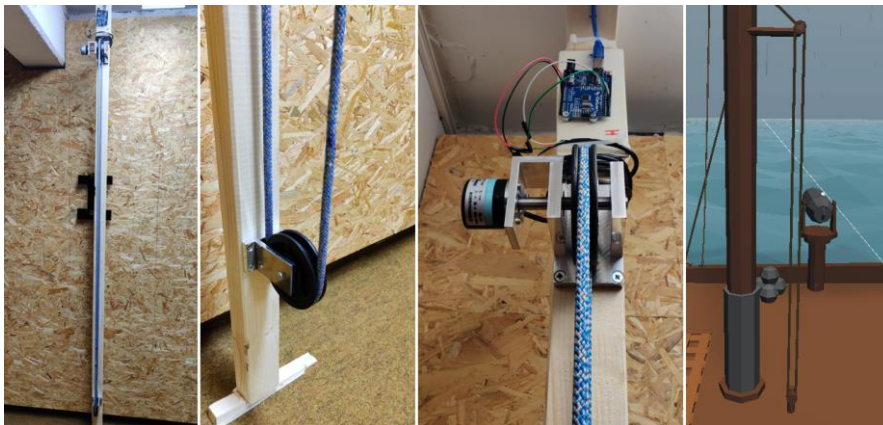


Abb. 19: Steuerungselement "Fall" in der Installation und als virtuelles Gegenstück in der Unity-Anwendung (Quelle: Eigene Aufnahme)

Das Modell für das Fall ist in seinem Aussehen ähnlich zum realen Gegenstück, wobei der Abstand der beiden Rollen nicht der gleiche ist. Da die Spieler das Seil vor sich greifen werden und mit den Rollen eigentlich nie in Berührung kommen werden, hat dies keinen Einfluss auf das Spiel. Die Position des Falls sollte sich ursprünglich am vorderen Mast befinden, an dem in der Virtualität ein Seilzugsystem weiter nach oben zum Rah des Segelns führt. Da sich in der umgesetzten Installation der vordere Mast außerhalb des realen Raums befindet, ist es Spielern nicht möglich dort hinzulaufen. Stattdessen befindet sich das Fall nun am hinteren Mast, der im realen Raum an der vorderen Wand steht. Genau wie die Schot konnte es nicht mittig platziert werden. Da das Fall dennoch das vordere Segel bedienen soll, führt in der virtuellen Welt das Seil über Umlenkrollen zum vorderen Mast.

#### 8.2.4 Kanonen

Die Kanonen der Spieler sollten zwei Funktionen haben. Zum einen sollten Spieler zielen können, in dem die Halterung der Kanone um zwei Achsen rotierbar ist. Als zweite Funktion sollte es möglich sein die Kanone zu feuern, um einen Schuss abzusetzen. Das Konzept der Kanonen für die Installation sieht eine kleine Holzbox vor an der ein Griff befestigt ist, den Spieler greifen können, um das Kanonenrohr auszurichten. Die Holzbox hat eine Kantenlänge von 20 cm auf jeder Seite und ist somit deutlich kürzer als ihr virtuelles Gegenstück, da das virtuelle Kanonenrohr über den Rand des Schiffs raus ragt und damit die realen Wände durchdringt. In der Konzeptzeichnung, die im Anhang zu finden ist, wird eine Halterung vorgesehen, die sich um zwei Achsen rotieren lässt (Pan und Tilt) und an einer vertikalen Holzlatte montiert werden sollte. Letztendlich wurde jedoch eine andere Lösung für die Befestigung gefunden, wie in der folgenden Abbildung (Abb. 20) zu sehen ist. Genutzt wird eine Monitorhalterung, die zwischen Holzbox und Wand montiert wurde. Die Halterung bietet genügen Rotationsfreiraum um die Kiste um die vertikale und horizontale zu drehen. Durch Festziehen oder Lockern von Muttern lässt sich der Widerstand der Bewegung verändern. Die Monitorhalterung ist in der Lage das Gewicht der „Kanone“ zu halten, und trotzdem die Rotationen zuzulassen. Um zu verhindern, dass die Holzbox die Bewegungsfreiheit der Monitorhalterung einschränkt, wurden sie mit einem Stück Holz als Abstandshalter montiert. Durch die Anbringung der Kanonen an den Wänden konnten zwei der geplanten vertikalen Gestelle im Raum wieder entfernt werden. Die Positionen der Kanonen, die im Raumplan im Anhang angegeben sind, sehen noch die Montage an den Holzplatten vor. In der folgenden

Abbildung (Abb. 20) sieht man die fertige Umsetzung einer Kanone mit dem Vive Tracker, sowie das virtuelle Gegenstück.



*Abb. 20: Steuerungselement "Kanone" in der Installation und als virtuelles Gegenstück in der Unity-Anwendung (Quelle: Eigene Aufnahme)*

Um die Ausrichtung des Kanonenrohrs auf sein virtuelles Gegenstück zu übertragen, werden jeweils ein Vive Tracker 3.0 verwendet. Die Tracker sind mit Stativschrauben auf der oberen Seite der Holzbox montiert und werden wie das VR-Headset mit Hilfe der Lighthouses getrackt. Sie können ihre Position und Rotation im Raum an die Unity-Anwendung weitergeben. Für die Kanonen werden nur die Rotationsdaten benötigt. Das Modell für die Kanonen besteht aus drei Hauptteilen. Die Halterung auf dem Standfuss wird genutzt, um die Kanone in der horizontalen zu rotieren (Pan) und das Kanonenrohr selbst wird für die vertikale Rotation (Tilt) verwendet. Am Kanonenrohr ist ein Griff, der das reale Gegenstück darstellt. So können Spieler diesen in die Hand nehmen, um das Kanonenrohr auszurichten.

Um die Kanone zu zünden, wird ein Luntentab verwendet. In der Realität handelt es sich dabei um einen gekürzten Besenstiel mit 90 cm Länge an dem ein weiterer Vive Tracker mit einer Stativschraube befestigt wurde. In der virtuellen Welt sieht er aus wie in der folgenden Abbildung (Abb. 21) zu sehen, mit dem Zusatz, dass durch Partikel die Lunte brennend, beziehungsweise glühend dargestellt wird.



*Abb. 21: Modell des Luntentabs in der Unity-Anwendung (Quelle: Eigene Aufnahme)*

Um eine Kanone zu zünden, müssen die Spieler den Luntentab in die Hand nehmen und die glühende Lunte an die Lunte der Kanone halten. Die Position und Rotation des virtuellen Luntentabs wird mit Hilfe des Vive Trackers an seinem realen Gegenstück ermittelt. In der Unity-Anwendung befinden sich *Collider* am Luntentab und den Lunten der Kanonen. Treffen diese aufeinander wird die Lunte der Kanone gezündet. Dies wird durch funkenartige Partikel visualisiert, die sich an der Lunte entlang zur Kanone bewegen. Nach kurzer Zeit feuert die Kanone eine Kugel und kann nach wenigen Sekunden Cooldown wieder gezündet werden.

Auf eine Nachlademechanik wurde verzichtet. Eine rein virtuelle Lösung war durch das Weglassen von VR-Controllern nicht möglich. Konzepte für Umsetzungen als Erweiterung der Installation sollten es möglich machen eine anfassbare Kugel in die Kanone zu laden. Über eine Rampe oder Ähnliches soll die Kanonenkugel wieder aus dem Kanonenrohr zurück an die Stelle gerollt werden, an der der Spieler sie aufgehoben hat. Dadurch hätten Spieler einen „unendlichen“ Vorrat an Kanonenkugel den sie verwenden können. Da alle erdachten Konzepte nicht zielführend oder zufriedenstellend waren wurde ganz darauf verzichtet.

## 9 Mikrocontroller-Input

### 9.1 Mikrocontroller-Hardware

Drei der Steuerungselemente in der VR-Installation (Steuerrad, Schot und Fall) nutzen Drehgeber in Kombination mit Mikrocontroller-Boards, um den Input der Spieler in Form von Rotationsdaten an das Spiel weiterzugeben. Die Drehgeber sind wie schon beschrieben über Wellen mit beweglichen Teilen der jeweiligen Station verbunden, um die Drehbewegungen zu registrieren. An den vertikalen Holzplatten, an denen die Steuerungselemente angebracht wurden, befindet sich nahe der Raumhöhe jeweils ein Mikrocontroller-Board. Die genutzten Drehgeber haben jeweils vier durch Farbe der Isolierung markierte Kabel, die an die Mikrocontroller-Boards angeschlossen werden mussten. Da es sich hierbei um offene Kupferdrähte handelte, konnten diese nicht direkt mit dem Board verbunden werden. Mit den Steckplätzen auf dem Board kompatible Kabel wurden benutzt. Diese haben einen weiblichen und männlichen Steckplatz auf jeder Seite. Um diese Kabel mit den Sensoren zu verbinden, wurden sie auf der Seite mit den weiblichen Steckplätzen durchtrennt und es wurde ein Stück der Isolation entfernt. An dieser Stelle konnte das Kabel dann mit einem Sensorkabel verlötet werden. Der männliche Stecker am Ende konnte jetzt für die Steckplätze auf dem Mikrocontroller-Board genutzt werden. Um Kurzschlüsse zu verhindern, wurden die Lötstellen mit Isolierband umwickelt.

Zwei der Kabel des Drehgebers (rot und schwarz) dienen der Stromversorgung (5 Volt) und Erdung und werden mit den entsprechend beschrifteten Steckplätzen auf dem Mikrocontroller-Board verbunden. Das weiße und grüne Kabel wird für die zwei Phasen des Drehgebers genutzt, mit den wie schon beschrieben die Rotation in beide Richtungen ermittelt werden kann. Diese Kabel müssen mit sogenannten „interrupt pins“ auf dem Board verbunden werden. Die Steckplätze mit den Beschriftungen „2“ und „3“ wurden hierfür genutzt. Um die Daten der Mikrocontroller-Boards an den Computer, auf dem die Unity-Anwendung läuft zu senden, sind diese mit USB-Kabel verbunden. (vgl. Electric Diy Lab, 2020)

In der folgenden Abbildung (Abb. 22) ist die Verkabelung des Falls zu sehen. Die vier Kabel des Drehgebers sind wie beschrieben mit dem Mikrocontroller-Board verbunden. Das blaue USB-Kabel, das oben zu sehen ist, führt von dem Board zum Computer, auf dem die Unity-Anwendung läuft.

