

# **ScribbleFight - Plattformbasiertes 2D-Brawl-Spiel mit Trainings-KI und Bilderkennung**

## **DIPLOMARBEIT**

verfasst im Rahmen der

**Reife- und Diplomprüfung**

an der

**Höheren Abteilung für Medientechnik und Informatik**

Eingereicht von:

Himmetsberger Jonas  
Rafetseder Tobias  
Weinzierl Ben

Betreuer:

Aistleitner Gerald

Projektpartner:

none

Leonding, April 2022

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die Arbeit wurde bisher in gleicher oder ähnlicher Weise keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Die vorliegende Diplomarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Leonding, April 2022

S. Schwammal & S. Schwammal

Zur Verbesserung der Lesbarkeit wurde in diesem Dokument auf eine geschlechtsneutrale Ausdrucksweise verzichtet. Alle verwendeten Formulierungen richten sich jedoch an beide Geschlechter.

# Abstract

Brief summary of our amazing work. In English. This is the only time we have to include a picture within the text. The picture should somehow represent your thesis. This is untypical for scientific work but required by the powers that are. Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.



# Zusammenfassung

Zusammenfassung unserer genialen Arbeit. Auf Deutsch.

Das ist das einzige Mal, dass eine Grafik in den Textfluss eingebunden wird. Die gewählte Grafik soll irgendwie eure Arbeit repräsentieren. Das ist ungewöhnlich für eine wissenschaftliche Arbeit aber eine Anforderung der Obrigkeit.

*Bitte auf keinen Fall mit der Zusammenfassung verwechseln, die den Abschluss der Arbeit bildet!* Suspendisse vel felis.

Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.



# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Abkürzungen . . . . .	1
1.2 Autoren der Diplomarbeit . . . . .	1
1.2.1 Himmetsberger Jonas . . . . .	1
1.2.2 Rafetseder Tobias . . . . .	1
1.2.3 Weinzierl Ben . . . . .	1
<b>2 Zieldefinition</b>	<b>2</b>
2.1 Projektanlass . . . . .	2
2.2 Ziele . . . . .	2
2.3 Aufgabenverteilung . . . . .	2
2.3.1 Web-Game und Deployment [R] . . . . .	2
2.3.2 Gamedesign und Lobby-System [B] . . . . .	3
2.3.3 Aufbereitung der Spielumgebung und Künstliche Intelligenz [H]	3
2.4 Dokumente . . . . .	3
<b>3 Umfeldanalyse</b>	<b>4</b>
3.1 Ähnliche Spiele . . . . .	4
3.1.1 Super Smash Bros (SSB) . . . . .	4
3.1.2 Brawlhalla . . . . .	4
3.1.3 Stick Fight: The Game . . . . .	5
3.2 Ist-Zustand . . . . .	5
<b>4 Technologien</b>	<b>6</b>
4.1 Java-Script [R] . . . . .	6
4.1.1 Wieso Java-Script? . . . . .	6
4.1.2 p5.js / p5.play [R] . . . . .	6
4.1.3 Node.js [R] . . . . .	8
4.1.4 Socket IO [R] . . . . .	11

4.2	Deployment [R] . . . . .	12
4.2.1	Docker [R] . . . . .	13
4.2.2	Kubernetes [R] . . . . .	14
4.3	Python [H] . . . . .	15
4.3.1	Flask [H] . . . . .	15
4.3.2	OpenCV2 [H] . . . . .	15
4.3.3	PIL [H] . . . . .	16
4.3.4	TensorFlow und Keras [H] . . . . .	16
4.3.5	OpenAI Gym [H] . . . . .	17
4.3.6	Künstliche Intelligenz allgemein [H] . . . . .	18
<b>5</b>	<b>Umsetzung</b>	<b>25</b>
5.1	Web-Game [R] . . . . .	25
5.1.1	Frontend [R] . . . . .	25
5.1.2	Server [R] . . . . .	41
5.2	Deployment [R] . . . . .	51
5.2.1	Docker-Image [R] . . . . .	51
5.2.2	Leo-Cloud [R] . . . . .	52
5.3	Map-Erkennung [H] . . . . .	54
5.3.1	Objekterkennung [H] . . . . .	56
5.3.2	Open-CV2 [H] . . . . .	62
5.3.3	Kommunikation mit der Lobby via Flask und Flask SocketIO [W]	79
5.4	KI [H] . . . . .	79
5.4.1	Verwendete Bibliotheken [H] . . . . .	81
5.4.2	Die ScribbleFight-KI [H] . . . . .	84
5.4.3	ScribbleFight Reinforcement Learning Implementierung [H] . . . . .	88
5.4.4	Evaluation der ScribbleFight KI [H] . . . . .	94
<b>6</b>	<b>Evaluation des Projektverlaufs</b>	<b>96</b>
6.1	Meilensteine . . . . .	96
6.2	Gelerntes . . . . .	96
6.3	Was würden wir anders machen? . . . . .	96
<b>Literaturverzeichnis</b>		<b>VII</b>
<b>Abbildungsverzeichnis</b>		<b>VIII</b>
		<b>IV</b>

<b>Tabellenverzeichnis</b>	<b>X</b>
<b>Quellcodeverzeichnis</b>	<b>XI</b>
<b>Anhang</b>	<b>XII</b>

# **1 Einleitung**

## **1.1 Abkürzungen**

## **1.2 Autoren der Diplomarbeit**

**1.2.1 Himmetsberger Jonas**

**1.2.2 Rafetseder Tobias**

**1.2.3 Weinzierl Ben**

## **2 Zieldefinition**

### **2.1 Projektanlass**

Die Möglichkeit seiner Kreativität freien Lauf zu lassen ist bei den meisten populären Online-Spielen sehr eingeschränkt, da man wenig Einfluss auf die Spielumgebung hat. Unsere Arbeit soll diesem Problem entgegenwirken. Der Spieler kann selbst entscheiden, wie die 2D-Spielumgebung auszusehen hat, indem er diese auf ein Blatt Papier zeichnet, welche dann via Bilderkennung als spielbare Welt aufbereitet wird.

### **2.2 Ziele**

Bis zum Abgabetermin sollen folgende Ziele erreicht werden:

- Das Spiel soll als Browserspiel funktionsfähig sein.
- Für die Benutzer soll es möglich sein ihre eigenen Kampfumgebungen zu erschaffen.
- Das Spiel soll als online-Multiplayer "Player versus PlayerSpiel funktionieren.
- Ein eigener Modus, in welchem der Spieler als Singleplayer gegen eine funktionsfähige KI antreten kann, soll umgesetzt werden.

### **2.3 Aufgabenverteilung**

Dadurch, dass die Diplomarbeit drei mitwirkende Schüler hat, wurde das Thema in drei ähnlich anspruchsvolle Teile unterteilt. Diese werden im Folgenden genauer erläutert:

#### **2.3.1 Web-Game und Deployment [R]**

Die wichtigsten Punkte, die im Bezug auf das Web-Game umzusetzen zu waren, sind:

- Am Client:

1. Die Spielphysik, also wie sich Spieler und Objekte verhalten
  2. Die Collision-Detection von Spielern mit der Umgebung und mit Objekten
  3. Die Hitregistration, falls ein Spieler von etwas getroffen wurde
  4. Ab wann ist das Spiel zu Ende, beziehungsweise wann hat jemand gewonnen
- Am Server:
    1. Übertragung von Positionen der Spieler
    2. Item-Erstellung alle Clients zum gleichen Zeitpunkt
    3. Wenn Items aufgesammelt werden, sollen diese bei allen verbundenen Spielern verschwinden
    4. Wenn eine Attacke jemanden trifft, soll diese bei allen verbundenen Spielern gelöscht werden

Das Deployment des Projekts in die Leocloud, ein Cloud-System der HTL-Leonding, ist mittels Kubernetes erfolgt.

### **2.3.2 Gamedesign und Lobby-System [B]**

### **2.3.3 Aufbereitung der Spielumgebung und Künstliche Intelligenz [H]**

Die Aufbereitung der Spielumgebung sollte folgende Funktionen erfüllen:

- Erkennung der Konturen in einer Live-View
- Aufnahme soll in brauchbare Daten umgewandelt werden

Folgende Forderungen waren an die KI gestellt:

- Die KI soll auf einem herausfordernden Niveau agieren
- Im Zuge der Forschung sollte ein Vergleich zwischen brauchbaren Lernalgorithmen gemacht werden

Im Laufe der Diplomarbeit und der damit zusammenhängenden Forschung änderte sich oft die Vorstellung darüber, wie das Endprodukt auszusehen hat.

## **2.4 Dokumente**

# **3 Umfeldanalyse**

## **3.1 Ähnliche Spiele**

Es gibt schon einige 'Player vs Player'-Brawlspiele, also Spiele, bei denen die Spieler gegeneinander kämpfen. Doch solche Spiele, die Benutzer ohne Download im Browser miteinander spielen können, und bei denen sie die Spielumgebung selber ausdenken können, gibt es noch nicht. Es gibt aber schon einige ähnliche Spiele:

### **3.1.1 Super Smash Bros (SSB)**

Super Smash Bros (SSB) sind eine Reihe von plattform-basierten Videospielen. Diese sind von Nintendo entwickelt und beinhalten die bekanntesten Charakteren des Unternehmens. Figuren, wie Super Mario oder Sonic, bekämpfen sich in einer Arena mit dem Ziel sich gegenseitig von einer Plattform zu stoßen. Was jedoch fehlt, ist die Plattformunabhängigkeit, da das Spiel nur auf Nintendo-Systemen funktioniert. Außerdem besteht eine Limitierung in der Auswahl von Spielumgebungen.

### **3.1.2 Brawlhalla**

Brawlhalla ist ein von der Firma Blue Mammoth entwickeltes 2D-Kampfspiel, und wurde für alle gängigen Betriebssysteme entwickelt. Auch wie in Scribble-Fight ist es das Ziel, den Gegner von einer Plattform zu stoßen.



Abbildung 1: Brawlhalla Spielumgebung

Ein Nachteil hierbei ist, dass sich der Benutzer vor dem Spielen einen Account erstellen und dann einen Download abschließen muss. Dazu kommt noch, dass die Spielumgebung nie beeinflusst werden kann.

#### 3.1.3 Stick Fight: The Game

In diesem Spiel kämpfen die Spieler als Strichmännchen gegeneinander, die durch eine Ragdoll-Engine gesteuert werden. Auch wie bei schon bei vorher erwähnten Spielen, kann der Benutzer aber nicht einfach plattformunabhängig im Browser gegeneinander antreten. Für die Spielekonsole Nintendo Switch zum Beispiel, gibt es das Spiel nicht. Hinzu kommt, dass auch die Umgebung nicht frei erstellt werden kann.

## 3.2 Ist-Zustand

Dadurch das es eine Diplomarbeit ist, fängt alles bei 0 an. Es gibt jedoch Frameworks, welche die Erarbeitung erleichtern. Zum Beispiel im Falle des Spiels, welches mittels Webtechnologien umgesetzt wird, kann p5.js verwendet werden, welches die Umsetzung eines Spiels erleichtert. Ein weiteres Beispiel ist die Objekterkennung für die Spielumgebung (Map). Diese wird von Open-CV übernommen.

# 4 Technologien

## 4.1 Java-Script [R]

### 4.1.1 Wieso Java-Script?

Ein Stand-alone-Programm ist ein Programm, dass selbständig arbeiten kann. Das Web-Game sollte kein Stand-alone-Programm werden, sondern im Browser erreichbar sein. Deswegen wurde Java-Script für die Programmierung am Client, sowie auch am Server ausgewählt. Der Vorteil davon ist, dass nur eine Programmiersprache benutzt werden muss, um eine Client-Server-Architektur zu bewerkstelligen. Außerdem besteht kein Aufwand, um Scribble-Fight spielen zu können - ein Browser ist alles, was benötigt wird.

### 4.1.2 p5.js / p5.play [R]

[1] p5.js ist eine open-source JavaScript-Library, die für Kreation von Spielen genutzt wird. p5.play ist eine Library für p5.js, mit der visuelle Objekte verwaltet werden können. Außerdem beinhaltet es Features wie Animation-Support, Kollisionserkennung, sowie aber auch Funktionen für Maus- und Tastatur-Interaktionen. P5.play ist für barrierefreies und simples Programmieren gedacht, nicht für performantes. Es ist keine eigene Engine, und unterstützt auch keine 3D-Spiele.