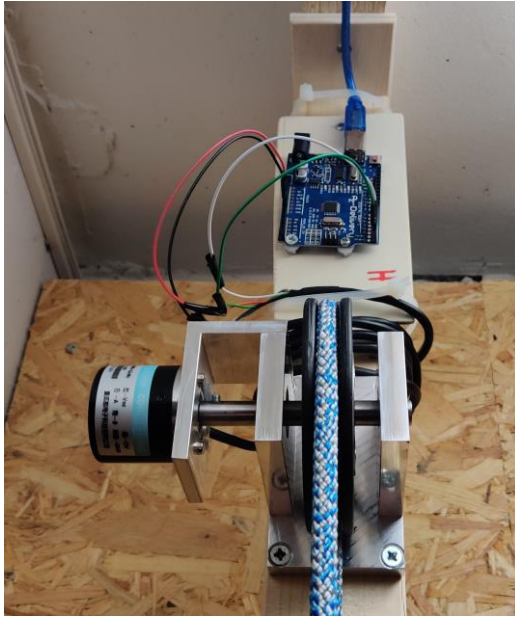


Abb. 22: Verkabelung eines Drehgebers mit einem Mikrocontroller-Board am Beispiel des Steuerungselements "Fall" (Quelle: Eigene Aufnahme)

## 9.2 Mikrocontroller-Software

### 9.2.1 Arduino-Code

Die Entwicklungsumgebung „Arduino IDE“ wurde genutzt, um ein C++-Skript zu schreiben, dass die Daten der Drehgeber auf dem Mikrocontroller-Board ausliest und in Form von verwertbaren Variablen an den PC weitergibt. Der Code basiert weitgehend auf dem Tutorial von Electric Diy Lab (vgl. Electric Diy Lab, 2020). Hier wird die Phasenverschiebung genutzt, um die Drehrichtung des Sensors zu bestimmen. Eine Zählvariable, die bei Start der Anwendung den Wert 0 hat, wird bei Rotation der Input-Elemente je nach Drehrichtung erhöht oder verringert. Eine volle Umdrehung bedeutet hierbei eine Veränderung um 1.200. Diese Zählvariable wird als *String* ausgegeben und kann in Unity abgefangen werden. Der Code ist auf allen drei Mikrocontroller identisch, bis auf einen Buchstaben der dem ausgegeben *String* mit dem Rotationswert vorangesetzt wird, um zwischen Steuerrad (W), Schot (S) und Fall (H) zu unterscheiden. Der vollständige Arduino-Code befindet sich im Anhang.

### 9.2.2 Unity-Code

Um die Daten der Mikrocontroller-Boards in der Unity-Anwendung zu verwenden, mussten diese über die COM-Port-Schnittstelle abgefangen werden. In einem C#-Skript werden mit der Klasse „SerialPort“ alle angeschlossenen Ports geöffnet. Anschließend werden die

einkommenden Daten jedes Ports überprüft. Die Mikrocontroller senden *Strings* die mit den Buchstaben W, S oder H beginnen, gefolgt mit einer Zahl für die Rotation. Die einzelnen Ports können anhand ihres Namens unterschieden werden („COM“ + Nummer, z.B. „COM1“). Sollte der ausgelesene Wert eines Ports mit einem der drei Buchstaben beginnen, kann zugeordnet werden, dass es sich hierbei um einen Port handelt, an dem eins der Steuerungselemente angeschlossen ist. Diese Überprüfung der Port wird so lange wiederholt, bis alle drei Input-Elemente einem COM-Port zugeordnet wurden. In der *Update*-Methode des C#-Skripts können die Steuerungselemente nun ausgelesen werden. Der erste *Char* des empfangenen *Strings* eines COM-Ports wird genutzt, um zu unterscheiden, ob es sich um den Wert für Steuerrad (W), Schot (S) oder Fall (H) handelt. Der Rest des *Strings* wird in einen *Float*-Wert umgewandelt und beschreibt die Rotation des Drehgebers seit Beginn der Anwendung. Da sich alle Steuerungselemente unendlich weit in beide Richtungen drehen lassen, ist die tatsächliche Rotation weniger interessant für die Anwendung. Stattdessen überprüft das C#-Skript ob und welche Richtung gedreht wird, um basierend darauf Variablen zu verändern, die Länge von Schot und Fall, sowie Winkel des Ruders beschreiben (genauer im folgenden Kapitel). Für die Schot wurde die Verarbeitung des Inputs abgeändert, da die reale Kurbel bei Loslassen ihre Drehung nicht beibehält und der Griff durch sein Gewicht immer nach unten fällt. Aufgrund der Bauweise der Halterung für die Kurbel war es nicht schnell möglich etwas einzubauen das Reibung erzeugt, um die Kurbel zu stabilisieren. Würde man nun nur beachten, ob sich die Kurbel gerade dreht, würde das Loslassen der Kurbel eventuell weitere Signale senden, obwohl der Spieler schon die gewünschte Schotlänge eingestellt hatte. Um dieses Problem zu verhindern, wird immer eine volle Umdrehung als Schot-Input verwendet. Durch Berechnen der vollen Umdrehungen der Kurbel kann bei Veränderung überprüft werden, ob eine weitere volle Umdrehung betätigt wurde und in welche Richtung diese ging. Tritt dieser Fall ein wird dementsprechend die Variable für die Länge der Schot um den eingestellten Wert erhöht oder verringert.

Um bei Steuerrad und Schot-Kurbel die Rotation des virtuellen Gegenstücks mit dem realen Objekt gleichzusetzen wird der absolute Rotationswert, der zuletzt empfangen wurde, genutzt. Da die Drehgeber eine Auflösung von 1.200 pro volle Umdrehung liefern wird der empfangene Wert durch 1.200 geteilt und anschließend mit 360 multipliziert, um die tatsächliche Gradzahl zu erhalten. Damit die Gegenstücke übereinstimmen ist es wichtig, dass



bei Start der Anwendung die realen Elemente in der richtigen Ausgangstellung sind, da diese den Rotationswert 0 erhält. Da das C#-Skript nur die Drehrichtung für Inputs verwertet, erhält jedes Steuerungselement einen Faktor in Form einer *Float*-Variablen, mit dem die Veränderungsgeschwindigkeit des Inputs angepasst werden kann.

## 10 Segelmechanik

Um das Segelsystem des Spielerschiffs möglichst zentral zu verwalten, wurde ein C#-Skript „SailsManager“ erstellt. Dort zu finden sind Parameter die Eigenschaften des Windes beschreiben und Referenzen zu Bestandteilen des Schiffs, die für die Segelmechanik relevant sind. Außerdem wird hier die Geschwindigkeit des Schiffs berechnet und auf das Schiff angewandt.

### 10.1 Wind

Der Wind in diesem Projekt wird durch zwei Variablen beschrieben. Ein *Vector2* wird verwendet, um die Richtung des Windes festzulegen. Diese ist immer parallel zur Wasserebene und wird aktuell vor dem Start der Anwendung, beziehungsweise schon im Build festgelegt und nicht weiter verändert. Bei Weiterentwicklung wäre es denkbar, dass sich die Windrichtung im Laufe der Spielzeit verändert, sodass Spieler darauf reagieren müssten. Eine zweite Eigenschaft des Windes ist die Windstärke. Sie wird durch eine *Float*-Variable beschrieben und bildet die Stellschraube, mit der die maximale Geschwindigkeit der Schiffe angepasst werden kann. Sowohl das Spielerschiff als auch die Gegnerschiffe verwenden in ihrer Geschwindigkeitsberechnung diese Windstärke, sodass die Verhältnisse gleichbleiben, sollte man den Parameter verändern. Auch diese Variable bleibt aktuell unverändert und wurde auf einen Wert festgelegt, der ein angenehmes Geschwindigkeitsgefühl vermittelt. In Weiterentwicklungen des Projekts könnte sich die Windstärke auch im Laufe der Zeit verändern. Denkbar wäre es auch wie in der Realität die Spielwelt in Bereiche zu teilen in den verschiedenen Windstärken herrschen. Flecken, in denen mehr Wind weht, sind auf dem Wasser aus der Ferne schon durch eine krausere Wasseroberfläche leicht erkennbar. Um dies zu visualisieren, müsste jedoch der Wasser-Shader weiter angepasst werden.

Damit Spieler die Segel passend zu Windrichtung einstellen können ist es wichtig ihnen diese Richtung zu kommunizieren. Da sie in der Installation den virtuellen Wind nicht spüren können brauchen sie eine visuelle Unterstützung. Auf Segelschiffen wird hierzu eine Hilfsvorrichtung namens Verklicker verwendet. Dabei kann es sich um ein kleines Fähnchen oder auch einem pfeilförmigen Anzeiger, der an der Spitze des Masts angebracht wird, handeln. Der Verklicker dreht sich in den Wind und Segler haben somit immer die Möglichkeit nach oben zu schauen, um die Windrichtung zu ermitteln. Im Modell des Spielerschiffs befindet sich auf der Spitze

des hinteren Masts ein pfeilförmiger Verklicker, der Spielern die tatsächliche Windrichtung anzeigt. Auf die Beachtung des „scheinbaren Winds“ wird wie schon beschrieben verzichtet. Der Verklicker hat ein rotes Material erhalten, um gut sichtbar zu sein. Wichtig zu erwähnen ist noch, dass die Pfeilspitze des Verklickers durch seine Bauweise nach Luv zeigt, also in die Richtung, aus der der Wind herkommt. Im Spielcode wird der Vektor der Windrichtung in eine Gradzahl umgerechnet und für die globale Rotation des Verklickers um seine vertikale Achse genutzt. Als weitere Visualisierung des Winds wurden noch Partikel erstellt, die mit einem Tutorial des YouTube-Kanals „Hennejoe“ erstellt wurden und dem Windeffekt des Spiels „Legend of Zelda: The Wind Waker“ nachempfunden sind (vgl. Hennejoe, 2020). Diese Partikel wurden um das Spielerschiff verteilt und sind in der Szenenhierarchie dem Verklicker untergeordnet, sodass sie auch die Windrichtung visualisieren.

## 10.2 Schratsegel

Das Schratsegel ist am hinteren Mast befestigt und wird mit der Schot bedient. Rotationspunkt für das Segel ist ein Ring am Baum des Segels, dem der Baum und die restlichen Bestandteile des Segels untergeordnet sind. Dieses Objekt hat in Unity die Komponenten *Rigidbody* und *HingeJoint* erhalten, um auf den Wind reagieren zu können. Zusätzlich wurde ein *Collider* erstellt der grob die Fläche des Segels umfasst. Der *HingeJoint* wird als eine Art Scharnier zwischen zwei *Rigidbody*s verwendet, im Normalfall, um zum Beispiel eine Tür umzusetzen. In diesem Fall sorgt diese Komponenten dafür, dass sich das Segel bei Krafteinwirkung nur um den gewählten Ankerpunkt dreht, ohne seine lokale Position zu verändern. Außerdem erlaubt es die *HingeJoint*-Komponente einen maximalen Winkel einzustellen um den sich der *Rigidbody* rotieren kann. Dies wird für die Schot noch relevant. Um das hintere Segel auf den Wind reagieren zu lassen wird zunächst ein *Vector3* berechnet der die Windkraft beschreibt und aus der Windrichtung sowie Windstärke gebildet wird. Diese Variable wird als Kraft auf den *Rigidbody* des Segels angewendet. Auf diese Weise wird das Segel immer nach Lee gedrückt, bis es im Wind steht.

Um Antrieb für das Schiff zu erzeugen, muss das Schratsegel mit Hilfe der Schot limitiert werden. Diese Limitierung durch die Schot wird durch die *Limits*-Parameter der *HingeJoint*-Komponente umgesetzt. Da es sich hierbei um einen maximalen Winkel handelt sind die Grenzwerte, auf die die Länge der Schot limitiert wird, gleichzeitig die Maximalwinkel, die das

Schratsegel nach außen schwingen soll. Eine maximal lange Schot hat den *Float*-Wert 80, was dafür sorgt, dass sich das Segel um maximal 80° weg von der Schiffsmitte dreht. Je nach Stellung des Schiffs zur Windrichtung kann der tatsächliche Winkel des Segels jedoch auch kleiner sein, sollte es schon früher im Wind stehen. Als Minimalwert für die Schotlänge wurde 1 gewählt, um das Segel immer noch ein bisschen wackeln zu lassen, auch wenn die Schot ganz angezogen ist. Je nachdem ob sich das Segel auf Backbord oder Steuerbord befindet, wird die Richtung des Bauchs im Segeltuch angepasst. Außerdem verändert sich die Tiefe des Bauchs ein wenig mit der Segelstellung. Die Länge der Schot wird wie schon beschrieben durch Drehen der Kurbel verändert.

Damit Segelunerfahrene Spieler weiter bei der richtigen Einstellung des Schratsegels unterstützt werden wurde das Segel noch um Windfäden erweitert. Dabei handelt es sich um, in diesem Fall rote, Fäden, die im Segel an einem Ende befestigt werden. Je nach Richtung, in die die Fäden wehen, können Segler erkennen, ob die Schot zu kurz, zu weit oder ideal eingestellt ist. Im Spiel ist auf jeder Seite des Segels ein Faden befestigt, der durch einen *LineRenderer* dargestellt wird und mit Hilfe eines C#-Skripts flattert. Normalerweise können Segler durch die Transparenz des Segels beide Fäden erkennen und benutzen die Richtung beider, um Informationen über ihre Segelstellung zu ermitteln. Da das virtuelle Segel jedoch nicht transparent ist und das Konzept etwas komplex für Laien sein könnte wurde eine vereinfachte Interpretation von Windfäden erstellt. Beide Windfäden flattern hier immer in die gleiche Richtung. Sind die Fäden Parallel zum Baum, ist die Schot ideal im Verhältnis zu Windrichtung und Fahrtrichtung des Schiffs eingestellt. Flattert der Faden nach oben ist die Schot zu dicht, nach unten ist die Schot zu locker. Eine Visualisierung des Windfadens ist im Anhang „Poster für Vermittlung der Segelgrundlagen“ zu finden.

### 10.3 Rahsegel

Das Rahsegel lässt sich in der aktuellen VR-Installation nur mit dem Fall bedienen. Die Stellung des Segels zum Wind passiert deshalb automatisch in Abhängigkeit zum Winkel des hinteren Segels. Um zu verhindern, dass das vordere Segel andere Teile des Schiffs durchdringt, wird die Stellung des Segels im Code immer so angepasst, dass es sich auf der vorderen Seite des Masts dreht. Für eine passende Stellung der beiden Segel muss der Winkel des Rahsegels größer werden, wenn sich der Winkel des hinteren Segels verringert. Bei einer Wende

wechseln die Segel die Seite des Schiffs, wobei der Winkel des hinteren Segels sehr klein wird, während das vordere Segel einen großen Winkel hat. Deshalb muss das vordere Segel bei der Wende um einen großen Winkel rotiert werden damit es auf der richtigen Seite steht und wieder die passende Stellung zum Wind hat. Damit es nicht in einem Frame schlagartig die Seite wechselt, wird der Winkel im C#-Skript langsam verändert bis die Segelstellung stimmt, um eine fließende Bewegung des Segels zu erzeugen.

Das Fall, das in der Installation mit der Seilschlaufe bedient werden kann wird genutzt, um das Rahsegel hoch und runterzuziehen und somit die Größe der Segelfläche zu verändern. Das Setzen des Segels wird durch vertikale Skalierung der Segelfläche visualisiert. Bei Grafikstilen mit höherem Detailgrad oder längerer Bearbeitungszeit wäre hier geplant das Segel beim Hochziehen zusammen zu falten. Durch den Low-Poly-Style reicht eine einfache Skalierung für eine ansprechende Darstellung aus. Durch Ziehen an der Seilschlaufe in der Installation verändern Spieler den Wert für die Falllänge. Dabei handelt es sich um einen *Float*-Parameter, der auf die Grenzwerte 10 und 100 beschränkt wird. Da die vertikale Skalierung des Rahsegels bei voller Fläche 1 beträgt handelt es sich hierbei gleichzeitig um den Wert für die Skalierung in Prozent. Die untere Grenze von 10% verhindert, dass das Segel so klein skaliert wird, dass man es nicht mehr sieht. An den unteren beiden Ecken befinden sich Metallringe, die sich mit dem Segel mitbewegen müssen, wenn es skaliert wird. Deshalb werden sie im C#-Skript basierend auf der Skalierung des Segels bewegt. Wird der Winkel zwischen Wind und Schiff so klein, dass das Rahsegel nicht mehr antreibend wirkt, wird zusätzlich das Segel so skaliert, dass der Bauch im Segeltuch flach wird. Die eingestellte Segelfläche wird Einfluss auf die Geschwindigkeit des Schiffs haben.

## 10.4 Geschwindigkeitsberechnung

Um das Schiff der Spieler zu bewegen, erhielt der Rumpf eine *Rigidbody*-Komponente, auf welche eine Kraft nach vorne wirkt. Der *Rigidbody* ist in seiner Rotation und in der vertikalen auch in seiner Position eingeschränkt, sodass die Kraft, die auf das Schiff wirkt, nur in Bewegungen auf der Wasseroberfläche resultiert. Für die vertikale Achse wurde ein C#-Skript geschrieben, um das Schiff leicht nach oben und unten zu bewegen, damit eine Schwimmbewegung imitiert wird. Auf Schräglagen des Schiffs basierend auf Wellenbewegungen und Druck auf den Segeln wurde mit Rücksicht auf Motion-Sickness

verzichtet. Da die Lenkgeschwindigkeit des Schiffs von der Vorwärtsgeschwindigkeit abhängig sein soll, bleibt das Schiff nie ganz stehen. Die Kraft, die auf das Schiff wirkt, um es nach vorne zu bewegen, wird jeden Frame neu berechnet und hat immer einen Wert von mindestens der eingestellten Windstärke. Auf diesen Wert werden pro Segel Anteile der Windstärke addiert, basierend auf ihrer Stellung zu Schiff und Windrichtung. Ist das Schratsegel ideal eingestellt wird die volle Windstärke aufaddiert, bei Abweichungen nur ein Teil. Beim Rahsegel hat die Größe der Segelstellung Einfluss auf die Geschwindigkeit. So kann die maximale Kraft auf das Schiff maximal die dreifache Windstärke betragen und minimal die Windstärke. Da diese Kraft jeden Frame auf das Schiff wirkt, muss der Trägheitswert-Wert der *Rigidbody*-Komponente genutzt werden, um zu verhindern, dass es unendlich weiter beschleunigt und stattdessen eine Höchstgeschwindigkeit erreicht.

Basierend auf dem Winkelverhältnis zwischen Schiff und Wind wird eine ideale Stellung für das Schratsegel ermittelt. Je weiter der tatsächliche Winkel des Segels davon abweicht, desto weniger Kraft wirkt auf das Schiff. Gleichzeitig werden die Windfäden rotiert, um die Abweichung den Spielern anzuzeigen. Weicht die Segelstellung zu weit ab wird keine Kraft vom Schratsegel aufaddiert und die Windfäden stehen in ihrem maximalen Winkel von  $45^\circ$  nach oben oder unten, je nachdem ob das Segel zu dicht oder zu offen ist.

Für das Rahsegel wird grundsätzlich einfach die Länge des Falls, und somit der aktuell eingestellte Anteil der maximalen Segelfläche als Kraft aufaddiert. Das bedeutet, dass eine volle Segelfläche die Kraft auf das Schiff um den vollen Wert der Windstärke erhöht. Wird das Segel möglichst hoch gezogen sind es immer noch 10% der Windstärke, da man das Rahsegel nicht vollständig einziehen kann. Ein Ausnahmefall ist die Situation, wenn der Wind eher von vorne auf das Schiff trifft. Als Grenze wurde hier ein Winkel von  $45^\circ$  zwischen Windrichtung und Schiff gewählt. Wird dieser unterschritten, wirkt das Rahsegel als Bremse, sodass abhängig von der aktuellen Größe der Segelfläche die Kraft auf das Schiff verringert wird. Um diesen Effekt einzuschränken, wird hier mit einem Faktor multipliziert, sodass maximal die halbe Windstärke verloren geht. Segeln Spieler also in einem Winkel nah am Wind, müssen sie das Rahsegel hochziehen, um nicht ausgebremst zu werden. Gleichzeitig ist es dadurch aber auch nicht möglich auf diesen Kursen die maximale Geschwindigkeit zu erreichen. Dafür müsste das Schiff wieder einen größeren Winkel zu Windrichtung einnehmen.

## 10.5 Lenken des Schiffs

Um die Lenkbewegung des Schiffs zu steuern, wird ein eigenes C#-Skript verwendet. Das Schiffmodell beinhaltet hier zwei bewegliche Teile, die abhängig von der Eingabe der Spieler Lenkbewegungen visualisieren müssen: Das Steuerrad und das Ruderblatt. Das Steuerrad wird wie schon beschrieben mit der Rotation des realen Gegenstücks in der Installation gleichgesetzt, damit Spieler die einzelnen Griffe greifen können, ohne sie zu sehen. Um das Schiff zu lenken, wird eine *Float*-Variable verwendet die gleichzeitig den Winkel des Ruderblatts beschreibt. Dieser ist zwischen  $-45^\circ$  und  $45^\circ$  beschränkt und wird durch Drehung des Steuerrads je nach Richtung erhöht oder reduziert. Steht das Steuerrad still bewegt sich das Ruderblatt langsam wieder in seine Ausgangsstellung und der Winkel geht somit Richtung  $0^\circ$ . Das bedeutet, dass das Schiff bei Loslassen des Steuers nicht im Kreis fährt, sondern nach kurzer Zeit wieder einen geraden Kurs einnimmt. Die Stellung des Steuerrads repräsentiert deshalb in diesem Fall nicht die Stellung des Ruderblatts. Da sich das reale Steuer unendlich in beide Richtungen drehen lässt, soll somit verhindert werden, dass Spieler durch übertriebene Eingaben Schwierigkeiten haben das Ruder wieder gerade zu stellen. Neben der Stellung des Ruderblatts ist die Lenkbewegung außerdem abhängig von der Geschwindigkeit des Schiffs. Diese kann von der *Rigidbody*-Komponente abgelesen werden. Eine höhere Geschwindigkeit sorgt hierbei für eine schnellere Rotation des Schiffs, um Spieler für eine gute Bedienung der Segel zu belohnen. Durch die Platzierung des Ruders lenken Segelschiffe über das Heck. Das heißt, das in Kurven der hintere Teil des Schiffs ausschwingt, anders als man es von Autos kennt. Um dieses Kurvenverhalten abzubilden, wird das Schiff bei Bedienung des Steuerrads nicht um eine Achse in der Mitte des Modells rotiert, sondern um einen Punkt, der sich weiter vorne im Schiff befindet. Dadurch wird eine Lenkung über das Heck imitiert.