#### Einbindung

Der einfachste Weg p5.js einzubinden, ist auf ein Java-Script File online zu verweisen.

```
1      <script  
2          src="https://cdn.jsdelivr.net/npm/p5@1.4.0/lib/p5.js">  
3      </script>
```

Die p5.js Library kann aber auch unter <https://p5js.org/download/> lokal heruntergeladen werden. Dann kann einfach auf das lokale File verwiesen werden.

```
1      <script src="../p5.min.js"></script>
```

Jedoch muss das Projekt dann auf einem lokalen Server (z.B. mit Node.js umgesetzt) gehostet werden.

### Struktur eines p5.js Projekts

Die Struktur ist sehr simpel. Im Ganzen ist es nur ein `index.html` File und ein `sketch.js` File. In dem HTML File wird die p5-Library und auch das `sketch.js` File eingebunden.

```
<!DOCTYPE html>
<html lang="">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Scribble Fight</title>
  <link rel="stylesheet" href="style.css">
  <script src="lib/p5.js"></script>
  <script src="sketch.js"></script>
</head>

<body>
  <main>
  </main>
</body>

</html>
```

Abbildung 2: Aufbau index.html

So können in dem `sketch.js` File die p5.js Methoden genutzt werden. Die wichtigsten Methoden sind die Setup- und die Draw-Methode. Die Setup-Methode wird vor der Draw-Methode aufgerufen um das Spiel zu initialisieren. (Es wird zum Beispiel ein Canvas erstellt). Wenn diese abgeschlossen ist, wird die Draw-Methode 60 mal in der Sekunde aufgerufen und aktualisiert jedes mal den Bildschirm.

```

function setup() {
  createCanvas(400, 400);
}

function draw() {
  background(220);
}

```

Abbildung 3: Simples sketch.js Beispiel

### P5.js vs Processing

p5.js ähnelt sich sehr stark mit Processing, eine Programmiersprache die eine stark vereinfachte Version von Java ist. Der Unterschied liegt darin, dass Java eine Umgebung, basierend auf der Java Programmiersprache ist, während p5.js eine Bibliothek, basierend auf der Java-Script Programmiersprache ist. Processing ist dafür geeignet, lokale Applikationen zu bauen, hingegen dazu kann p5.js nur im Browser ausgeführt werden.

p5.js kann als direkter Java-Script Port für die Processing Programmiersprache definiert werden. Die Funktionen, die in Processing mit Java geschrieben werden, werden nun mit Java-Script geschrieben.

Vorteile von p5.js:

- Die Programme die mit p5.js umgesetzt werden, funktionieren in jedem modernen Browser (plattformunabhängig)
- Das Programm ist nicht nur lokal auf dem eigenen Gerät, was das Teilen sehr viel leichter macht
- Es besteht die Option, den p5.js Editor im Web zu verwenden: Überhaupt kein Aufwand, um loszuprogrammieren

Nachteile von p5.js:

- Ist kein Stand-alone-Programm, d.h. ein Browser wird benötigt

### 4.1.3 Node.js [R]

Node.js ist eine plattformübergreifende Open-Source-JavaScript-Laufzeitumgebung, mit der Besonderheit, dass sie JavaScript-Code außerhalb eines Webbrowsers ausführen kann. Ursprünglich wurde Node.js von Ryan Dahl 2009 entwickelt, einem Software-Entwickler

aus San Diego, Kalifornien. Die Laufzeitumgebung wurde darauf spezialisiert, leicht skalierbare Server zu bauen.

Vorteile von Node.js:

- Starke Performance. Node.js ist mit Fokus auf Optimierung des Datentransport und der Skalierbarkeit bei Web-Applikationen designet worden
- Da mit Java-Script entwickelt wird, wird nur eine Sprache für Client und Server benötigt
- Da Java-Script eine relativ neue Programmiersprache ist, profitiert diese auch von Verbesserungen bei Programmiersprachen-Design (im Vergleich zu anderen Web-Server-Sprachen wie PHP oder Python)
- Durch den Node-Package-Manager (NPM) kann auf sehr viele Bibliotheken zugegriffen werden
- Es ist nicht wichtig, welches Betriebssystem verwendet wird. Node.js ist kompatibel mit Microsoft Windows, macOS, Linux, Solaris, FreeBSD, OpenBSD etc.
- Hinter Node.js steht eine große Entwickler-Community, die bereit ist, in Problemfällen zu helfen

## **Node Package Manager**

Neben Node.js an sich, ist NPM (Node Package Manager) das wichtigste Werkzeug für Node.js Applikationen. Mit NPM können alle Packages, die das Projekt benötigt, gefetched werden. Es ist möglich, alle Packages einzeln zu fetchen, jedoch wird in der Praxis immer ein package.json-File benutzt. In diesem File stehen alle Dependencies für jedes JavaScript Package, das benötigt wird, sowie auch Meta-Daten zu dem Node.js Projekt. Erstellt wird dieses mit dem Befehl `npm init`. Es spielt jedoch eine Rolle, wo dieser Befehl ausgeführt wird, also muss darauf geachtet werden, dass der Befehl in dem Verzeichnis ausgeführt wird, in dem das Projekt erstellt werden soll. Die Struktur eines package.json-Files sieht wie folgt aus:

```
{
  "name": "p5_backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Rafetseder Tobias",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1",
    "socket.io": "^4.1.3"
  }
}
```

Abbildung 4: Package.json des Scribble-Fight Backends

## Express

Express ist das am meisten verbreitetste Node-Web-Framework und ist auch die Basis für andere Node-Web-Frameworks. Die Hauptverantwortung von Express ist das Bereitstellen von Server-Logik, wie zum Beispiel das Schreiben von Handlers für Requests mit unterschiedlichen Http-Verbs auf unterschiedlichen URL-Pfaden. Express wird mit dem Node-Package-Manager mit `npm install express` installiert. Befindet sich der Benutzer in einem Verzeichnis das ein package.json-File beinhaltet, bei dem Express als Dependency hinzugefügt wurde, dann reicht `npm install`.

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', function(req, res) {
  res.send('Hello World!')
});

app.listen(port, function() {
  console.log(`Example app listening on port ${port}!`)
});
```

Abbildung 5: Simpler Web Server mit Express

#### 4.1.4 Socket IO [R]

Socket.IO ist eine Library, die eine bidirektionale Echt-Zeit Verbindung zwischen Server und Client ermöglicht. Sie baut auf das WebSocket-Protokoll auf, das bedeutet es gibt neben den WebSocket-Funktionen auch noch additionale Funktionen wie zum Beispiel eine automatische Reconnection.

Um Scribble-Fight die bidirektionale Echt-Zeit Kommunikation mit dem Server bereitzustellen, wurde benutzt:

- Ein Node.js Server inklusive SocketIO Package
- Eine JavaScript Client Library von SocketIO (Es bestehen auch einige andere Client Implementationen für Sprachen wie Java, C++, Python, etc.)

Socket.IO funktioniert so, dass der Client, falls möglich, eine SocketIO-Verbindung mit dem Server herstellt. Ist keine Verbindung möglich, setzt der Client einen HTTP-long polling-Request ab.

```
const socket = io("http://localhost:3000");

socket.on("connect", () => {
    // Entweder mit send()
    socket.send("Hello!");

    // oder mit emit und eigenen Event Namen
    socket.emit("greetings", "Hello my friend!");
});

// Umgehen mit dem Event, das mit socket.send() geschickt worden ist
socket.on("message", data => {
    console.log(data);
});

// Umgehen mit dem Event, das mit socket.emit() geschickt worden ist
socket.on("greetings", data => {
    console.log(data);
});
```

Abbildung 6: Socket.IO Client Beispiel

Damit der Server die Verbindung annehmen kann, müssen folgende Kriterien erfüllt sein:

- Der Browser unterstützt WebSocket
- Die Verbindung wird nicht von Elementen wie Firewall gestört

Die API ist auf der Server-Seite dem Client sehr ähnlich, es existiert auch wieder ein socket Objekt, welches von der EventEmitter Klasse von Node.js erbt.

```
const io = require("socket.io")(3000);

io.on("connection", socket => {
    // Entweder mit send()
    socket.send("Hello!");

    // oder mit emit() und eigenen Event Namen
    socket.emit("greetings", "Hello from the Server");

    // Umgehen mit dem Event, das mit socket.send() geschickt worden ist
    socket.on("message", (data) => {
        console.log(data);
    });