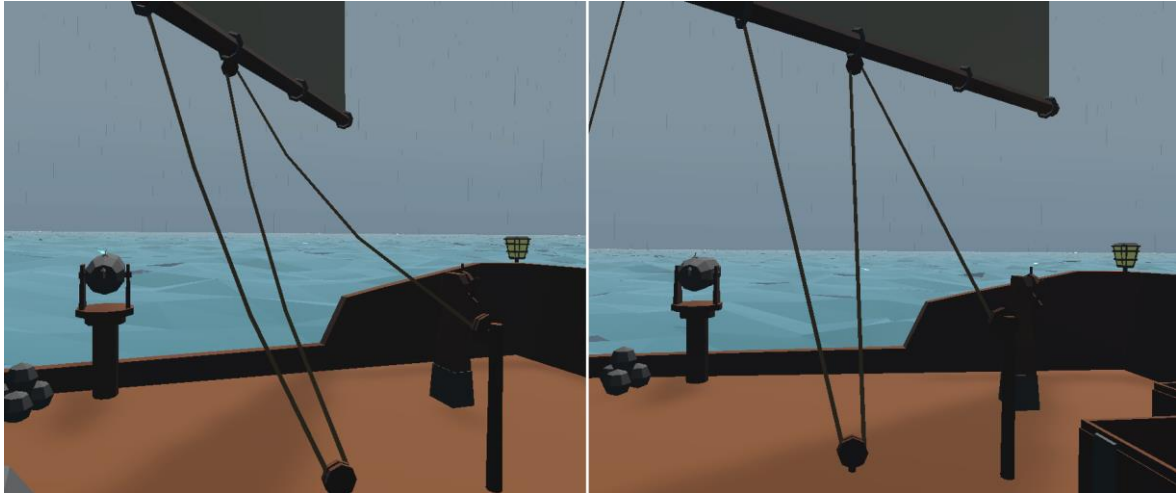
## 10.6 Darstellung der Schoten

Die Darstellung der Schiffe beinhaltet mehrere Seile, die teilweise zur Laufzeit unverändert bleiben, aber sich teilweise auch an bewegliche Teile des Schiffs anpassen müssen. Seile die starr bleiben können sind Teil des *Mesh* der Schiffsmodele. Bei Seilen, die zur Laufzeit verändert werden, wird auf eine Darstellung mit dem Unity-*LineRenderer* gewechselt. Bei diesen Seilen geht es hauptsächlich um die Schoten. Zum einen das Seilzugsystem zwischen Deck und Baum des Schratsegels, aber auch Schoten zwischen den unteren Ecken des

Rahsegels und dem Schiff. Zusätzliche dekorative Seile, die an beweglichen Teilen des Schiffs hängen sollen, werden mit der gleichen Methode dargestellt. Um ein Seil zwischen zwei Punkten zu visualisieren, wird mit dem *LineRenderer* eine Linie mit passender Breite und Farbe gezeichnet. Die beiden Endpunkte der Linie werden in einem C#-Skript während der Laufzeit angepasst, basierend auf Positionsdaten von leeren *GameObjects* die den beweglichen Teilen des Schiffs angehängt wurden. Somit verändern die Seile immer ihre Länge, sodass die beiden Stellen die sie verbinden sollen erreicht werden. Das ist zwar keine realistische Darstellung von Seilen, da sie ihre Länge verändern, aber sie ist ausreichend für den gewählten Low-Poly-Style und lässt sich auch gut für Seile nutzen mit denen Spieler nicht interagieren.

Für die Schot am Schratsegel des Spielerschiffs wurde das C#-Skript für die Seildarstellung noch erweitert. Hier kann nämlich der Fall eintreten, dass Spieler die Schot weiter Lockern als das Segel ausschlagen kann, da es schon in einem geringeren Winkel im Wind steht. Um zu visualisieren, dass die Schot aktuell nicht gespannt ist, wurden hier weitere Ankerpunkte zwischen den Endpunkten an den Schiffsteilen eingefügt. Nun wird der Winkel des Segels mit der Länge der Schot verglichen. Ist der Winkel des Segels kleiner als die Schotlänge zulassen würde, werden die mittleren Ankerpunkte bewegt, um das Seil durchhängen zu lassen. Die Stärke des entstehenden Bauchs ist hierbei davon abhängig wie groß der Unterschied zwischen Schotlänge und Winkel des Segels ist. Wird die Schot gespannt verringert sich die Größe des Bauchs, sodass das Seil wieder eine gerade Linie bildet. Auf diese Weise sehen Spieler, wenn das Segel im Wind steht und die Schot angezogen werden muss, um wieder Antrieb zu erzeugen. In der folgenden Abbildung ist zu sehen, wie das Durchhängen der Schot im Spiel dann aussieht. Sie wird durch drei einzelne Seile zwischen den Umlenkrollen erzeugt, soll aber eine Leine darstellen. Jedes dieser Teilstücke hat zwei Zwischenpunkte erhalten, um das Seil durchhängen zu lassen. Es wurde, wie in der folgenden Abbildung (Abb. 23) zu sehen, bewusst eine grobe Unterteilung gewählt um die Darstellung an den kantigen Low-Poly-Style anzupassen. Für eine realistischere Darstellung könnte man weiter Zwischenpunkte einfügen, um eine glattere Kurve zu erzeugen.





*Abb. 23: Durchhängende (links) und gespannte (rechts) Schot im Vergleich (Quelle: Eigene Aufnahme)*

# 11 Kanonenmechanik

## 11.1 Ausrichten der Kanonen

Der Input, um die Kanonen auf das gewünschte Ziel auszurichten erfolgt in der Installation durch Nutzung der beiden Steuerungselemente für die Kanonen. Die hierfür genutzten Holzboxen sind mit einem Griff versehen und lassen sich dank der verwendeten Monitorhalterung auf zwei Achsen rotieren. Das Modell der virtuellen Kanonen ist aus mehreren Teilen aufgebaut, wobei sich eine Gabelhalterung um die vertikale Achse drehen wird und das Kanonenrohr in der Halterung um die horizontale Achse. Um die Rotation der Holzboxen in die Virtualität zu übertragen, kommen Vive Tracker zum Einsatz die auf den Boxen montiert wurden. In Unity lassen sich die Rotationsdaten des Vive Trackers abfangen und können für die virtuellen Kanonen genutzt werden. Die Ausrichtung der lokalen Achsen der Vive Tracker ist hier jedoch durch die Platzierung nicht übereinstimmend mit den virtuellen Objekten, sodass eine direkte Übernahme der Werte nicht das gewünschte Ergebnis liefert. Außerdem muss die Rotation eines Vive Trackers in die vertikale und horizontale Komponente zerlegt werden, um die verschiedenen virtuellen Objekte jeweils um nur eine Achse zu rotieren. Diese Übersetzung ist zwar möglich, resultierte jedoch in zitternden Bewegungen der virtuellen Kanonen, auch wenn die Tracker nicht bewegt wurden. Eine Lösung, die dieses Problem verhinderte, war es, die virtuellen *GameObjects*, die direkt die Rotation der Vive Tracker übernehmen im Rotationsmittelpunkt der Kanonen zu platzieren. Den *GameObjects* wurde ein weiteres leeres Objekt angehängen und in seiner lokalen Position so verschoben, dass es in Ausgangslage in die Richtung des Kanonenrohrs zeigt. Wird nun der echte Vive Tracker rotiert, zeigt das leere *GameObject* in die Richtung, in die das Kanonenrohr zielen muss. Um das zu erreichen, wird in der *Update*-Methode des Kanonenskripts für die rotierbaren Teile der Kanone die *LookAt*-Funktion aufgerufen, die das Objekt rotiert, um es auf ein Ziel auszurichten. In diesem Fall wird als Ziel das leere *GameObject*, dass die gewünschte Zielrichtung zeigt, angegeben. Um jeweils nur um eine Achse der Objekte zu rotieren, werden direkt nach dem *LookAt*-Aufruf die lokalen Eulerwinkel des Objekts in den ungewünschten Achsen auf null gesetzt, sodass nur der passende Anteil der Rotation auf das Objekt angewendet wird. Durch diese Lösung ist es nun möglich die Kanonen innerhalb der Rotationsgrenzen des realen Gegenstücks auszurichten, um auf gegnerische Schiffe zu zielen. Auf Zielhilfen wie Fadenkreuze oder Anzeige der Flugbahn wurde verzichtet. Spieler müssten

durch das Spielen in der virtuellen Realität in der Lage sein, Distanzen gut abzuschätzen und mit einigen Probeschüssen ausreichend treffsicher werden. Eine Zielhilfe könnte außerdem zu Folge haben, dass feuern der Kanonen zu einfach wird.

## 11.2 Feuern der Kanonen

Neben dem Ausrichten der Kanonen soll es auch möglich sein sie zu feuern. Auf eine Nachlademechanik wurde wie beschrieben verzichtet, sodass nach dem Zünden der Kanone ein Schuss abgefeuert wird und keine weitere Spielerinteraktion nötig ist, um den nächsten Schuss zu zünden. Eine Kanone lässt sich zünden indem Spieler den Luntentab, dessen Position und Rotation im Raum durch einen Vive Tracker ermittelt wird, an die Lunte der Kanone halten. Luntentab und Kanone sind hier jeweils mit *Collidern* versehen, um diese Interaktion in Unity abzufangen. Kommt es zu einer Kollision wird zunächst nicht direkt eine Kugel gefeuert, sondern die Lunte der Kanone gezündet. Visualisiert wird dies durch Partikel die Funken darstellen. Die Partikel beginnen an der Spitze der Lunte und wandern durch ein C#-Skript weiter bis sie das Kanonenrohr erreichen. Mehrere leere *GameObjects* werden hier genutzt, um die Route der Funken an die Kurven der Lunte anzupassen. Während die Lunte runterbrennt, läuft ein Zähler runter der für die Verzögerung des Schusses genutzt wird. Erreichen Lunte und Zähler ihr Ende wird eine neue Instanz des Kanonenkugel-*Prefabs* erstellt und im Kanonenrohr platziert. Dieses *Prefab* beinhaltet neben dem 3D-Modell für die Kanonenkugel eine *Rigidbody*-Komponente um der Kugel physikalische Eigenschaften wie Gravitation zu verleihen. Um die Kugel zu feuern erhält der *Rigidbody* eine Geschwindigkeit in Richtung des Kanonenrohrs, um die Ausrichtung durch Spieler zu beachten. Sie wird mit einem Faktor multipliziert, um eine passende Schussweite zu erreichen. Zusätzlich muss die Geschwindigkeit des Schiffs auf die Geschwindigkeit der Kugel addiert werden, um ein realistisches Flugverhalten zu schaffen. Beim Abfeuern eines Kanonenschusses wird zusätzlich eine Animation ausgelöst, die das Schiff ein wenig wackeln lässt, um die Kraft des Schusses für Spieler spürbarer zu machen. Mit Rücksicht auf mögliche Motion-Sickness ist das Wackeln der Animation aber eher kurz und schwach. Nach dem Schuss müssen Spieler einen kurzen Moment warten, bis sie die Kanone wieder zünden können. Die Kanonenkugel interagiert mit verschiedenen Objekten in der Spielwelt. Landet sie im Wasser oder auf Landobjekten werden Partikel erzeugt, um den Einschlag zu visualisieren und das Kugelobjekt wird gelöscht. Die Interaktion mit gegnerischen Schiffen wird im folgenden Kapitel beschrieben.

## 12 Gegnerverhalten

In der aktuellen Version der Spielwelt befinden sich fünf gegnerische Schiffe, die von den Spielern versenkt werden können. Um Gegnerschiffe in beliebiger Anzahl einzufügen wurden alle benötigten Objekte und C#-Skripte für ein Schiff in einem *Prefab* zusammengefasst, sodass einfach Kopien des *Prefabs* erstellt werden können und Änderungen an den Gegnerschiffen nur an einer Stelle stattfinden müssen. Grundsätzlich soll jedes Gegnerschiff zwei Verhaltensarten beinhalten. Bei Start der Anwendung soll ein gegnerisches Schiff einer vorher definierten Route in der Spielwelt folgen. Sobald das Spielerschiff gesichtet wird, sollen Gegner in ihr Kampfverhalten wechseln und auf das Spielerschiff schießen. Sollte ein Gegnerschiff den Spielern entkommen, soll es wieder zu seiner Route zurückkehren.

### 12.1 Folgen der Route

Um das Fahrverhalten der Gegnerschiffe zu definieren, wurde anfangs mit einem C#-Skript gearbeitet, dass leere *GameObjects* als Marker einer Route nutzt und diese entlangfährt. Endgültig wurde dann jedoch das Unity *NavMesh-System* verwendet, um gegnerische Schiffe zu bewegen. Dieses System erlaubt es ein *Mesh*, in diesem Fall die Wasserfläche, als Bereich zu definieren auf dem sich von Unity gesteuerte Objekte bewegen können. Diese werden als *NavMesh Agent* bezeichnet und liefern mehrere Parameter in denen sich Geschwindigkeit, Drehgeschwindigkeit, Beschleunigung und weitere Eigenschaften des *Agents* definieren lassen. Jedes Gegnerschiff hat also eine *NavMesh Agent*-Komponente, die für die Bewegung des Schiffs verantwortlich ist. Um für jedes Schiff eine eigene Route zu definieren, werden wie in den ersten Prototypen leere *GameObjects* genutzt. In einem C#-Skript werden die Routenmarker in einer Liste gespeichert. Um die Routen im Unity Editor zu visualisieren, wurde die Funktion *OnDrawGizmosSelected* verwendet, um Punkte an den Stellen der Marker zu zeichnen und diese mit Linien zu verbinden. Diese Visualisierung dient nur Entwicklungszwecken und ist nur in der Szenenansicht des Editors sichtbar. Im Anhang ist eine Übersicht der Spielwelt zu finden auf der die Routen auf dieser Weise eingezeichnet wurden. Um ein Gegnerschiff zu einem der Marker fahren zu lassen, lässt sich in der *Agent*-Komponente ein Ziel festlegen. In einem C#-Skript wird der Abstand von Schiff und aktuellem Ziel in der *Update*-Methode kontrolliert. Sobald sich das Schiff in der Nähe des Ziels (< 1 Meter) befindet, wird das Ziel des *Agenten* mit dem nächsten Marker in der Liste ausgetauscht und das Schiff sucht einen Weg dorthin. Diese Umsetzung hat zu Folge das der Index der Marker in der Liste gleichzeitig die Reihenfolge ist, in der das Schiff die Route abfährt. Um eine geschlossene Route zu erzeugen, die ein Schiff theoretisch unendlich lange befahren kann, wird bei Erreichen des letzten Markers in der Liste der Marke mit Index

0 als nächstes Ziel gewählt. Die Verwendung des *NavMesh* lässt zusätzlich zu, dass die Landobjekte wie Inseln und Felsen, sowie andere Schiffe als *NavMesh Obstacle* markiert werden können. Hierbei handelt es sich um Objekte, die bei der Wegfindung der *NavMesh Agents* automatisch umfahren werden.

Um das Segelverhalten der Gegnerschiffe an das Schiff der Spieler anzupassen sind seine Segel in ähnlicher Weise aufgebaut. Das hintere Rahsegel ist mit einer *HingeJoint*-Komponente und einem *Collider* versehen, sodass auf gleiche Weise Windkraft und Richtung darauf angewendet werden können. Um das Rahsegel in einen passenden Winkel zum Wind zu drücken, wird die Kraft jedoch auf zwei Stellen im Segel angewandt, auf Backbord- und Steuerbordseite. Auf diese Weise nimmt das Segel eine senkrechte Stellung zum Wind ein, sofern die Grenzen der Rotation noch nicht erreicht wurden. Der Winkel des vorderen Rahsegels wird einfach mit dem hinteren gleichgesetzt. Das dreieckige Vorsegel der Gegnerschiffe wird in seinem Winkel basierend auf den Rahseglern angepasst. Die Rahseglern sollen wie beim Spielerschiff in einem Wind-Schiff-Winkel von  $45^\circ$  und weniger nicht mehr antreibend wirken. Segeln Gegner aufgrund ihrer Route einen solchen Kurs, werden die Rahseglern hochgezogen, um ihre Segelfläche zu verkleinern. Hier kommt dann nur noch das Vorsegel zum Einsatz. Ist der Winkel zum Wind wieder groß genug werden die Rahseglern erneut gesetzt. In der aktuellen Umsetzung wird noch nicht beachtet ob die Gegnerschiffe direkt in Windrichtung fahren. Die Geschwindigkeit der Gegnerschiffe ist grundsätzlich von den gleichen Faktoren abhängig wie das Spielerschiff. Sie wird auf Basis der Windstärke und aktuellen Segelfläche berechnet. Ein Faktor reduziert die Maximalgeschwindigkeit jedoch ein wenig, um es Spielern einfacher zu machen Gegner einzuholen.

## 12.2 Kampfverhalten

Sobald das Spielerschiff eine Distanz zu einem Gegner unterschreitet, die ungefähr der maximalen Schussweite der Schiffe entspricht, wechselt das Gegnerschiff in den Kampfmodus. In diesem Moment hört es auf seiner Route zu folgen und nimmt zunächst einen geraden Kurs an. Die Kanone des Gegnerschiffs, die dem Spielerschiff näher liegt, beginnt auf das Spielerschiff zu zielen. Die Rotation der Kanone ist hierbei auch wie bei den Spielern limitiert. Um zu verhindern, dass Gegner bei jedem Schuss treffen, wird die Zielrichtung mit einem zufälligen, kleinen Fehlerbetrag auf beiden Drehachsen versehen. Ist die Kanone ausgerichtet wird ein Schuss gefeuert. Dieser ist wie beim Spielerschiff durch die Lunte verzögert und lässt das Schiff beim Abfeuern kurz wackeln. Auch die Geschwindigkeit, mit der Schüsse gefeuert werden ist bei Gegnern und Spielern aktuell gleich. Auf jeden Schuss folgt eine Abkühlzeit, bis

der nächste gefeuert werden kann. Trifft der Fall ein, dass sich das Spielerschiff in Sichtdistanz zu einem Gegnerschiff befindet, dieses aufgrund der Rotationslimitierungen der Kanonen aber nicht in der Lage ist auf das Spielerschiff zu schießen, passt es seinen Kurs an. Dafür lenkt das Gegnerschiff in eine Richtung, die durch die Richtung, in der die Kanone ihr Limit erreicht hat, definiert wird. Es kann der Fall eintreten das Spieler den Sichtbereich eines gegnerischen Schiffs wieder verlassen, ohne dass es versenkt wurde. Sollte das passieren sucht das Gegnerschiff den Punkt seiner Anfangsroute dem des aktuell am nächsten ist und fährt dort hin. Danach wird die Route in der normalen Reihenfolge weiter abgefahren, bis das Spielerschiff wieder in Sichtweite kommt.

## 12.3 Versenken von Gegnern

Sowohl das Spielerschiff als auch die Gegnerschiffe sind mit *Collidern* um den Rumpf ausgestattet, um zu registrieren, wann sie von einer Kanonenkugel getroffen werden. Bei beiden Schiffstypen wird der Einschlag durch einen Partikeleffekt an der Trefferstelle sowie einer Wackel-Animation visualisiert, wobei die Schiffe verglichen mit der Abschussanimation ein wenig mehr wackeln, wenn sie getroffen werden. In der aktuellen Version der Umsetzung haben Kanonentreffer von Gegnern von der Visualisierung abgesehen keine Konsequenzen für das Spielerschiff. Ein Spielabbruch würde bei Präsentation des Projekts stören und wäre eher für regelmäßiges Spielen sinnvoll. Gegnerische Schiffe können hingegen von Spielern mit Hilfe der Kanonen versenkt werden. Ein Gegnerschiff muss zweimal mit einem Kanonenschuss getroffen werden, um zu sinken. Der erste Treffer mit einer Kanonenkugel hat zur Folge, dass das Gegnerschiff beginnt zu brennen. Das Feuer wurde hier, wie in der folgenden Abbildung (Abb. 24) zu sehen, mit dem Partikelsystem umgesetzt.



Abb. 24: Brennendes Gegnerschiff (Quelle: Eigene Aufnahme)

Sobald ein Gegnerschiff von einer zweiten Kanonenkugel getroffen wird, beginnt es zu sinken. Hierbei werden sofort alle Komponenten und C#-Skripte, die das Schiffsverhalten zuvor definierten entfernt. Das Schiff kippt langsam zu Seite und sinkt gleichzeitig nach unten. Sobald es unter der Wasseroberfläche verschwindet, wird das Objekt vollständig gelöscht. An der Stelle, an der das Gegnerschiff gesunken ist, wird nun ein Modell eingefügt das schwimmende Überreste von Schiff und Ladung darstellen soll. Dieses Objekt ist eine Instanz eines *Prefabs*, das gleichzeitig ein C#-Skript für eine Schwimmbewegung sowie einen *Collider* beinhaltet. Durch den *Collider* kann abgefragt werden ob das Spielerschiff durch die hinterlassene Ladung, die in der folgenden Abbildung (Abb. 25) betrachtet werden kann fährt, um den Plunder einzusammeln. Sobald die Spieler mit ihrem Schiff durch das Treibgut fahren, wird das Objekt wieder gelöscht und die Kisten des Spielerschiffs füllen sich ein Stück mit Gold, um den Spielfortschritt zu zeigen. Teilweise mit Gold gefüllte Kisten sind in der folgenden Abbildung (Abb. 25) zu sehen. Sobald alle fünf Gegnerschiffe versenkt und ihre Ladung eingesammelt wurden, sind auch die Kisten der Spieler bis zum Rand gefüllt. Das Gold-Material verwendet einen anderen Shader, damit es metallisch glänzt.

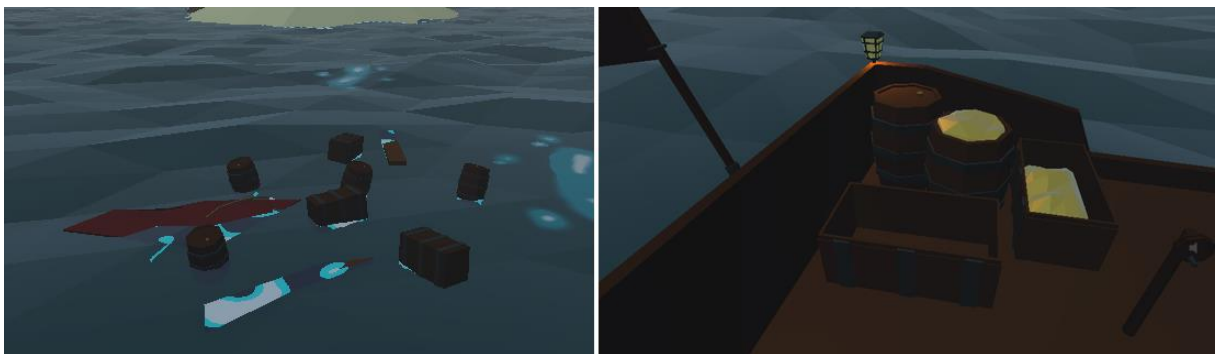


Abb. 25: Hinterlassene Ladung eines versenkten Gegnerschiffs und mit Gold befüllte Kisten auf Spielerschiff (Quelle: Eigene Aufnahme)

## 13 Sound-Design

In Unity ist es möglich Soundquellen im 3D-Raum zu platzieren. Die Kameras an den VR-Rigs der Spieler sind mit *AudioListener*-Komponenten versehen. Durch Vergleich der Spielerkamera- und Soundquellen-Position entsteht so automatisch ein 3D-Sound, der sich den Bewegungen der Spieler anpassen kann. Die Soundausgabe erfolgt aktuell durch die Kopfhörer an den VR-Headsets. Dabei handelt es sich um Lautsprecher, die nicht anliegen oder das Ohr umschließen, sondern in einem kleinen Abstand vor dem Ohr hängen. Dadurch wird sichergestellt, dass Spieler Geräusche in der realen Welt auch beim Tragen des Headsets weiterhin gut hören können. Das ist für dieses Projekt wichtig, da die Kommunikation der Spieler für das kooperative Steuern des Schiffs zentraler Bestandteil des Gameplays ist.