    // Umgehen mit dem Event, das mit socket.emit() geschickt worden ist
    socket.on("greetings", data => {
        console.log(data);
    });
});
```

Abbildung 7: Socket.IO Server Beispiel

### Unterschied Socket.IO zu WebSocket

Socket.IO ist keine WebSocket Implementation. Auch wenn Socket.IO WebSocket als Transportmittel benutzt, werden bei jedem Paket zusätzliche Metadaten angehängt. Das ist auch der Grund, warum ein Socket.IO Client keine Verbindung mit einem schlichten WebSocket Server herstellen kann, und umgekehrt. Socket.IO ist also ein Wrapper rund um die WebSocket API.

## 4.2 Deployment [R]

Das Scribble-Fight Browser-Game wurde in die Leocloud, ein Cloud-System der HTL-Leonding, unter <https://student.cloud.htl-leonding.ac.at/t.rafetseder/scribble-fight/> deployed. Zuerst wurde mithilfe von der Docker-Technologie ein Docker-Image erstellt und auf die Leocloud hochgeladen. Das Deployment wurde dann mithilfe von Kubernetes umgesetzt. Was Docker und Kubernetes genau ist, folgt in den nächsten Sktionen.

### 4.2.1 Docker [R]

Die Software Docker ist eine Technologie zum Containerisieren von Prozessen, die dann unabhängig voneinander und isoliert ausgeführt werden können. Diese isolierte Prozesse werden Container genannt. Durch die Unabhängigkeit, die dadurch entsteht, wird die Infrastruktur besser genutzt und auch die Sicherheit bewahrt, die sich aus der Arbeit mit voneinander getrennten System ergibt. Docker arbeitet mit einem Image-basierten Bereitstellungsmodell. Dieses wird gerne bei Containertools verwendet, da Applikationen mit all deren Dependencies, egal in welcher Umgebung, genutzt werden können.

- Wenn der Benutzer einmal seine containerisierte Applikation getestet hat, kann er sich sicher sein, dass die Applikation auf jeder anderen Umgebung, auf dem Docker installiert ist, auch funktioniert
- Alle Docker Container sind komplett voneinander unabhängig
- Falls Skalierung notwendig ist, können schnell neue Container erstellt werden
- Im Gegensatz zu virtuellen Maschinen beinhalten Container keine eigenen Betriebssysteme, deshalb können sie schneller erstellt und auch schneller gestartet werden

Wichtige Begriffe, im Zusammenhang mit Docker:

- Image: Speicherabbild eines Containers
- Container: aktive Instanz eines Images
- Dockerfile: eine Textdatei, die den Aufbau des Images beschreibt
- Registry: Unter Registry wird eine Ansammlung gleicher Images mit verschiedenen Tags verstanden, meistens Versionen

### Docker Architektur

Die Docker Architektur ist eine Server-Client Architektur. Der Docker Client kommuniziert mit dem Docker Daemon (Ein Hintergrundsprozess), der dann Docker Container z.B. baut und ausführt. Dieser Docker Daemon kann lokal installiert sein, aber der Client kann sich auch mit einem Daemon Remote verbinden.

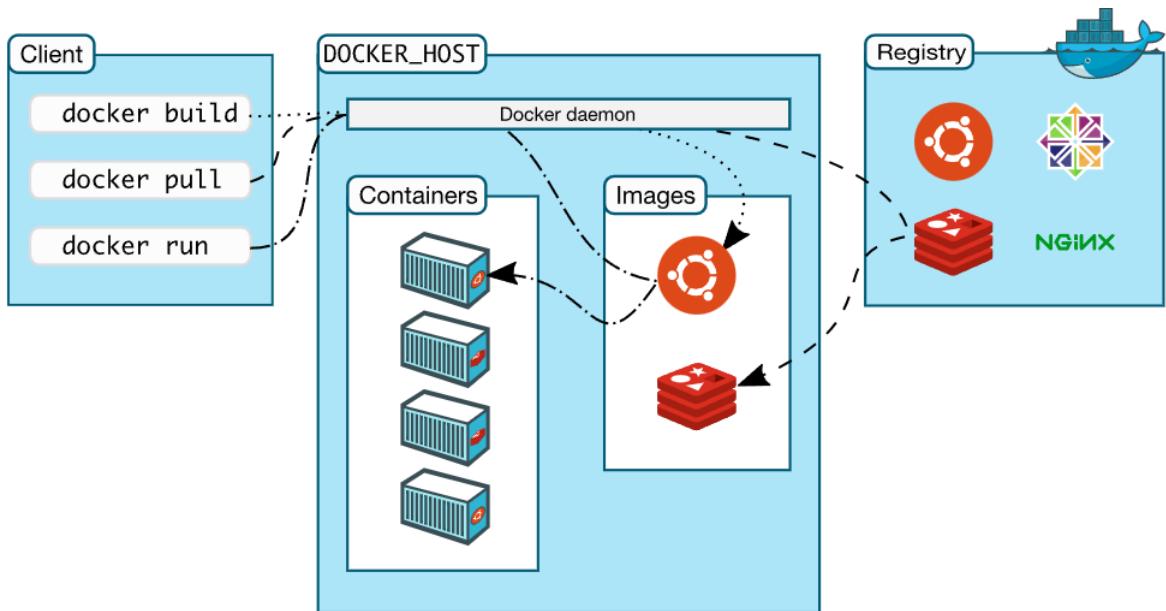


Abbildung 8: Veranschaulichung Docker Architektur

### 4.2.2 Kubernetes [R]

Kubernetes ist eine Open-Source-Plattform, die dafür genutzt wird, containerisierte Applikationen und Services zu verwalten (z.B. Computer-, Netzwerk und Speicherinfrastruktur von Containern). Das Kubernetes-Projekt ist 2014 als Open-Source Projekt in die Welt gerufen worden.

Kubernetes hat mehrere Funktionen, zum Beispiel ist Kubernetes:

- eine Containerplattform
- eine Microservices-Plattform
- eine Cloud-Plattform

In dieser Diplomarbeit wird Kubernetes benutzt, um ein mit Docker gebautes Images des Web-Servers und des Clients auf ein Cloudsystem zu deployen. Näheres zur Umsetzung wird in Kapitel TODO beschrieben

## 4.3 Python [H]

Diese Programmiersprache, welche nach der Britischen comedy Serie “Monty Python” benannt ist, fand in unserer Arbeit zwei Haupteinsatzgebiete.

Erstens, zur Erkennung des Blatt Papiers und der Umwandlung der Zeichnung in eine spielbare Map und zweitens um die Künstliche Intelligenz zu erschaffen.

Die kostenlosen Bibliotheken, welche wir dabei in Verwendung haben, werden im folgenden gelistet und näher erklärt.

### 4.3.1 Flask [H]

Flask ist das am wohl häufigste verwendete Python Web-Framework. Somit gibt es viele Tutorials, Tools und Bibliotheken. Diese sind sehr gut bis gut dokumentiert und teilweise geprüft. Aus diesen Gründen haben wir den Teil der Bild- und Maperkennung, welche als Webanwendung funktioniert, mittels Flask umgesetzt. In Kombination mit Flask-SocketIO werden Bilder von der Webcam in Echtzeit direkt an den Server geschickt, welcher dann via OpenCV2 Informationen aus dem Bild generiert. Genau wie bei SocketIO in JavaScript, agiert Flask-SocketIO als bidirektionale Kommunikation zwischen Server und Client. Die extrem geringe Latenzzeit, welche dabei auftritt, ist wichtig um eine flüssige Verarbeitung der Bilder zu gewährleisten.

### 4.3.2 OpenCV2 [H]

Open “Computer Vision” (CV) wurde in unserem Kontext als Python Bibliothek verwendet. OpenCV verfügt über eine breitgefächerte Auswahl an Bildverarbeitungs-Algorithmen. Folgende wurden bei der Maperkennung eingesetzt:

- Resizing
- Farbraumkonvertierung
- Weichzeichnung
  - Median-Blur
  - Gaussian-Blur
- adaptive Schwellenwertbildung von Pixelwerten
- Konvertierung eines Zahlen Arrays in eine “.png” datei

Auf die Funktionsweise dieser Algorithmen wird im Folgenden (5.3.2) genauer eingegangen.

Um zum Beispiel ein beliebiges Bild unscharf zu zeichnen würde man so vorgehen:

Listing 1: OpenCV Demo

```

1      # Dieses kurze Programm soll ein Bild in Python mittels Open Computer Vision
2      # weichzeichnen
3
4      # Importieren der gebrauchten Bibliothek
5      import cv2
6      import numpy
7
8      # Bild einlesen
9      source = cv2.imread('./Pfad/zum/Bild.png', cv2.IMREAD_UNCHANGED)
10
11     # Das Quell-Bild wird nun unscharf gezeichnet
12     destination = cv2.GaussianBlur(source,(5,5),cv2.BORDER_DEFAULT)
13
14     # Anzeigen von dem Quellbild und dem bearbeiteten Bild
15     cv2.imshow('Weichzeichnung',numpy.hstack((source, destination)))
16
17     # warten, bis eine Taste gedrueckt wurde
18     cv2.waitKey(0)
19
20     # Alle Fenster, welche die Bilder anzeigen, werden geschlossen
21     cv2.destroyAllWindows()

```

### 4.3.3 PIL [H]

PIL (Python Image Library) ist, wie Flask und OpenCV2, eine kostenlose Zusatzbibliothek für Python. PIL wird verwendet um Bilder zu speichern und zu pixel zu manipulieren. Dabei unterstützt PIL diese Dateiformate: PPM, PNG, JPEG, GIF, TIFF, und BMP. Pixel manipulation bedeutet, dass jeder Pixel auf einem Input Bild angepasst werden kann. Beispiele hierfür sind zum Beispiel das Aufhellen oder Abdunkeln von Bildern oder das Anpassen der Sättigung. Auch Kontrast- oder Schärfeeinstellungen können mit Pixelmanipulation erzielt werden. Dafür werden meistens Matrizen und/oder Formeln pro Pixel verwendet um deren Farbwerte anzupassen

Listing 2: PIL Demo