Die Sounds für dieses Projekt wurden nicht selbst aufgenommen, sind durch ihre Lizenzen jedoch frei zugänglich. Da das Spiel durch die Abhängigkeit von der gebauten VR-Installation nicht verbreitet werden kann und nicht für kommerzielle Zwecke genutzt werden soll, könnte auch keine Urheberrechtsverletzung stattfinden.

Grundsätzlich wurden zwei Arten von Sounds in das Spiel integriert. Umgebungsgeräusche, die in einem dauerhaften Loop abgespielt werden und die Stimmung der Spielwelt vermitteln sollen und aktionsgebundene Sounds, die durch ein bestimmtes Event ausgelöst werden.

### 13.1 Ambient-Sounds

Die Ambient-Sounds umfassen alle Geräusche, die dauerhaft von den Spielern wahrgenommen werden. Einige dieser Geräusche sind hierbei nicht Teil des 3D-Sounds und werden als Stereo-Spur direkt für die Spieler abgespielt. Hierzu gehören Umgebungsgeräusche, denen keine konkrete Position zugeordnet werden kann wie Wind, Wasserplätschern, Wellen, Regen und Holzknarzen auf dem Schiff. Andere Geräusche, die in Dauerschleife laufen sind mit bestimmten Objekten verbunden und werden als 3D-Sound wahrgenommen. Beispiele hierfür sind das Flattern der Segel oder das Schreien des Möwenschwarms. Der Papagei auf der Schulter des Kapitäns gibt in unregelmäßigen Abständen ein kurzes Krächzen von sich.



## 13.2 Aktionsgebundene Sounds

Aktionsgebundene Sounds werden durch ein bestimmtes Ereignis im Spielverlauf ausgelöst und sind mit einem Objekt verbunden und somit Teil des 3D-Sounds. Ein Beispiel wäre das Schießen einer Kanone. Zunächst wird ein Geräusch ausgelöst, das das Brennen der Lunte darstellt. Daraufhin folgt ein Kanonenschuss, sowie Geräusche, die zur Wackelanimation des Schiffs passen. Schlägt die Kanonenkugel in ein Objekt ein, wird ein zum Material passender Sound abgespielt, je nachdem, ob es sich um Wasser, ein Schiff oder Land handelt. Weitere Geräusche sind mit beweglichen Teilen des Schiffs verbunden. Wenn durch Spielerinteraktion zum Beispiel die Schotkurbel oder das Steuerrad bewegt werden, werden passende Geräusche abgespielt. Auch die Bewegung der Segel, die daraus resultiert, wird durch Sounds unterstützt. Um Spielern zu zeigen, dass sie von einem gegnerischen Schiff entdeckt wurden, sind die Gegnerschiffe mit einer Alarmglocke ausgerüstet. Sobald das Spielerschiff die maximale Sichtdistanz eines Gegners unterschreitet, wird die Glocke mit einer Animation geläutet und ein passender Sound wird abgespielt. Das Läuten der Glocke geschieht nicht dauerhaft, sondern für die ersten paar Sekunden, nachdem das Spielerschiff entdeckt wurde. Auch das Feuer auf brennenden Gegnerschiffen ist mit Geräuschen versehen, aber nur hörbar, wenn Spieler sehr nah an den Gegner heran fahren.

## 14 Multiplayer

Damit drei Spieler ein Schiff zusammen in der VR-Installation steuern können, musste das Game um eine Multiplayer-Komponente erweitert werden. Jede der drei VR-Brillen benötigt einen eigenen Computer, auf dem die Anwendung läuft, sodass sich die drei Instanzen des Spiels gegenseitig Daten zuschicken müssen, um diese zu synchronisieren. Da die verwendeten Lighthouses nur passiv für das Tracking verwendet werden und die eigentliche Berechnung über die Headsets stattfindet, ist es nicht nötig verschiedene Lighthouses für die drei VR-Headsets aufzubauen. Sie können unabhängig voneinander die gleichen Lighthouses verwenden. Um den Multiplayer umzusetzen, wurde das Unity-eigene *Package* „Netcode for GameObjects“ verwendet. Dieses *Package* beinhaltet Netzwerk-Funktionen, die für den Aufbau von Multiplayerspielen genutzt werden. Da dieses Projekt jedoch nicht wie ein klassisches Multiplayer-Game aufgebaut ist und mit sehr asymmetrischen Instanzen des Spiels arbeiten wird, wurden die Hauptfunktionen des *Netcodes* nicht verwendet. Was aber zu Verwendung kam, war die Möglichkeit eine Instanz des Spiels als Host oder Client zu starten, um eine Netzwerkverbindung zwischen diesen herzustellen. Der Host beinhaltet hierbei die Hauptfunktionen des Spiels und übernimmt die Verarbeitung der meisten Inputs. Die beiden Clients verbinden sich mit dem Host und erhalten über das Netzwerk Daten, die zur Synchronisation der Spielinstanzen genutzt werden. Diese Daten werden mit einer weiteren Funktion des *Packages*, den Netzwerkvariablen übertragen. Dies sind Variablen, die im Netzwerk zu Verfügung gestellt werden. Die Clients können die Variablen dann abfangen und verwenden. Eine genauere Erklärung zum Aufbau von Host und Client folgt den nächsten beiden Unterkapiteln. Die folgende Abbildung (Abb. 26) zeigt einen groben Überblick welche Daten zwischen Host und Clients versendet werden.

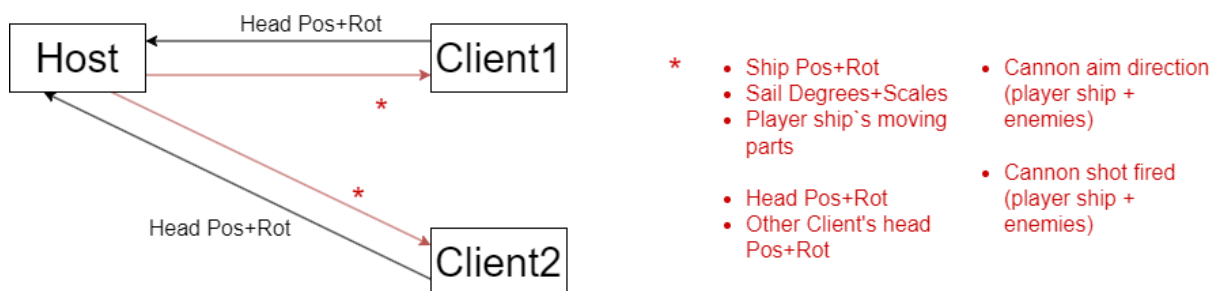


Abb. 26: Übersicht über Struktur des Multiplayers (Quelle: Eigene Aufnahme)

Um die drei Computer in einem Netzwerk zu verbinden wird ein LAN-Switch verwendet. Die Datenübertragung erfolgt dadurch unabhängig vom Internet. Damit die Clients sich mit dem Host verbinden können benötigen sie die IP-Adresse des Hosts.

## 14.1 Host

Als Host wird, der im Raum schon vorhandene Computer in der Verbindung mit der HTC Vive Pro verwendet. Neben dem Headset sind außerdem die drei Vive Tracker, sowie die drei Mikrocontroller-Boards aus der Installation mit diesem Computer verbunden. Das heißt alle Interaktionen mit denen Spieler das Schiff und die Kanonen steuern, werden vom Host verarbeitet. Damit auch die anderen beiden Spieler die Effekte der Inputs in ihrer Instanz des Spiels sehen, muss der Host die dafür nötigen Daten versenden. Dazu gehören grundlegende Transformations-Information wie die Position und Rotation des Spielerschiffs und die Positionen und Rotationen der Gegnerschiffe. Auf dem Spielerschiff gibt es bewegliche Teile wie Segel und Kanonen, die durch die Steuerungselemente in der Installation verändert werden. Diese Veränderungen müssen auch über das Netzwerk an die Clients weitergegeben werden. Da die Gegnerschiffe ihre Segel an ihren Winkel zum Wind ausrichten kann dies weiterhin Client-seitig passieren. Nur die Windrichtung und Windstärke wird hierfür übertragen für den Fall, dass sich diese Werte bei Weiterentwicklungen des Projekts während der Laufzeit ändern. Alle Funktionen, die keinen Einfluss auf das Gameplay haben, wie zum Beispiel die Routenberechnung des Möwenschwarms sind aktuell Client-seitig, auch wenn es hier zu Asymmetrien kommen kann. Viele Elemente wie Skripte für das Verhalten am Rand der Welt oder die *HingeJoint*-Komponente für das Segel des Spielerschiffs konnten bei den Clients entfernt werden, da die Berechnung auf Host-Seite stattfindet und nur die Ergebnisse übertragen werden. Damit die Clients sehen wo sich der Spieler, der den Host-PC nutzt im Raum befindet wird die Position und Rotation seines VR-Headsets an die Clients übertragen.

Bei den Kanonen wird sowohl für das Spieler- als auch für die Gegnerschiffe die Ausrichtung Host-seitig berechnet. Auch das Zünden einer Kanone von Spielern oder Gegnern wird auf Host-Seite registriert. Der eigentliche Schuss passiert aktuell jedoch Client-seitig. Alle Collider, die für Kanonenschüsse relevant sind, sind sowohl bei Host als auch bei Client vorhanden, sodass die Schüsse von jeder Instanz einzeln berechnet werden. Diese Struktur soll die Menge an Daten, die versendet werden muss reduzieren.

## 14.2 Clients

Die beiden Clients erhalten die beschriebenen Daten vom Host und wenden diese auf Kopien der Schiffe in ihrer Instanz an. Den Client-Schiffen wurden alle Funktionen entfernt, die für die Verarbeitung von Inputs verantwortlich waren, da sie nicht mit den Steuerungselementen verbunden sind. Die Veränderungen durch Inputs werden auf die Host-Schiffe angewandt und das Resultat wird an die Clients weitergegeben, damit sich ihre Instanzen synchronisieren können. Die einzige Information, die der Host von den Clients benötigt ist, die Position und Rotation des jeweiligen VR-Headsets, um die Modelle für die Charaktere richtig zu positionieren. Da der Host von beiden Clients ihre Headset-Daten erhält, kann er seine eigenen, sowie die Daten des anderen Clients senden, ohne dass die beiden Clients miteinander kommunizieren müssen. Durch den neuen Aufbau der Charaktermodelle, können die Positionen und Rotation der einzelnen Köpfe mit den jeweiligen Headset-Daten gleichgesetzt werden. Die entsprechenden Körper werden mit einem kleinen Abstand unter dem Kopf platziert und um ihre vertikale Achse rotiert, um sich an die Blickrichtung des Kopfes anzupassen.

## 15 Fazit und Ausblick

### 15.1 Aktueller Zustand von VR-Installation und Game

Die VR-Installation konnte vollständig entsprechend des geplanten Konzepts umgesetzt werden. Die einzelnen Steuerelemente funktionieren wie gewünscht. Kleinere Veränderungen des Konzepts die sich während der Umsetzungsphase ergeben haben hatten eher Verbesserungen als Nachteile zur Folge. Das größte Problem der Installation, das sich während Tests des Projekts bemerkbar gemacht hat war das unsaubere Tracking der VR-Headsets. Die beiden Lighthouses im Raum reichen nicht aus, um den gesamten Raum zuverlässig abzudecken, sodass es immer wieder zu Trackingproblemen kommt, die zu Folge haben, dass entweder das Bild in der Brille stehen bleibt, grau wird oder sich das Display des Headsets vollständig ausschaltet und ein Neustart der Anwendung nötig wird. Verstärkt wird dieses Problem durch die gebauten Steuerungselemente, da die vertikalen Holzplatten im Raum zusätzlich für Verdeckungen sorgen, sodass in weiteren Bereichen des Raums das Tracking nicht funktioniert. Nutzen wie geplant drei Spieler die Installation gleichzeitig kann es außerdem dazu kommen, dass sie sich gegenseitig verdecken. Bei den beiden Kanonen dürfen sich Spieler nicht direkt vor das Input-Element stellen, da ihr Körper sonst die Vive Tracker blockiert. Um diesen Problemen entgegenzuwirken, müssten zusätzliche Lighthouses im Raum platziert werden. Diese könnten Bereiche abdecken, in denen es zuvor Trackingprobleme gab.

Das Spiel wurde mit seinen geplanten Funktionen umgesetzt. Es ist möglich das Schiff mit den Steuerungselementen der VR-Installation wie beschrieben zu segeln. Gegnerschiffe bewegen sich wie geplant in der Spielwelt und reagieren auf das Spielerschiff, sodass Kanonengefechte zu Stande kommen können. Der Multiplayer ist fertiggestellt, konnte aktuell jedoch noch nicht ausgiebig getestet werden. Die Synchronisation von Host und Client funktioniert wie beschrieben, jedoch könnten noch Fehlerquellen vorhanden sein die durch Tests der Anwendung gefunden und beseitigt werden müssen.

Um Spielern die Funktionsweise der Installation sowie die benötigten Grundlagen des Segelns zu vermitteln, wurde ein Poster erstellt, das im Anhang zu finden ist. Es beschreibt die passenden Segelstellungen für verschiedene Schiffswinkel zum Schiff und beinhaltet eine Zeichnung des Spielerschiffs mit Begriffen für die wichtigsten Bestandteile. Ein paar der hier

zu sehenden Segelbegriffe sind für das Spiel nicht unbedingt notwendig, können aber genutzt werden falls sich Spieler dafür interessieren.

## 15.2 Mögliche Erweiterungen und Verbesserungen

Auch wenn die VR-Installation erfolgreich umgesetzt wurde, bietet das Projekt Erweiterungsmöglichkeiten. Ein erster Schritt wäre es die Komplexität von Spiel und Installation zu erhöhen. Ein Beispiel wäre es die Ausrichtung des Rahsegels zum Wind nicht länger automatisch geschehen zu lassen und stattdessen ein weiteres Steuerungselement einzufügen. Weitere Segel mit eigenen Inputelementen wären auch denkbar, vor allem wenn ein größerer Raum als Schiffsdeck genutzt werden kann. Die vorhandenen Steuerungselemente lassen sich teilweise auch noch verbessern. Für ein besseres haptisches Gefühl könnte man die Kurbel für die Schot durch eine Holzkurbel austauschen. Gleichzeitig sollte man die Halterung umbauen, um zu erlauben den Griff in beliebiger Position loszulassen, ohne dass die Gravitation ihn nach unten zieht. Eine Begrenzung der Umdrehungen, die man mit der Kurbel tätigen kann, zum Beispiel durch ein Seil, das sich aufrollt, würde es möglich machen die tatsächliche Rotation der Kurbel als Input zu verwenden, anstatt wie aktuell nur Veränderungen zu beachten. Auch das Steuerrad sollte in seiner Rotation begrenzt werden. Zusätzlich wäre es hier noch gut, wenn es sich beim Loslassen wieder in seine Ausgangslage dreht, sodass das Ruder wieder geradesteht wenn man nicht lenkt. Auch wenn es hier nicht umgesetzt wurde, wäre eine Nachlademechanik für die Kanonen dennoch eine Bereicherung für die VR-Installation. Mit mehr Entwicklungszeit wäre eventuell ein umsetzbares Konzept zustande gekommen.

Die Unity-Anwendung kann in ihrer grafischen Gestaltung noch stark verbessert werden. Die aktuellen Landobjekte haben nur wenig Varianz, die sich auch in der Spielwelt bemerkbar macht. Mit mehr Entwicklungszeit könnte hier mehr gestaltet werden. Auch in den Gegnerschiffen gibt es Potential für mehr Variation. Hier könnte man verschiedene Schiffstypen mit eigenen Verhaltensweisen erstellen, um die Spieler auf unterschiedliche Arten zu fordern. Auch das Kampfverhalten der Gegner könnte erweitert werden, sodass die Schiffe auch komplexere Manöver fahren, um Oberhand zu gewinnen.

Für eine dauerhaftere Ausstellung der Installation wäre es außerdem sinnvoll das Spielerschiff angreifbar zu machen, sodass man sinken kann.

# Literaturverzeichnis

**CG Geek (2019):** How to Create a Low Poly Character in Blender 2.8.

[https://www.youtube.com/watch?v=Ljl\\_QFs9xhE](https://www.youtube.com/watch?v=Ljl_QFs9xhE)

(Letzter Abruf: 15.02.2023)

**Electric Diy Lab (2020):** How to connect optical rotary encoder with Arduino.

<https://electricdiy.com/how-to-connect-optical-rotary-encoder-with-arduino/>

(Letzter Abruf: 15.02.2023)

**Europäisches Segel-Informationssystem [1] (2021):** Alles über Segel: Das Rahsegel.

<https://www.esys.org/segel/Rahsegel.html>

(Letzter Abruf: 15.02.2023)

**Europäisches Segel-Informationssystem [2] (2021):** Alles über Segel: Das Schratsegel.

<https://www.esys.org/segel/Schratsegel.html>

(Letzter Abruf: 15.02.2023)

**Europäisches Segel-Informationssystem [3] (2021):** Alles über Segel: Das Gaffelsegel.

<https://www.esys.org/segel/Gaffelsegel.html>

(Letzter Abruf: 15.02.2023)

**Flaroon (2021):** Unity Low Poly Water in Shader Graph With Foam v2!

<https://www.youtube.com/watch?v=78WCzTVmc28>

(Letzter Abruf: 15.02.2023)

**Hennejoe (2020):** HOW TO CREATE: Cartoon Wind Effect (Legend of Zelda: The Wind Waker) in Unity.

<https://www.youtube.com/watch?v=Jj8UHGe5Aps>

(Letzter Abruf: 15.02.2023)

**HTC Corporation [1] (o. D.):** Vive Pro Full Kit.

<https://www.vive.com/de/product/vive-pro-full-kit/>

(Letzter Abruf: 15.02.2023)

**HTC Corporation [2] (o. D.):** Vorstellung von VIVE Tracker (3.0).

<https://www.vive.com/de/accessory/tracker3/>

(Letzter Abruf: 15.02.2023)

**HTC Corporation [3] (2021):** Hand Tracking Unity Plugin.

<https://hub.vive.com/storage/tracking/unity/index.html>

(Letzter Abruf: 15.02.2023)

**Microsoft Store (2018):** Sea of Thieves.

<https://www.xbox.com/en-gb/games/store/sea-of-thieves/9p2n57mc619k>

(Letzter Abruf: 15.02.2023)

**Schell, Jessie (2020):** Die Kunst des Game Designs. 3rd edition.

[https://learning.oreilly.com/library/view/die-kunst-](https://learning.oreilly.com/library/view/die-kunst-des/9783747502099/Text/final_stage2_17.htm#:text=Bei%20der%20Arbeit,so%20%C2%BBnormaler%C2%AB%20erschien.)

[des/9783747502099/Text/final\\_stage2\\_17.htm#:-](https://learning.oreilly.com/library/view/die-kunst-des/9783747502099/Text/final_stage2_17.htm#:text=Bei%20der%20Arbeit,so%20%C2%BBnormaler%C2%AB%20erschien.)

[:text=Bei%20der%20Arbeit,so%20%C2%BBnormaler%C2%AB%20erschien.](https://learning.oreilly.com/library/view/die-kunst-des/9783747502099/Text/final_stage2_17.htm#:text=Bei%20der%20Arbeit,so%20%C2%BBnormaler%C2%AB%20erschien.)

[Erscheinungsort nicht ermittelbar]: mitp Verlag

(Letzter Abruf: 15.02.2023)

**Sea of Thieves Wiki (2022):** Sailing.

<https://seaofthieves.fandom.com/wiki/Sailing>

(Letzter Abruf: 15.02.2023)

**Smithsonian American Art Museum (2015):** The Art of Video Games Exhibition Checklist.

[https://web.archive.org/web/20151221025900/http://americanart.si.edu/exhibitions/archive/2012/](https://web.archive.org/web/20151221025900/http://americanart.si.edu/exhibitions/archive/2012/games/winninggames.pdf)  
[games/winninggames.pdf](https://web.archive.org/web/20151221025900/http://americanart.si.edu/exhibitions/archive/2012/games/winninggames.pdf)

(Letzter Abruf: 15.02.2023)

**Tom (o. D.):** Was ist scheinbarer Wind?

<https://segelplanet.de/was-ist-scheinbarer-wind/>

(Letzter Abruf: 15.02.2023)

**Ubisoft (o. D.):** Assassin's Creed IV Black Flag.

<https://www.ubisoft.com/de-de/game/assassins-creed/iv-black-flag>

(Letzter Abruf: 15.02.2023)

**World of VR (o. D.):** Was ist Outside-In Tracking?

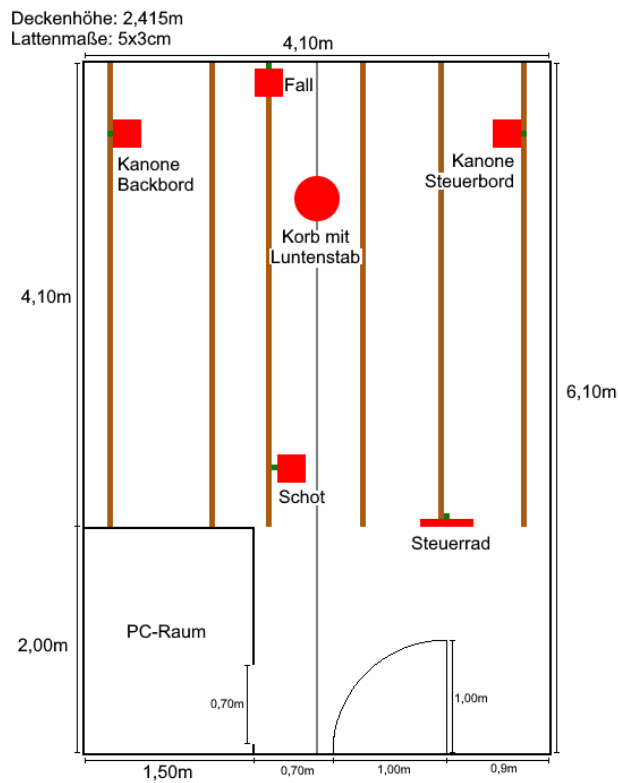
<https://worldofvr.de/outside-in-tracking/>

(Letzter Abruf: 15.02.2023)