```

1      # verwendete Klassen der Bibliothek einbinden
2      from PIL import Image, ImageFilter
3
4      source = Image.open("file.ppm") # Load an image from the file system.
5      destination = source.filter(ImageFilter.BLUR) # Blur the image.
6
7      # Display both images.
8      original_image.show()
9      destination.show()

```

### 4.3.4 TensorFlow und Keras [H]

TensorFlow ist eine Python Bibliothek, welche beim erstellen von Projekten, welche maschinelles Lernen in irgend einer Art und Weise eingebunden haben, extrem unter-

stütz. Wie der Name schon vermuten lässt, basiert TensorFlow auf zwei Grundlagen: Tensoren und Graphen (Flow vom Wort dataflow).

Tensoren sind besser bekannt als Skalare, Vektoren oder Matrizen. Tensoren sind also null-, ein- oder mehrdimensionale Daten-Tupel. TensorFlow bietet alles von der effizienten Ausführung von Befehlen auf der CPU, oder GPU, über der Skalierung von Berechnungen auf viele Endgeräte bis hin zur Visualisierung der gelernten Daten mittels dem sogenannten “TensorBoard”. TensorFlow ist also ein sehr mächtiges Framework, das viele Möglichkeiten bietet, künstliche Intelligenzen zu trainieren und analysieren. Jedoch ist die Benutzerfreundlichkeit von TensorFlow sehr eingeschränkt. Viele Entwickler empfanden es als ungeeignet für schnelles Prototyping und verwendeten daher das auf TensorFlow basierende Keras.

Keras verwendet standardmäßig die GPU zum ausführen von Code und ist somit um einiges schneller als TensorFlow. In dieser Diplomarbeit wurden zwei der von TensorFlow (Stable-Baselines3) bereits vorgefertigten Algorithmen, namens “A2C” und “PPO”, verwendet um die KI zu trainieren. Auf die Auswertung wird im Kapitel 5.4.2 näher eingegangen. Die Logik hinter der Künstlichen Intelligenz wurde mittels OpenAI Gym implementiert.

### 4.3.5 OpenAI Gym [H]

OpenAI Gym ist ein Toolkit, mit welchem das Erlernen und Erstellen von reinforcement learning Algorithmen sehr leicht fällt. Da viele Menschen nicht wissen, was “reinforcement learning” ist, wird es im folgenden kurz erläutert.

#### Reinforcement Learning [H]

Reinforcement learning bedeutet auf deutsch so viel wie bestärkendes Lernen oder verstärkendes Lernen. Diese Art und Weise zu lernen ist der, wie ein Lebewesen mit einem biologischen Gehirn lernt, am ähnlichsten. Dabei basiert es auf folgendem Konzept: Ein Agent, welcher etwas erlernen soll, wird in eine ihm unbekannte Umwelt gesetzt. Dieser kennt nicht mehr als seinen eigenen Zustand und welche Aktionen er ausführen kann. Bewegt sich diese Entität nun in der Umwelt, so passieren ihm Dinge. Diese können nach einer Observation entweder zu einer negative (-) oder einer positive (+) Belohnung führen. Das einzige Ziel des Agenten besteht nun darin sein Tun so anzupassen, dass er so viel und so effizient wie möglich an diese Belohnung gelangt. Folgende Grafik erläutert das Beispiel an einem Hund, welcher lernen soll einen Stecken zu apportieren.

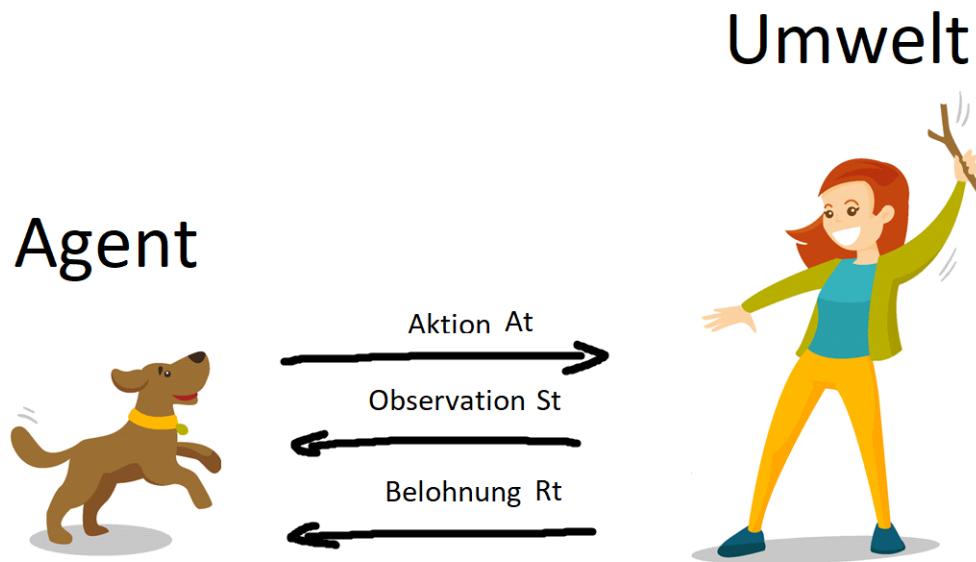


Abbildung 9: Veranschaulichung bestärkendes Lernen

OpenAI Gym hat also einen mehr oder weniger vorgegebenen Bauplan, und vorgegebene Regeln, nach welchen man so eine Künstliche Intelligenz aufbauen muss.

Weitere, weit verbreitete Formen von machine learning sind supervised und unsupervised learning.

#### 4.3.6 Künstliche Intelligenz allgemein [H]

Künstliche Intelligenz funktioniert im Grunde genommen wie das menschliche Gehirn. Es basiert genauso auf Neuronen und deren Axonen, Dendriten und Terminale. Obwohl es verschiedene Arten von Neuronen gibt, haben sie alle gemeinsam, dass sie einen elektrischen Impuls als Reaktion auf einen Input aussenden. Abhängig von diesem Input ist durch das Neuron, welches das chemische Signal verarbeitet, der Output.



Abbildung 10: Neuron

Genauso funktioniert auch ein Neuronales Netz. Auf eine Eingabe folgt über eine Verarbeitung der Eingabe eine Ausgabe. Und genau wie bei einem Menschen, welcher etwas Neues kennenlernt, weiß auch das Neuronale Netz nicht, was dies Korrekte Antwort auf ein Problem ist. Es muss sich also herantasten an die Lösung.

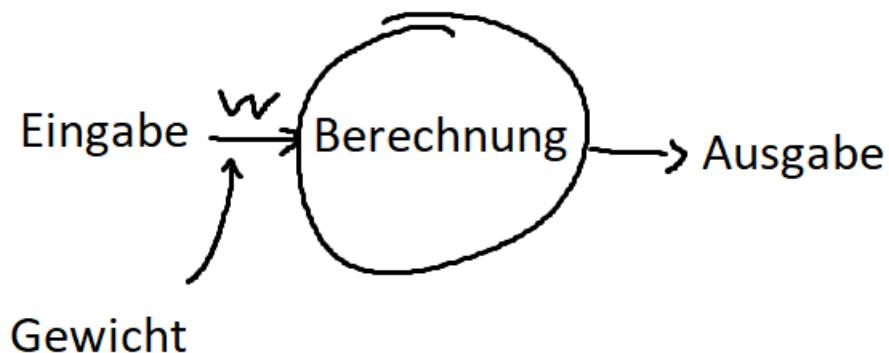


Abbildung 11: Eingabe → Berechnung → Ausgabe

Ein einfaches mathematisches Beispiel hierfür wäre, wenn man einen einfachen Faktor mit zwei Nachkommastellen herausfinden möchte. Beispiel: Euro in USD umrechnen. Hierfür ist ein Faktor kleiner 1 nötig. Bevor der ganze Prozess des Lernens startet, muss

man mit einer Eingabe starten. Dies ist zum Beispiel die Zahl 1. Somit erkläre ich der künstlichen Intelligenz im übertragenen Sinne: "Bitte errechne mir wie viele USD in meiner Hand liegen.". Es wurde also ein Input für die KI gefunden. Dieser Input wird über eine nicht ganz zufällig gewählte Gewichtung in eine Berechnung umgewandelt. Nehmen wir an, diese Gewichtung sei ein Faktor  $<1$  also 0,99. "Nicht ganz zufällig gewählt", weil der Anwender, welcher die KI schreibt im Vorhinein schon wusste, dass ein USD weniger Wert ist, als ein Euro; der genaue Wert war jedoch unbekannt. Nach dieser Berechnung kommt ein Wert raus – nämlich 0,99 – welcher falsch ist. Damit die künstliche Intelligenz jedoch weiß, ob sie richtig oder falsch rechnet, muss man ihr ein Feedback geben. Das heißt, dass der errechnete Wert und die Abweichung vom tatsächlichen Ergebnis (wenn dieses bekannt ist) zurückgegeben wird. In den meisten Fällen ist das Ergebnis bei präparierten Daten bereits bekannt, da diese als Trainingsdaten agieren. Wenn die künstliche Intelligenz jedoch selbst Entscheidungen vorhersagen muss, muss diese bereits mit solchen Daten trainiert worden sein, um ein akkurate Ergebnis liefern zu können. In dieser Phase lernt sie jedoch auch nicht mehr dazu. Sind die Ergebnisse unbekannt kann zum Beispiel das zuvor erklärte Reinforcement Learning als Lernstrategie herangezogen werden. Diese Abweichung wird nun mit einer weiteren Berechnung rückpropagiert und die Gewichte werden aktualisiert. Dieses Prozedere (Eingabe → Berechnung → Ausgabe → Fehler → Fehler rückpropagieren → Gewichte anpassen) wird so oft mit weiteren Trainingsdaten wiederholt, bis die KI einen minimalen Fehler (Differenz zwischen dem tatsächlichen Ergebnis und der Vorhersage) erreicht. Allerdings können hier einige Faktoren zusätzliche beeinflusst werden, bevor die KI tatsächlich zu lernen beginnt, um ein maximal genaues Ergebnis zu erzielen. Beispielsweise beträgt der Wechselkurs von Euro zu USD 0,89. Als ein USD ist nur 0,89 so viel wert wie ein Euro. Wenn man es der KI ermöglicht sich nur in zehntel-Schritte an die Lösung anzupassen, so passiert folgendes: Die KI wird das tatsächliche Ergebnis von 0,89 nie erreichen. Die von der KI errechnete Lösung beträgt entweder 0,9 oder 0,8. Wenn man diesen Anpassungswert jedoch kleiner ansetzt, auf ein Hundertstel zum Beispiel, so wird diese den Wert zwar langsamer erreichen, jedoch wird er genau dem Ziel entsprechen. Dieser Wert darf jedoch auch nie zu klein gewählt werden. Schaut man sich dies an einem etwas komplexeren Beispiel an, so ist klar zu erkennen, dass in der folgenden Kurve der tatsächliche minimale Fehler nie erreicht wird. Um dies zu vermeiden, gibt es verschiedene Methoden. Als Exempel kann eine Art Fehler-Abtastungs-Beschleunigung herangezogen werden. Hier wird der Faktor, um welchen die Abweichung korrigiert wird in einer Art Beschleunigung

angepasst. Man muss sich also den Aktuellen wert wie einen Ball vorstellen, welcher über einen Berg hinunterrollt.



Abbildung 12: Fehlerkurve

Durch das Momentum kann der Ball nachfolgende Hindernisse überwinden. Die Gefahr dabei besteht jedoch, dass wenn der Ball nicht noch einmal angeschubst wird in einem höheren Tal zum Stehen kommt, weil das Momentum zum Zurückkommen fehlt. Es kann aber auch sein, dass der Ball nie genug Momentum hatte, da er zu wenig stark losbewegt wurde. All das und VIELES mehr fällt unter dem Begriff "Hyperparameter Tuning". Es muss also schon zu Beginn, bevor das Neuronale Netz überhaupt zu lernen beginnt, darauf geachtet werden, dass die Parameter stimmen, damit das Ergebnis einer perfekten Lösung am ähnlichsten ist. Auch bei der Auswahl der Trainingsdaten ist enorme Vorsicht geboten. So ist es in einem amerikanischen Experiment dazu gekommen, dass eine Künstliche Intelligenz, rassistisch wurde. Diese Software sollte Richtern dabei helfen Häftlinge nach ihrer Entlassung zu beurteilen, wie hoch die Wahrscheinlichkeit sei, dass diese eine Wiederholungsstrafat begehen. Dabei kam heraus, dass dunkelhäutige Menschen weitaus gefährlicher eingestuft wurden als alle anderen. Das passierte, nicht weil diese tatsächlich eine höhere Gefahr darstellen, sondern weil in den USA dunkel-häutige Menschen öfter verhaftet werden. So stimmten auch die Probationen in den für die KI vorliegenden Trainingsdatensätzen nicht. Daraufhin meinte ein beteiligter Journalist: „Künstliche Intelligenz hat keine Meinung oder ein Bewusstsein, sondern handelt nach dem, was wir ihr vorgeben. Diese Daten und Informationen sind oft ein Spiegel der Gesellschaft und reproduzieren so auch Vorurteile, zum Beispiel durch Über-

und Unterrepräsentation“. Zitat **Tobias Matzner**

Und dass man mit Mathematik auch tatsächlich Spiele gewinnen kann, beziehungsweise, dass eine KI, welche immer wieder eine Gewichtung einer Eingabe aktualisiert, auch wirklich schlauer wird, wird im folgenden Beispiel erklärt. Hier geht es um ein Spiel in welchem man mit Mathematik eine Gewinnchance erhöht. Dies wird erzielt, indem man wiederholt Berechnungen ausführt und Gewichte aktualisiert. Somit wird die Gewinnchance in dem Spiel erhöht.

Der Name des Spiels ist “Hexapawn” und wurde von Martin Gardner entwickelt, welcher im Zweiten Weltkrieg half den “Nazicode” zu knacken.

Das Spiel ist wie folgt aufgebaut: ein 3x3 Schachbrett bildet den Untergrund des Spiels. Auf den jeweils gegenüberliegenden Seiten befinden sich 3 Schachfiguren. Deutlich wird dies in der nachfolgenden Abbildung.

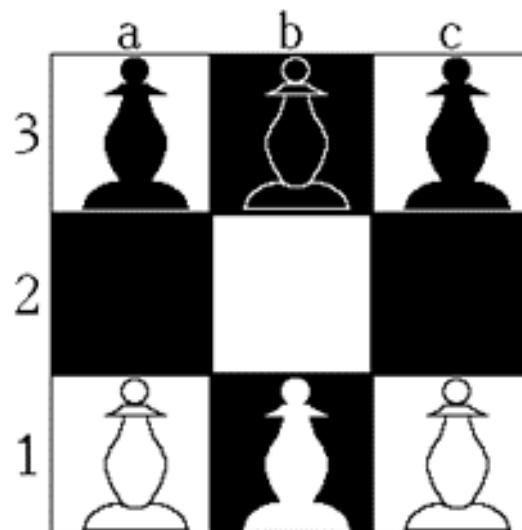


Abbildung 13: Hexapawn Spielumgebung

Diese drei Figuren sind gleichwertig und dürfen sich Verhalten wie ein “Bauer” im Spiel Schach (also nur nach vor oder zum Schmeißen nach links und rechts). Martin Gardner setzt dabei die Regel fest, dass der Mensch, welche Seite er auch immer spielt, den ersten Zug macht. Somit ist der Mensch immer in den ungeraden Zügen dran (1, 3, 5, 7) und der Computer/die künstliche Intelligenz/die Mathematik immer in den geraden Zügen (2, 4, 6) dran. Nach acht Zügen gibt es garantiert immer einen Gewinner. Das Ziel des Spiels ist es, dass man alle gegnerischen Figuren schmeißt.

Um einen solchen “Hexapawn”-Computer zu bauen braucht man vierundzwanzig Streichholzschahteln. Auf jedem dieser Streichholzschahteln sind alle möglichen “Hexapawn”-Züge aufgezeichnet, und wie auf diese reagiert werden kann. Dies wird in der nachstehenden Grafik veranschaulicht. Seitlich haben diese Schachteln eine Öffnung.



Abbildung 14: Hexapawn mögliche Züge

Wähle der Mensch und Gegner der Maschine den ersten Zug, so ist dieser garantiert auf einen der beiden ersten Abbildungen zu sehen. In den Zündholzschachteln befinden sich farbige Kugeln. Schüttelt man diese nun, und lässt eine Kugel aus der Öffnung seitlich fallen, so wählt man den Zug, bei welchem die Pfeilfarbe der Kugelfarbe entspricht. So spielt man das Spiel zu Ende, bis es nach dem achten Zug einen garantierten Gewinner gibt und zieht dann ein Resultat. Das heißt, wenn die KI gewonnen hat, kann man die Gewinnchance bei einem nächsten Spiel erhöhen. Dies erreicht man, indem man alle farbigen Kugeln, welche aus der Streichholzschachtel fielen, doppelt zurücklegt. Wenn es also in der ersten Schachtel grün, rot und blau gibt und grün als erster Zug eines Spiels gewählt wurde, welches zu einem Sieg führte, so gibt man in die erste Schachtel eine zweite grüne Kugel. Somit wird die Wahrscheinlichkeit, dass man bei einem nächsten Spiel einen Zug wählt, welcher schon einmal zu einem Sieg führte, erhöht. Ähnlich kann man dies auch bei Zügen machen, welche dazu führten, das Spiel zu verlieren. Hier kann man die Wahrscheinlichkeit zu verlieren verringern, indem man die Farbige Kugel aus dem Spiel entnimmt.

Und so hat Martin Gardner bewiesen, dass man Spiele mit Mathematik gewinnen kann.

Auf diesem Prinzip basieren auch viele andere künstliche Intelligenzen. Natürlich können in dem Spiel “Hexapawn” binnen weniger Augenblicke alle möglichen Kombinationen errechnet werden und somit eine perfekte Spielstrategie entwickelt werden. Anders ist dies bei Spielen wie “Schach” oder “GO”. Bei diesen Spielen gibt es unzählige Kombinationsmöglichkeiten, bei welchen es selbst für eine Maschine nahezu unmöglich ist, alle Spielkombinationen auszuprobieren. Jedoch ist es für einen Computer möglich in kürzester Zeit extrem viele Berechnungen durchzuführen. Ein tage-/monate-/jahrelanges lernen in einer Geschwindigkeit, welche für Menschen unantastbar ist, führt für die KI also auch zu einer immer höheren Gewinnchance für jedes Spiel. Dadurch ist es für die sogenannte “AlphaGo”-KI möglich gewesen selbst die besten Spieler der Welt im Spiel “GO” zu besiegen. Durch die vielen Kombinationsmöglichkeiten ist es trotzdem nicht gegeben, dass die KI pro Zug die perfekte Auswahl trifft. Jedoch werden immer Züge gewählt, welche in bisherigen Spielen zu der höchsten Gewinnchance führten. Somit ist es also auch für eine Maschine möglich nicht deterministische Ereignisse mit einer gewissen Wahrscheinlichkeit, welche oft der realen Zukunft entspricht, vorherzusagen (Beispiel: Wettervorhersage).

# 5 Umsetzung

## 5.1 Web-Game [R]

### 5.1.1 Frontend [R]

Für die Umsetzung des Frontends wird p5.js beziehungsweise p5.play.js verwendet. Eine detaillierte Beschreibung zu diesen JavaScript-Bibliotheken und wie sie in ein Projekt eingebunden werden kann, wird in Kapitel 4.1.2 genau beschrieben. p5.play.js basiert auf einer Sprite Klasse. Diese Sprite-Klasse hat einige praktische vordefinierte Funktionen für zum Beispiel Collisiondetection oder Animation-Support. Um einen Sprite zu erstellen, wird einfach die Funktion `createSprite()` aufgerufen.

```
1      function setup() {
2          createCanvas(1920,1080);
3          // create a sprite
4          createSprite(50,50,30,30);
5      }
6
7      function draw() {
8          // draw all the sprites added to the sketch so far
9          // the positions will be updated automatically at every cycle
10         drawSprites();
11     }
```

Es ist zu beachten, dass die ersten 2 Parameter der Funktion jeweils die Position am Bildschirm in Pixel angeben, und die letzten 2 die Breite und die Höhe definieren. Das Ergebnis:



Abbildung 15: Einfacher Sprite

Es ist noch nicht viel zu sehen, nur ein simpler Sprite, der default-mäßig ein einfaches Rechteck mit zufälliger Farbe auf einer Position erschienen ist. Für den Spieler, der in

dem Canvas angezeigt werden soll, wird diese Sprite-Klasse noch um ein paar Attribute erweitert:

```
class Player {
    constructor(sprite) {
        this.sprite = sprite;
        this.id = null;
        this.knockback = 1;
        this.death = 0;
        this.kills = 0;
        this.dmgDealt = 0;
        this.item = [];
        this.damagedBy = null;
        this.direction = "";
    }
}
```

Abbildung 16: Player-Klasse

Zum Erstellen der Player-Klasse wird zwar ein Sprite benötigt, damit der Player am Bildschirm angezeigt wird, aber Attribute wie:

- id: Zur eindeutigen Identifizierung
- knockback: Wert, der erhöht wird, desto öfter der Spieler getroffen wird; desto höher der Wert, desto weiter wird der Spieler von Projektilen weggestoßen
- death: Anzahl, wie oft der Spieler gestorben ist
- kills: Anzahl an Kills, die der Spieler gemacht hat (Abschüsse, die dazu geführt haben, dass der Gegner hinuntergefallen ist)
- dmgDealt: Anzahl, wie oft der Spieler jemanden getroffen hat
- item: Array von Items, die der Spieler gerade besitzt
- direction: String, der die Richtung angibt, in die der Spieler gerade schaut (links oder rechts)

Während des Spiels werden auf der Map Items gespawned. Welcher Algorithmus dahinter steckt, wird im Kapitel 5.1.1 genauer beschrieben. Es wird zwischen 5 unterschiedlichen Items entschieden. Eine Item-Klasse hat ähnlich wie die Player-Klasse als Hauptbestandteil einen Sprite, doch auch hier werden noch weitere Attribute benötigt.

```

class Item {
    constructor(type) {
        this.type = type;
        this.dropped = false;
        switch (this.type) {
            case "bomb":
                this.ammo = 5;
                this.sprite = undefined;
                break;
            case "black_hole":
                this.ammo = 5;
                this.sprite = undefined;
                break;
            case "piano":
                this.ammo = 10;
                this.sprite = undefined;
                break;
            case "mine":
                this.ammo = 3;
                this.sprite = [];
        }
    }
}

```

Abbildung 17: Item-Klasse

- type: Gibt den Typ des Items an
- dropped: Gibt an, ob das Item irgendwo auf der Umgebung gelandet ist
- ammo: Anzahl, wie oft der Spieler das Item benutzen kann

Je nachdem, welchen Typ das Item bekommen hat, wird auch die Anzahl wie oft der Spieler es benutzen kann, verändert. Wurde es zum Beispiel zum Type "bomb", kann der Spieler 5 Mal eine Bombe werfen.

Bei dem Typ "mine", also einer Mine, kann der Spieler mehrere Minen gleichzeitig legen, deshalb ist das Sprite-Attribut auch ein Array. Bei den anderen Items kann immer nur ein Sprite davon existieren.

## Die Items

Grundsätzlich gibt es 5 unterschiedliche Items in Scribble-Fight. Je nachdem, welches Item der Spieler aufgesammelt hat, kann er oder sie verschiedene Fähigkeiten aktivieren. Für jedes Item existiert eine physics-Methode, die in der Draw-Funktion aufgerufen wird.

## Keyboard-Access

Die Items werden alle durch Tasten auf der Tastatur ausgelöst. Durch p5.js kann sehr leicht auf die Tastatur zugriffen werden. Mit der p5.js Methode `<keyWentDown(Key)>` wird überprüft, ob eine Taste gerade gedrückt wurde. (Als Parameter wird der ASCII-Code der gewünschten Taste übergeben)

Listing 3: Keyboard-Access

```

1   // E
2   if (keyWentDown(69)) {
3     bombAttack();
4   }
5   // Q
6   if (keyWentDown(81)) {
7     blackHoleAttack();
8   }
9   // R
10  if (keyWentDown(82)) {
11    pianoTime();
12  }
13  // C
14  if (keyWentDown(67)) {
15    placeMine();
16  }
17  // F
18  if (keyWentDown(70)) {
19    makeMeSmall();
20 }
```

## Bomb

*Das Bomben-Item wird mit der Taste <E> aktiviert.*

Durch p5.play.js kann die vertikale Geschwindigkeit eines Sprites mit `<sprite.velocity.y>` verändert werden. Dadurch kann eine Gravitation simuliert werden. Außerdem können mit der Methode `<sprite.bounce(otherSprite)>` Sprites an anderen Sprites oder Gruppen von Sprites 'abspringen'. Dies wird benutzt, damit die Bombe an der Umgebung abprallt. Danach wird mit `<sprite.overlap(myPlayer.sprite)>` überprüft, ob eine Bombe mit meinem Spieler-Sprite kollidiert. Falls ja, wird die Bombe mit `<sprite.remove()>` entfernt und mein Spieler wird weggestoßen. Falls nein, wird noch überprüft, ob sich die Bombe außerhalb des Bildschirms befindet, und falls dies zutrifft, wird auch in dem Fall die Bombe entfernt.

Listing 4: Bomb Item Physics

```

1   // verringern der vertikalen Geschwindigkeit
2   bomb.velocity.y -= GRAVITY;
3   // bombe prallt an der Umgebung ab
4   bomb.bounce(environment);
5
6   // kollidert die Bombe mit meinem Player
7   if(bomb.overlapSprite(myPlayer.sprite)) {
8     // my Player gets knocked back
9     myPlayer.sprite.getsThrownAway();
10    // bomb gets deleted
11    bomb.remove();
12  } else if(bombIsOutsideMonitor()) {
13    // bomb gets deleted
```

```

14         bomb.remove();
15     }

```

## Black Hole

Das Black-Hole-Item wird mit der Taste <Q> aktiviert.

Wie bei dem Bomben-Item wird auch bei dem Black-Hole-Item Gravitation simuliert. Jedoch nur für eine kurze Zeit, bis das Item in der Luft stehen bleibt und in einem Radius alle Spieler, die sich in diesem Radius befinden, anzieht, und diese auch alle Fähigkeiten nimmt. Um zu überprüfen, wie lange das Item schon existiert, stellt p5.play.js das <life>-Attribut zur Verfügung. Dieser Wert ist ein Countdown, der sich bei jedem Draw-Zyklus um 1 verringert, bis sich das Item dann bei dem Wert 0 selbst löscht. Damit das Item funktionieren kann, muss es andere Sprites anziehen können. Dazu wurde die attraction-Funktion von p5.play.js (mit leichten Veränderungen) benutzt:

Listing 5: Attraction

```

1  // attraction
2  if (myPlayer.sprite.overlap(b)) {
3      noGravity = true;
4      var angle = atan2(myPlayer.sprite.position.y - b.position.y,
5          myPlayer.sprite.position.x - b.position.x);
6      if (myPlayer.sprite.velocity.y >= -pixelWidth &&
7          myPlayer.sprite.velocity.y <= pixelWidth) {
8          myPlayer.sprite.velocity.x -= cos(angle);
9      }
10     myPlayer.sprite.velocity.y -= sin(angle);
11 }

```

Die Black-Hole-Physics Funktion sieht also (vereinfacht) so aus:

Listing 6: Black Hole Item Physics

```

1  // if the item has reached a certain life, make it static and attract players
2  if (b.life <= 400) {
3      attraction(b);
4      b.velocity.y = 0;
5      b.velocity.x = 0;
6  }
7
8  // if the item has not reached a certain life, let it bounce off the
9  // environment
10 if (b.life > 400) {
11     b.velocity.y -= GRAVITY;
12     b.bounce(environment);
13 }
14
15 // if the item is outside of the monitor, delete it
16 if (b.position.x > windowHeight || b.position.y > windowWidth || b.life ==
17     0) {
18     b.remove();
19 }

```

## Piano

Das Piano-Item wird mit der Taste <R> aktiviert.

Das Piano-Item ist, wie der Namen schon vermuten lässt, ein Klavier, das am höchsten

Punkt der Map erscheint und nach unten fällt. Das bedeutet, die y-Koordinate ist 0 und die x-Koordinate ist die selbe wie die, die der Sprite des Spielers hat, der das Piano aktiviert hat. Bei Kontakt zu einem Spieler oder der Umgebung wird das Klavier zerstört. Zum Überprüfen auf Kollisionen wird die p5.js Methode <sprite.collide(otherSprite)> verwendet. Diese wird true, falls eine Kollisionen zwischen zwei Sprites oder Sprite-Gruppen stattfindet.

Listing 7: Piano-Item Physics

```

1 // check for collisions
2 if (p.collide(environment)) {
3     p.remove();
4 } else if (p.overlap(myPlayer.sprite)) {
5     myPlayer.sprite.getsThrownAway()
6     p.remove();
7 }
8 p.velocity.y -= GRAVITY;

```

## Mine

*Das Minen-Item wird mit der Taste <C> aktiviert.* Das Minen-Item ist das einzige Item, bei dem der Spieler mehrere Instanzen auf einmal entsenden kann. Es taucht hinter dem eigenen Player-Sprite auf und fliegt so lange nach unten, bis es auf der Umgebung landet. Erst wenn es wo gelandet wird, wird es 'aktiv'. Wenn eine Mine aktiviert worden ist, und ein Spieler in Berührung mit dem Item kommt, wird dieser in die Luft gestoßen und die Mine wird gelöscht.

Listing 8: Mine-Item Physics