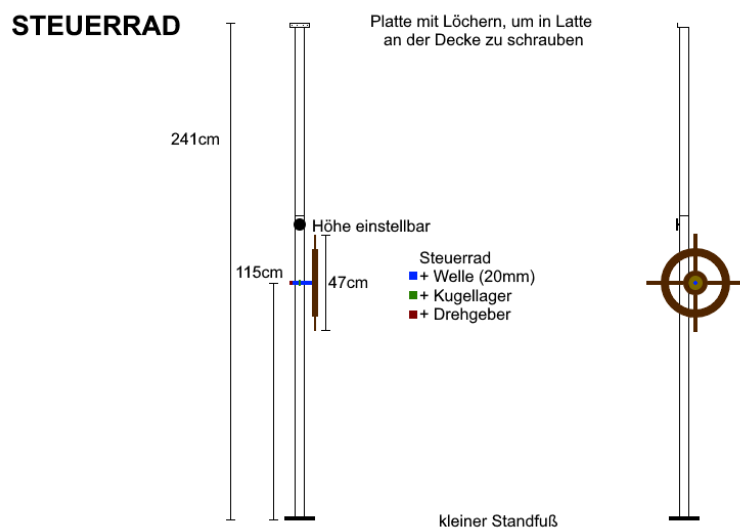
# Anhang

## Raumplan



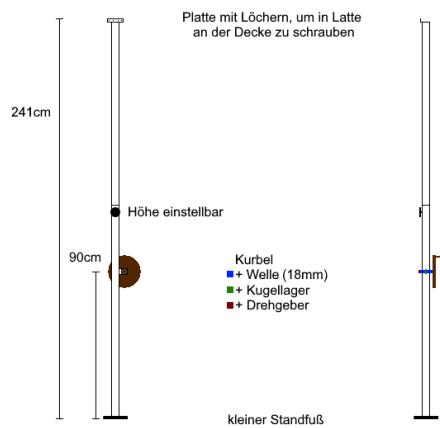
Anmerkungen: Die Positionen der Kanonen haben sich verändert. Sie wurden an den Wänden angebracht. Für den Luntenstab wurde noch keine Korb platziert, um Laufwege weniger zu blockieren. Spieler können diesen auf dem Boden ablegen oder in der Hand behalten.

## Konzeptskizze Steuerrad



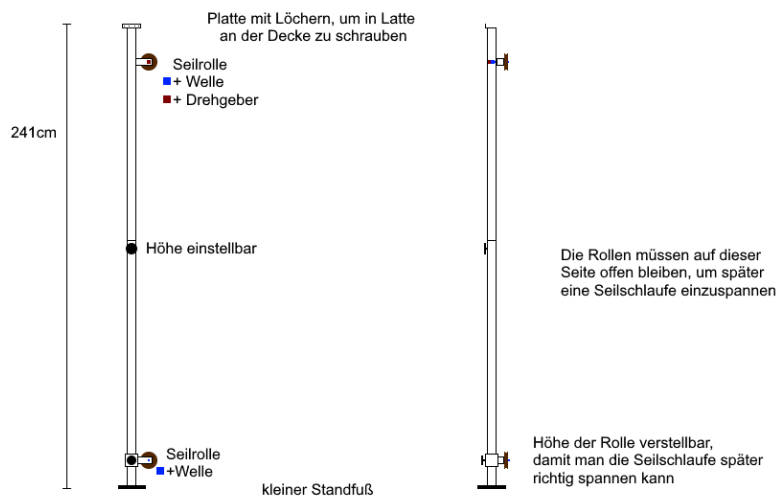
# Konzeptskizze Schot

## SCHOT



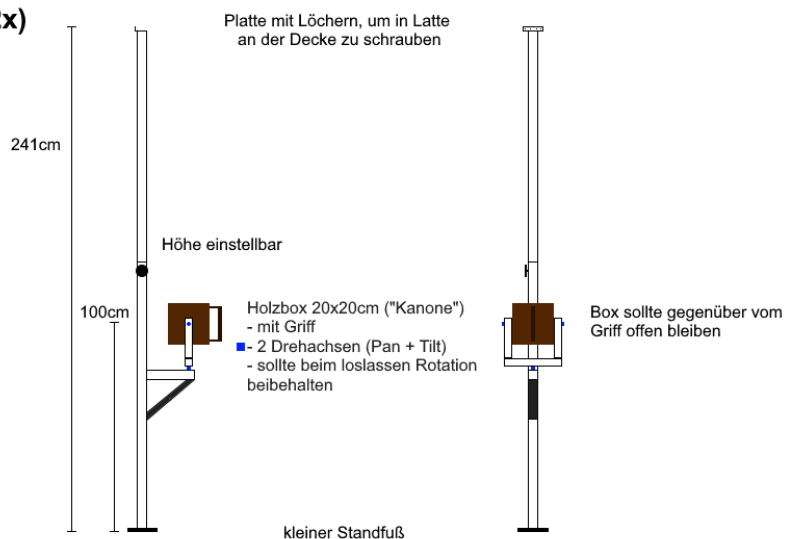
# Konzeptskizze Fall

## FALL



# Konzeptskizze Kanone

## KANONE (2x)



## Arduino-Code

```
volatile long temp, counter = 0;

void setup()
{
    Serial.begin(9600);

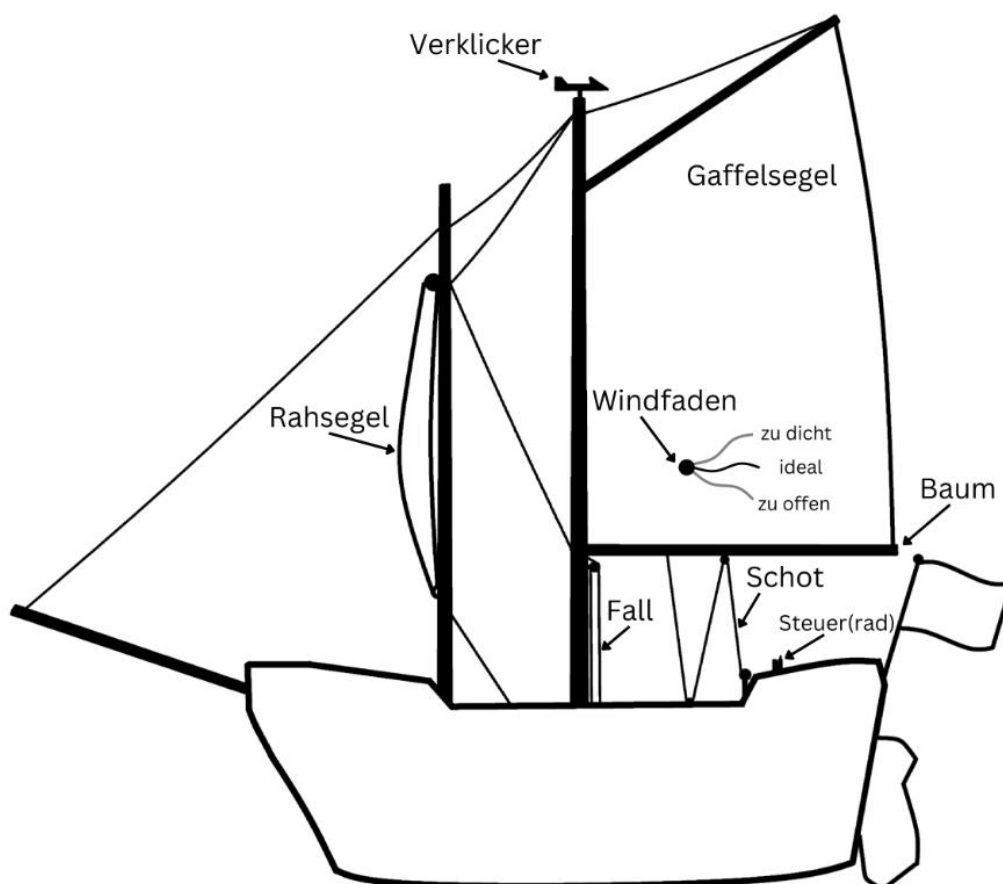
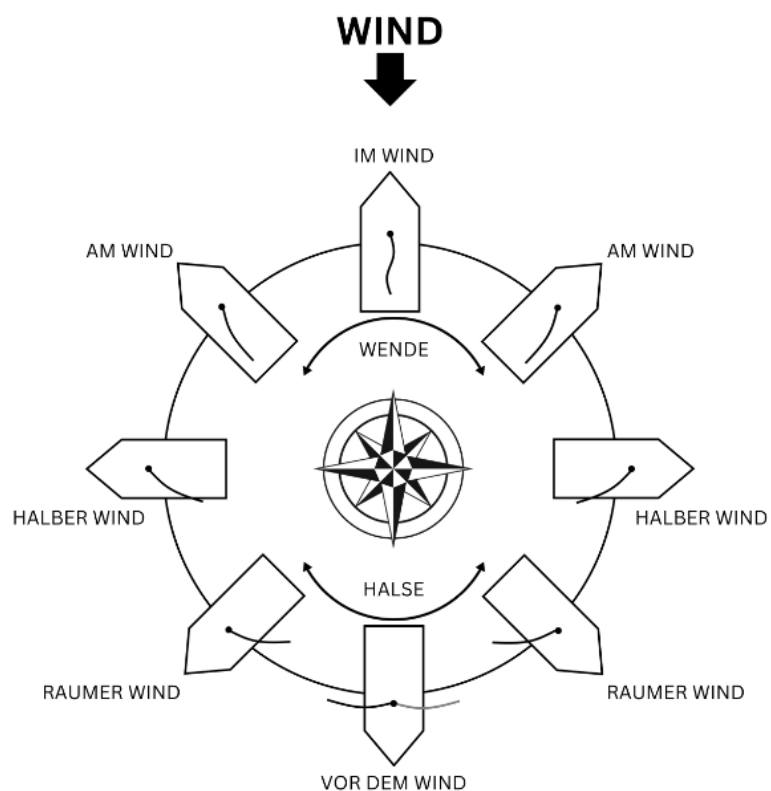
    pinMode(2, INPUT_PULLUP);
    pinMode(3, INPUT_PULLUP);

    attachInterrupt(digitalPinToInterrupt(2), ai0, RISING);
    attachInterrupt(digitalPinToInterrupt(3), ai1, RISING);
}
void loop()
{
    if (counter != temp)
    {
        Serial.print("W"); //Letters W (Wheel), S (Sheet) and H (Halyard) are used
                           //to differentiate between the three input methods.
        Serial.println(counter);
        temp = counter;
    }

    delay(100);
}

void ai0()
{
    if (digitalRead(3) == LOW)
    {
        counter++;
    }
    else
    {
        counter--;
    }
}
void ai1()
{
    if (digitalRead(2) == LOW)
    {
        counter--;
    }
    else
    {
        counter++;
    }
}
```

## Poster für Vermittlung der Segelgrundlagen



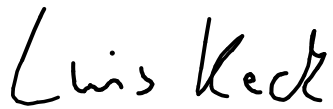
## Übersicht der Spielwelt mit Routen der Gegnerschiffe



## Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Thesis selbständig und ohne unzulässige fremde Hilfe angefertigt habe. Alle verwendeten Quellen und Hilfsmittel sind angegeben. Der Einsatz von KI-Anwendungen ist dem betreffenden Thesisteil, der Art sowie dem Umfang nach detailliert benannt.

Furtwangen, den 28.02.2023

A handwritten signature in black ink, reading 'Luis Keck'. The 'L' is large and stylized, and the 'Keck' is written in a cursive script.

Luis Keck