```

1 // check if mine has landed somewhere (if true: activate mine)
2 if (m.collide(environment) && m.touching.bottom) {
3     m.set = true;
4 }
5 if (m.overlap(myPlayer.sprite) && m.set) {
6     myPlayer.sprite.getsThrownAway();
7     m.remove();
8 }
9 m.velocity.y -= GRAVITY;

```

## Size-Reduction

*Das Size-Reduction-Item wird mit der Taste <F> aktiviert.*

Dieses Item ist das einzige, das keinen eigenen Sprite hat. Das einzige was diese Item macht, ist, den Spieler-Sprite zu verkleinern. Dadurch wird dieser schwieriger zu treffen. Wenn das Item aktiviert wird, wird der Sprite des Spielers verkleinert und ein Counter wird gestartet. Dieser wird alle 60 Frames um eins verringert. Ist der Counter 0, wird der Sprite wieder in seine Originalgröße gebracht. Der Code, der dies umsetzt, sieht

vereinfacht so aus:

Listing 9: Size-Reduction

```

1 // gets called on key press F
2 function makeMeSmall() {
3     if (doIHaveTheItem()) {
4         imSmall = true;
5         smallTimer = 10;
6     }
7 }
8
9 // gets called in draw function
10 function smallChecker() {
11     if (imSmall) {
12         // scale the sprite down at the start of countdown
13         if (smallTimer == 10) {
14             myPlayer.sprite.scale = 0.6;
15         }
16         // every second (60 frames), the countdown gets reduced
17         if (frameCount % 60 == 0 && smallTimer > 0) {
18             smallTimer--;
19         }
20         // if the countdown is over, rescale the sprite back to the original form
21         if (smallTimer == 0) {
22             myPlayer.sprite.scale = 1;
23             smallTimer = 10;
24             imSmall = false;
25         }
26     }
27 }
28 }
```

## Die Default-Attacke

*Die Default-Attacke wird mit <Left-Click> aktiviert.*

p5.js bietet sehr leicht die Möglichkeit, auf User-Input zu überprüfen. Das einzige, was nötig ist um zu erkennen, ob der User gerade die linke Maustaste geklickt hat, ist die Funktion `<mouseClicked()>`. Mit diesen Funktionalitäten wird nun zu dem Punkt, auf dem der Spieler gerade die Maus hält und die Default-Attacke aktiviert, ein Projektil abgeschossen. Um die Information zu erhalten, auf welcher X- und Y-Position sich die Maus befindet, werden die von p5.js vordefinierten Eigenschaft `<camera.mouseX>` und `<camera.mouseY>` verwendet.

Listing 10: Default-Attacke

```

1     function mouseClicked() {
2         // Maus-Position
3         let x = camera.mouseX,
4             y = camera.mouseY;
5         // Sprite wird bei meiner Player-Sprite-Positon erstellt
6         projectile = createSprite(myPlayer.sprite.position.x,
7             myPlayer.sprite.position.y, pixelWidth, pixelWidth);
8         // Geschwindigkeit wird auf die Position der Maus ausgerichtet
9         projectile.velocity.x = (x - myPlayer.sprite.position.x);
10        projectile.velocity.y = (y - myPlayer.sprite.position.y);
11    }
```

## Movements

Es gibt 3 fundamentale Bewegungsmöglichkeiten in Scribble-Fight. Springen, links/rechts laufen und 'klettern'. Diese Bewegungen werden im Folgenden genauer erläutert.

### Springen

In dem Web-Game kann der Spieler mittels Leertaste springen. Genau wie bei den Item-Aktivierungen, wird mithilfe von p5.js Methoden der User-Input ermittelt. Bei dem Springbewegung wird einfach die vertikale Geschwindigkeit des Spieler-Sprites so verändert, dass dieser etwas nach oben springt.

Listing 11: Jumping

```

1  // check if user pressed spacebar
2  if (keyWentDown(32)) {
3      jump()
4  }
5
6  function jump() {
7      // user is only allowed to jump 2 times (it resets when touching the ground)
8      if (!(JUMP_COUNT >= MAX_JUMP)) {
9          // make the user fly up a bit (JUMP is a global variable)
10         myPlayer.sprite.velocity.y = -JUMP;
11         JUMP_COUNT++;
12     }
13 }
```

### Links/Rechts Laufen

Das Prinzip des Links oder Rechts Bewegens ist sehr simpel. Es wird einfach die horizontale Geschwindigkeit des Player-Sprites auf eine konstante Variable gesetzt. Wenn sich der Spieler nach rechts bewegt, ist diese Konstante positiv, bei einer Linksbewegung negativ. Im Gegensatz zur Springbewegung wird diesmal aber die Methode `<keyIsDown(key)>` verwendet, und nicht `<keyWentDown(key)>`. Der Grund dafür ist, dass die Bewegung so lange anhalten soll, wie der User die Taste drückt, und nicht nur einmal pro Tastendruck.

Listing 12: Links/Rechts-Movement

```

1  //A
2  if (keyIsDown(65)) {
3      moveLeft()
4  }
5  //D
6  if (keyIsDown(68)) {
7      moveRight()
8  }
9
10 // SPEED is a global variable
11 function moveLeft() {
12     myPlayer.sprite.velocity.x = -SPEED;
13 }
14
15 function moveRight() {
16     myPlayer.sprite.velocity.x = SPEED;
17 }
```

## Bewegung auf der Spielumgebung

Die Bewegung auf der Spielumgebung wird durch die Methode <collisions()> bestimmt. Diese wird in der draw-Methode aufgerufen und wird somit 60 mal die Sekunde ausgeführt. Berührt der Sprite des Players nichts, fällt er mit konstanter Beschleunigung nach unten. Findet jedoch eine Kollision mit der Spielumgebung statt, dann wird überprüft, welche Art von Berührung gerade stattfindet:

- Falls Berührung seitlich stattfindet: Player-Sprite bekommt eine Klettergeschwindigkeit und behält diese so lange, wie die Berührung stattfindet
- Falls Berührung unten stattfindet: Die vertikale Geschwindigkeit des Player-Sprites wird auf 0 gesetzt und der Sprung-Counter zurückgesetzt
- Falls Berührung oben stattfindet: Mit einer Berührung die zwischen Spielumgebung und Sprite stattfindet, kann der Spieler nicht 'klettern' noch wird der Sprung-Counter zurückgesetzt

Listing 13: Bewegung auf der Spielumgebung

```

1      // check for collisions
2      if (myPlayer.sprite.collide(environment)) {
3          // if the collision is on the side, the sprite will start "climbing" with a
4          // certain speed
5          if (myPlayer.sprite.touching.left || myPlayer.sprite.touching.right) {
6              myPlayer.sprite.velocity.y = CLIMBINGSPEED;
7          }
8          // standing on the environment
9          if (myPlayer.sprite.touching.bottom) {
10             myPlayer.sprite.velocity.y = 0;
11         }
12         // jump count gets only reset when the collision is not on the top of the
13         // player-sprite
14         if (!myPlayer.sprite.touching.top) {
15             JUMP_COUNT = 0;
16         }
17     }

```

## Knockback-Bewegung

Wenn der Spieler von einem Projektil getroffen wurde, wird der eigene Player-Sprite für kurze Zeit bewegungsunfähig und prallt von der Spielumgebung ab. Wie lang dieses Knockback-Movement andauert, kommt auf die Art des Projektiles und auf den Wert des Knockbacks des Players an. Soll der Player-Sprite also nun diese Bewegung ausführen, wird das mit einer Funktion namens <sendHimFlying()> bewerkstelligt. Bevor diese aufgerufen wird, muss noch wie der Countdown, wie lange der Sprite nun in dieser Bewegung bleiben soll, festgelegt werden. Bei jedem draw-Zyklus wird dieser Countdown um den Wert eins verringert. Hinzu kommt, dass wenn der Spieler getroffen wurde, er kurz etwas verlangsamt wird. Es soll so wirken, als wäre er gerade etwas betäubt worden.

Listing 14: Knockback-Bewegung

```

1 // before function gets called, make sure to set flying to true and set a
2 // flying-duration
3 function sendHimFlying() {
4     if (flying) {
5         timeFlying--;
6         //slowdown but only at the first half of flying-duration
7         if (timeFlying <= flyingDuration / 2 && timeFlying > 0) {
8             if (myPlayer.sprite.velocity.x > 0) { myPlayer.sprite.velocity.x -= 0.3; }
9             if (myPlayer.sprite.velocity.x < 0) { myPlayer.sprite.velocity.x += 0.3; }
10            if (myPlayer.sprite.velocity.y > 0) { myPlayer.sprite.velocity.y -= 0.3; }
11            if (myPlayer.sprite.velocity.y < 0) { myPlayer.sprite.velocity.y += 0.3; }
12        }
13        // flying-duration is over
14        if (timeFlying == 0) {
15            flying = false;
16        }
17    }
}

```

## Richtungswechsel des Sprites

Ein weiterer wichtiger Aspekt bei der Bewegung des Player-Sprites ist, dass sich auch die Orientierung des Bildes ändern muss, wenn dieser die Richtung wechselt. Auch für diese Problemstellung stellt p5.play.js eine Lösung zu Verfügung. Mit der Methode `<mirrorX()>` wird der Sprite entlang seiner vertikalen Achse gespiegelt. Jedes mal, wenn der User nun also die Richtung seines Player-Sprites wechselt, wird auch der Sprites passend seiner Bewegung gespiegelt.

Listing 15: Sprite Richtungswechsel

```

1     function mirrorSprite() {
2         // A
3         if (keyWentDown(65)) {
4             mirrorSpriteLeft()
5         }
6         // D
7         if (keyWentDown(68)) {
8             mirrorSpriteRight()
9         }
10    }
11
12    function mirrorSpriteLeft() {
13        // if the mirrorX attribute is 1, then the sprite is looking to the right
14        if (myPlayer.sprite.mirrorX() === 1) {
15            myPlayer.sprite.mirrorX(myPlayer.sprite.mirrorX() * -1);
16            myPlayer.direction = "left";
17        }
18    }
19
20    function mirrorSpriteRight() {
21        // if the mirrorX attribute is 1, then the sprite is looking to the left
22        if (myPlayer.sprite.mirrorX() === -1) {
23            myPlayer.sprite.mirrorX(myPlayer.sprite.mirrorX() * -1);
24            myPlayer.direction = "right";
25        }
26    }
}

```

## Wie gewinne ich?

Um in Scribble-Fight zu gewinnen, muss der Spieler seinen Gegner/seine Gegner drei mal erfolgreich von der Spielumgebung schießen, so, dass der Sprite des Gegners ins 'Nichts' fällt. Dies ist durch die Attacken, die in Kapitel 5.1.1 genau beschrieben werden, möglich. Um zu überprüfen, ob der Sprite nun hinuntergefallen und somit 'gestorben'

ist, wird in der draw-Methode die Funktion <deathCheck()> aufgerufen. Diese hat einige Aufgaben:

- Überprüfen ob sich der Player-Sprite außerhalb des Bildschirms befindet
- Falls ja, überprüfen ob der Player noch mindestens ein Leben hat
- Falls der Player keine Leben mehr hat, wird sein Sprite zerstört
- Hat der Player noch weitere Leben, dann wird er nach drei Sekunden an einem zufälligen Punkt, an dem er nicht direkt wieder aus dem Bildschirm fällt, respawned
- Wird der Player respawned, werden ihm alle seine Items, die er eventuell noch hatte, wieder genommen
- Wird der Player respawned, wird sein Knockback wieder zurückgesetzt

Vereinfacht sieht die Methode also so aus:

Listing 16: Überprüfung nach Toden

```

1  function deathCheck() {
2      // check if sprite has fallen outside of the monitor
3      if (myPlayer.sprite.position.y - player_height > windowHeight) {
4          youDied();
5      }
6  }
7
8  function youDied() {
9      myPlayer.removeItem();
10     myPlayer.death++;
11     myPlayer.knockback = 1;
12
13     // after 3 seconds the player gets respawned on a location, if he still has at
14     // least one live left
15     setTimeout(() => {
16         if (myPlayer.death < 3) {
17             myPlayer.sprite.position.x = xCoordinates[Math.floor(Math.random() *
18                 xCoordinates.length)];
19             myPlayer.sprite.position.y = 0;
20         }
21     }, 3000);
22 }
```

Außerdem zählt es auch als Tod, wenn der Knockback des eigenen Players über 100 ansteigt. Dies wird mit der <fatalHit()>-Methode überprüft. Diese Methode wird immer dann aufgerufen, wenn der eigene Player von irgendeinem Projektil getroffen wurde.

Listing 17: Fatal Hit

```

1  function fatalHit() {
2      if (myPlayer.knockback >= 100) {
3          youDied();
4      }
5  }
```

Ein Indikator, ob der Spieler gut abgeschnitten hat, sind die Kills, die er erzielen konnte. In Scribble-Fight kann der Spieler Kills sammeln, indem er jemanden mit jeglicher Art von Projektil trifft, und dieser innerhalb von 3 Sekunden aus der Spielumgebung

fliegt. Dies funktioniert so, dass wenn jemand meinen player-Sprite trifft, in meinem Player-Objekt abgespeichert wird, welcher Spieler mich gerade getroffen hat. Diese Information wird aber alle 3 Sekunden wieder gelöscht.

```

1      function draw() {
2          // every second, the countdown gets reduced by 1
3          if (frameCount % 60 == 0 && damagedByTimer > 0 && myPlayer.damagedBy != null) {
4              damagedByTimer--;
5          }
6
7          // if the countdown has reached 0, the information gets deleted and the
8          // countdown restarts
9          if (damagedByTimer == 0) {
10              damagedByTimer = 3;
11              myPlayer.damagedBy = null;
12      }

```

Wenn ich nun sterbe, und die Information, dass mich jemand getroffen hat in meinem Player-Objekt gespeichert ist, bekommt dieser einen Kill.

## Erstellung der Spielumgebung

Um sich auch auf dem Bild, das der User gezeichnet hat, bewegen zu können, müssen einiges Schritte durchgeführt werden. Mittels Objekterkennung kann das Bild abfotografiert, und daraus ein Array mit Bildaten erstellt werden, aus dem dann die Spielumgebung kreiert wird. Wie genau dieser Array zustande kommt, wird in Kapitel TODO beschrieben.

Listing 18: Vereinfachte Darstellung eines Bilddaten-Arrays

```

1      [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0],
2      [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0],
3      [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0],
4      [255, 255, 255, 0], [252, 252, 252, 0], [251, 251, 251, 0],
5      [248, 248, 248, 0], [249, 249, 249, 0], [247, 247, 247, 0],
6      [255, 255, 255, 0], [240, 240, 240, 0], [250, 250, 250, 0],
7      [174, 174, 174, 0], [255, 255, 255, 0], [173, 173, 173, 0],
8      [226, 226, 226, 0], [255, 255, 255, 0], [253, 253, 253, 0],
9      [163, 163, 163, 0], [254, 254, 254, 0], [255, 255, 255, 0],
10     [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0],

```

Desto genauer dieser Array ist, desto genauer werden auch die Hitboxen der Spielumgebung, denn die gezeichneten Formen werden mit Rechteck-Hitboxen nachgestellt. Eine Hitbox ist ein Bereich, der für die Berechnung für Kollision benutzt wird. Hat ein Eintrag des Arrays an Stelle vier mehr als 0 als Wert, wird dort ein Pixel erstellt. Das liegt daran, dass an der vierten Stelle eines solchen Eintrags die Deckkraft der Bildstelle angegeben wird. Das bedeutet, der User hat dort etwas gezeichnet. Natürlich muss der Sprite-Pixel noch einen richtigen X- und Y-Wert bekommen. Diese Koordinate muss auch mit dem Punkt des Bildes übereinstimmen, an den der Sprite als Hitbox

agieren soll. Aus Performance-Gründen (Es wird durchgehend auf Kollisionierung zwischen Player und Spielumgebung geprüft) werden die Sprite-Pixel entlang der X-Achse noch zusammengefasst. Wurden nun alle Sprite-Pixel ausfindig gemacht, mit richtigen Koordinaten versehen und entlang der horizontalen Achse zusammengefasst, werden alle einer p5-Group-Variable hinzugefügt. Das macht das Überprüfen auf Kollision sehr leicht.

Die nächsten drei Bilder sollen den Ablauf bildlich darstellen.

Zuerst das originale, vom User gezeichnete Bild.

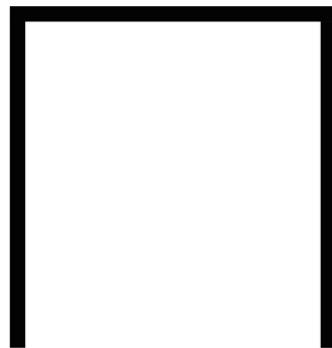


Abbildung 18: Originale Zeichnung

Aus diesem Bild kann ein Array aus Bilddaten erstellt werden (Wie in Kapitel TODO beschrieben). Wenn aber einfach mit diesem Array die Pixel für die Spielumgebung erstellt, sind diese noch viel zu klein und an der falschen Position.



Abbildung 19: Bilddaten-Array bildlich dargestellt

Um die richtigen Koordinaten für die Sprite-Pixel zu bestimmen, werden noch einige andere Faktoren miteinbezogen und daraus dann die richtige Position der Pixel am Bildschirm ermittelt. Außerdem werden sie noch entlang der X-Achse kombiniert.

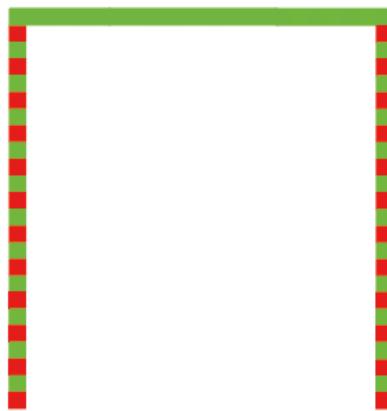


Abbildung 20: Spielumgebung

Natürlich werden dann diese Sprites versteckt. Es soll ja so wirken, also würde sich der User auf dem, was er gerade gezeichnet hat, bewegen können. Im Folgenden wird der Algorithmus zum Aufbereiten der Spielumgebung vereinfacht dargestellt.

```

1   environment = new Group();
2   // looping through image data array
3   for (let i = 0; i < pixel_clumps.length; i++) {
4     sprite_pixels[i] = [];
5     for (let j = 0; j < pixel_clumps[0].length; j++) {
6       if the value is greater than 0, then something has been drawn there
7       if (pixel_clumps[i][j][3] > 0) {
8         //if the last value in the array is not undefined, we can merge the
9         // sprite-pixels
10        if (sprite_pixels[i][j - 1] !== undefined) {
11          same_x_counter++;
12          sprite_pixels[i][j] = createSprite(pixelWidth * faktorX, pielWidth *
13                                              faktorY, pixelWidth * same_x_counter, pixelWidth);
14          // add sprite to environment group
15          environment.add(sprite_pixels[i][j]);
16          // remove the last value, because it has been replaced by a new
17          // sprite, with a greater width
18          sprite_pixels[i][j - 1].remove();
19          sprite_pixels[i][j - 1] = undefined;
20        } else {
21          same_x_counter = 1;
22          sprite_pixels[i][j] = createSprite(pixelWidth * faktorX, pielWidth *
23                                              faktorY, pixelWidth, pixelWidth);
24          // add sprite to environment group
25          environment.add(sprite_pixels[i][j]);
26        }
27      }
28    }
29  }

```

### Item-Spawns

Alle 10 Sekunden wird ein Item erstellt. Es wird an oberster Stelle des Bildschirms kreiert ( $y$ -Koordinate = 0); die  $x$ -Position dieses Items wird dann zufällig ausgewählt. Wichtig dabei ist jedoch, dass das Items auf keiner Position spawnen darf, bei der es einfach oben auftaucht und dann aus dem Bildschirm fällt. Wie diese  $x$ -Koordinaten berechnet werden, wird in Kapitel 5.1.1 genau beschrieben. Damit die Items voneinander unterschieden werden können, wird jede Art von Item farblich gekennzeichnet:

- Rot: Bomb
- Blau: Black-Hole
- Gelb: Piano
- Orange: Mine
- Grün: Size-Reduction

Den Input, wann ein Item genau erstellt wird, (muss bei jedem User der gerade spielt gleich sein) liefert der Server. Genauere Details wie der Server von des Web-Games funktioniert, wird in Kapitel 5.1.2 beschrieben.

Listing 19: Erstellen eines Items

```

1  funciton createItem(data) {
2      // number between 1-5 from the server to create a random item
3      let num = data.num;
4      // random x-Coordinate from the server
5      let x = data.x
6      // this equation gets you the highest point of the map
7      let y = (windowHeight - ImageHeight) / 2;
8      // if x = -1, something on the server-side went wrong
9      if (x != -1) {
10          // itemSize is a global variable
11          switch (num) {
12              case 1:
13                  i = createSprite(x, y, itemSize, itemSize);
14                  i.type = "bomb";
15                  break;
16              case 2:
17                  i = createSprite(x, y, itemSize, itemSize);
18                  i.type = "black_hole";
19                  break;
20              case 3:
21                  i = createSprite(x, y, itemSize, itemSize);
22                  i.type = "piano";
23                  break;
24              case 4:
25                  i = createSprite(x, y, itemSize, itemSize);
26                  i.type = "mine";
27                  break;
28              case 5:
29                  i = createSprite(x, y, itemSize, itemSize);
30                  i.type = "small";
31                  break;
32          }
33          i.dropped = false;
34          items.push(i);
35      }
}

```

Nachdem nun das Item erstellt wurde, braucht dieses natürlich auch so wie zum Beispiel die Projektilen oder der Player eine eigene Physik-Methode. Diese wird, so wie die anderen Physik-Methoden auch, in der Draw-Methode aufgerufen. Die Methode funktioniert so, dass das Item so lange fällt, bis es irgenwo auf der Spielumgebung landet. Jeder Player kann jeder Zeit das Item berühren und somit eine neue Fähigkeit bekommen.

Listing 20: Item-Physik

```

1  function itemPickUp() {
2      if (items.length > 0) {
3          items.forEach(item => {
4              // if the item collides with the environment, the
5              // dropped-attribute becomes true
6          });
7      }
}

```

```

5             if(item.collide(environment)) {
6                 item.dropped = true;
7             }
8             // if the item has not landed anywhere, let it fall
9             if(!item.dropped) {
10                 item.velocity.y -= GRAVITY;
11             }
12
13             // if I collide with the item, I get a new ability depending on
14             // the type of the item and the item gets deleted
15             if (item.overlap(myPlayer.sprite)) {
16                 myPlayer.item[item.type] = new Item(item.type);
17                 deleteItem(item);
18             }
19         }

```

### Bestimmung von gültigen X-Koordinaten

Mit gültiger X-Koordinaten sind jene X-Koordinaten gemeint, bei denen ein Sprite erstellt werden kann, und dieser dann irgendwo auf der Spielumgebung landet und nicht sofort aus dem Bildschirm fliegt. Bei dem Algorithmus wird zuerst überprüft, auf welchen Stellen Sprite-Pixel (Pixel aus denen die Spielumgebung besteht) vorhanden sind. Von diesen kann schon der Mittelpunkt als gültige X-Koordinaten genommen werden, solange dieser Pixel eine gewisse Breite aufweist. Doch damit bei einem sehr breiten Pixel nicht nur eine einzige X-Koordinate ausgewählt wird, wird schrittweise überprüft, ob noch Stellen vor oder hinter dem Mittelpunkt des Sprites in Frage kommen. Dazu wird dieser Sprite-Pixel nach vorne und nach hinten abgetastet, ob noch genügend Platz da ist, ein Item dort landen zu lassen. Es können höchstens halb so viele X-Koordinaten pro Sprite-Pixel ausgewählt werden, wie der Pixel (in Pixel-Width gemessen) breit ist. Die Funktion wird im Setup von Scribble-Fight aufgerufen.

Listing 21: Bestimmung gültiger X-Koordinaten

```

1     function getXCoordinates() {
2         let sprite;
3         // looping through the sprite_pixel array
4         for (let i = 0; i < sprite_pixels.length; i++) {
5             for (let j = 0; j < sprite_pixels[i].length; j++) {
6                 sprite = sprite_pixels[i][j];
7                 // if the sprite variable is not undefined, it means a sprite
7                 // exists there
8                 // the sprite hast to be a width of at least 4 times the normal
8                 // pixel width
9                 if (sprite !== undefined && sprite.width >= pixelWidth * 4) {
10                     // sprites gets checked if there are more Coordinates to let
10                     // an item spawn there (to the right of the center)
11                     for (let index = 0; index < sprite.width / 2; index +=
11                         pixelWidth * 2) {
12                         if (sprite.position.x + index < sprite.position.x +
12                             sprite.width / 2) {
13                             let x = sprite.position.x + index;
14                             xCoordinates.push(x);
15                         }
16                     }
17                     // sprites gets checked if there are more Coordinates to let
17                     // an item spawn there (to the left of the center)
18                     for (let index = sprite.width; index > sprite.width / 2; index =
18                         - pixelWidth * 2) {
19                         if (sprite.position.x + index > sprite.position.x +
19                             sprite.width / 2) {
20                             let x = sprite.position.x + index - sprite.width;
21                             xCoordinates.push(x);
22                         }
23                 }
24             }
25         }
26     }

```

```

23         }
24         // doing this eliminates duplicates
25         xCoordinates = Array.from(new Set(xCoordinates));
26     }
27 }
28 return xCoordinates;
29
30 }

```

## 5.1.2 Server [R]

Der Server von Scribble-Fight ist ein einfacher Web-Server, der mit Node.js und express.js umgesetzt wurde. Dieser hostet statische Files, auf denen sich der Code für das Frontend befindet (HTML Files, p5.js library, etc). Viele Teile dieses Codes werden im vorherigen Kapitel detailliert erklärt. Durch SocketIO können multiple Clients eine Verbindung mit dem Server aufbauen. So wird es ermöglicht, dass unterschiedliche Clients gemeinsam spielen können.

### Erstellen des Servers

Ein Node.js Server ist einfach zu erstellen. Zwei Voraussetzungen müssen vorhanden sein:

- Eine funktionierende Node.js-Version
- Java-Script Grundlagen

Ist dies gegeben, kann ein neuen Ordner erstellt, und in diesem dann der Befehl `npm init` ausgeführt werden. Dort wird dann ein `package.json` File für das Node-Projekt erstellt. Nähere Infos zu diesem File kann in Kapitel 4.1.3 nachgelesen werden. Danach kann entweder durch eine Entwicklungsumgebung ein neues Java-Script File erstellt, oder im Terminal der Befehl `touch server.js` eingegeben werden. In dem Backend von Scribble-Fight werden `express.js` und `SocketIO` benutzt, deshalb müssen die Befehle `npm install express -save` für `express.js`, und `npm install socket.io` für `SocketIO` im Terminal ausgeführt werden. Dies installiert die Module und fügt die Dependencies zu dem `package.json`-File hinzu.

Wurde all das berücksichtigt, wird jetzt die Logik des `server.js`-Files umgesetzt. Zuerst wird eine Instanz von `express.js` erstellt. Danach wird ein HTTP-Server-Objekt und eine `SocketIO` Instanz angelegt, die das HTTP-Server-Objekt als Paramter nimmt. Zusätzlich wird noch Cross-Origin Resource Sharing freigegeben. Der Code sieht also so aus:

```
1 var express = require("express");
```

```

2     var app = express();
3     var http = require("http").createServer(app);
4     var io = require("socket.io")(http, {
5       cors: {
6         origin: '*',
7       }
8   });

```

Das war aber nur ein Teil, um einen funktionierenden SocketIO-Webserver zu erstellen. Bei unserem Web-Game müssen, wie vorher schon erwähnt, das Frontend als statische Files gehostet werden. Für dieses Problem wurde das Paths-Module von node.js verwendet. Dieses stellt Funktionen für das Arbeiten mit File-und Directory-Pfaden zur Verfügung. Darauf zugegriffen werden wird mit:

```
1 const path = require('path');
```

Nun können die Files einfach mit

```

1 // path.join is used to get the right path
2 app.use(express.static(path.join(__dirname, '/..../p5_frontend/src')));

```

gehostet werden. (Diese befinden sich in dem Folder <src>);

Als letzten Schritt wird noch der Port bei dem HTTP-Server-Objekt angegeben, im Fall von unserem Server ist dieser 3000.

```

1 // Server is listening on port 3000
2 http.listen(3000);

```

Ist das alles abgeschlossen, wird nun die SocketIO Logik umgesetzt.

## SocketIO Logik

Der nun funktionierende Web-Server soll nun SocketIO-Verbindungen, die vom Browser kommen, annehmen können. Mit der SocketIO Instanz, die im letzten Abschnitt erstellt worden ist, kann dieses Problem sehr leicht gelöst werden:

```

1 io.sockets.on('connection', newConnection);
2
3 function newConnection(socket) {
4   // LOGIC COMES HERE
5 }

```

Findet nun eine Verbindung zu dem Server von einem Client statt, wird die Methode <newConnection(socket)> aufgerufen. Der Parameter der Methode ist eine Socket-Instanz, die sehr viele Informationen und Funktionalitäten der gerade bestehenden Verbindung enthält. Wie der Client eine Verbindung zu diesem Server aufbauen kann, ist von Programmiersprache zu Programmiersprache unterschiedlich. Im Frontend von Scribble-Fight wird Java-Scritp benutzt.

## SocketIO im Frontend

Um im Frontend eine SocketIO-Verbindung aufzubauen, wird zuerst in dem index.html File die SocketIO Library verlinkt:

```
1 // link SocketIO in index.html
2 <script src="https://cdn.socket.io/4.1.2/socket.io.min.js"></script>
```

Jetzt kann einfach eine neue Socket-Instanz erstellt werden mit der Client Events empfangen, aber auch aussenden kann. Da der Server von Scribble-Fight nicht mehr lokal läuft, sondern auf einem Kubernetes-Cluster, wird hierbei noch ein Pfad definiert, um eine korrekte Instanz zu erstellen.

```
1 var socket = io({
2   path: "/t.rafetseder/scribble-fight/socket.io"
3});
```

Einfach nur der Befehl `io()` würde die Pfad-Parameter (`/t.rafetseder/scribble-fight/socket.io`) der derzeitigen Domaine einfach ignorieren und die Verbindung könnte nicht hergestellt werden.

Nun kann mit dem Befehl

```
socket.on('event', funciton) Events empfangen, und mit socket.emit('event') Events auslösen.
```

## SocketIO Events

Nachdem nun eine aufrechte Verbindung hergestellt werden konnte, gibt es 10 Events, die passieren können.

- Ein neuer Spieler soll erstellt werden
- Abruf von schon bestehenden Spielern
- Die Position von meinem Sprite soll geupdatet werden
- Die Richtung, in die mein Sprite schaut, soll geupdatet werden
- Ein Item wurde aufgehoben, also soll es bei jedem entfernt werden
- Jemand hat eine Attacke ausgeführt, also soll bei jedem ein Projektil erstellt werden
- Jemand hat einen Kill erzielt
- Ein Spieler wurde von einem Projektil getroffen, das heißt das Projektil, das getroffen hat, muss bei jedem entfernt werden
- Jemand ist gestorben
- Es soll ein neues Item erstellt werden (Passiert alle 10 Sekunden)

Diese Events werden im Folgenden genau erklärt.

### **Erstellen eines neuen Spielers und Abrufen von vorhandenen Spieler**

Führt der Spieler das Frontend aus, stellt dieses sofort eine Verbindung mit dem Server

her. Um nun schon vorhandene Spieler, die gerade auch verbunden sind zu erhalten, wird ein Event <getPlayers> emittiert. Danach muss dem Server noch mitgeteilt werden, dass der Spieler nun selbst auch eine spielbare Figur erhalten möchte. Dies wird mit dem <newPlayer>-Event bewerkstelligt. Wird nun das Event <newPlayer> vom Server ausgelöst, wird ein neues Player-Objekt dem Player-Array des Frontends hinzugefügt. Der Konstruktor von diesem Player-Objekt benötigt nur einen Sprite, der auch neu erstellt wird.

```

1      // listen to an event from the server
2      socket.on('newPlayer',createNewPlayer);
3      // get existing players
4      socket.emit('getPlayers');
5      // create my own player
6      socket.emit('newPlayer');
7
8      function createNewPlayer(data) {
9          // new player gets added an new sprite is created in the middle of the map
10         players[data.id] = new Player(createSprite(ImageWidth / 2,ImageHeight / 2,
11                                         player_width, player_height));
12     }

```

Im Server sieht das Ganze etwas komplizierter aus. Grundsätzlich speichert der Server alle Player-Objekte in einem Map-Objekt ab. Die ID für jedes dieser Player-Objekte dient einfach die ID der Socket-Verbindung. Will ein Client nun alle schon vorhandenen Spieler, wird die Map (falls Spieler vorhanden) durchlaufen, und die ID von jedem der Spieler an den anfragenden Client geschickt. Dieser erstellt dann einen neuen Player / mehrere neue Player mit der dazugehörigen ID.

```

1      socket.on('getPlayers',sendPlayers);
2
3      // the object players is the map that has the player-objects saved
4      function sendPlayers() {
5          if (players.size > 0) {
6              players.forEach((values, keys) => {
7                  let data = {
8                      id: values.id,
9                  }
10                 socket.emit('newPlayer', data);
11             })
12         }
13     }

```

Nachdem nun der Server alle schon vorhanden Spieler dem Client mitgeteilt hat, muss noch ein neuer Spieler für diesen Client erstellt werden.

```

1      socket.on('newPlayer',createPlayer);
2
3      function createPlayer() {
4          players.set(socket.id, new Player(socket.id));
5          let data = {
6              id: socket.id,
7          }
8          // im using io.emit() so it sends to everyone, including the client that
9          // triggered the event
10         io.emit('newPlayer', data);
11     }

```

Der Client erstellt nun genau gleich wie bei dem <getPlayers>-Event den Spieler.

## Update Position and Update Direction

Die Position des Sprites von jedem Spieler soll in jedem Draw-Zyklus geupdatet werden. Die Orientierung des Sprites soll nur dann aktualisiert werden, wenn der Spieler die Richtung seines Sprites wechselt. (Beschrieben in Kapitel 5.1.1) Leider ergibt sich beim Aktualisieren von Positionen von Sprites ein Problem. Die Koordinaten in p5.js weden in Pixel angeben. Das bedeutet, haben unterschiedliche Client unterschiedlich große Endgeräte, befinden sich Sprites mit gleichen Koordinaten an unterschiedlichen Stellen.



Abbildung 21: Pixel-Unterschied

Aus diesem Grund können nicht einfach nur die rohen Koordinaten an alle Clients geschickt werden, sie müssen in Relation zu dem Bildschirm des Endgeräts und der Größe der Map, also dem Bild auf dem gerade gespielt wird, gesetzt werden. (Dies ist nicht nur bei den Spieler-Koordinaten, sondern auch bei z.B. Items der Fall) Der Code am Ende von jedem Draw-Zyklus sieht also so aus:

```

1  let transferX = (myPlayer.sprite.position.x - (windowWidth - ImageWidth) / 2)
   * originalBildBreite / neueBildBreite;
2  let transferY = (myPlayer.sprite.position.y - (windowHeight - ImageHeight) /
   2) * originalBildHoehe / neueBildHoehe;
3  relPosData = {
4    x: transferX,
5    y: transferY
6  }
7  socket.emit('update', relPosData);

```

Im Server werden diese Koordinaten einfach zusammen mit der passenden ID des Players an alle Spieler verteilt:

```

1  socket.on('update', updatePosition)
2
3  function updatePosition(data) {
4    let posData = {
5      x: data.x,
6

```

```

7             y: data.y,
8             id: socket.id
9         }
10        // with socket.broadcast, everyone except the one who triggered the event
11        // gets the data
12        socket.broadcast.emit('updatePosition', posData);
13    }

```

Empfängt ein Client die Koordinaten, muss er diese dann wieder für sein Endgerät richtig umwandeln:

```

1  socket.on('updatePosition',updatePosition);
2
3  function updatePosition(data) {
4      players[data.id].sprite.position.x =
5          data.x * neueBildBreite / originalBildBreite + (windowWidth -
6              neueBildBreite) / 2;
7
8      players[data.id].sprite.position.y =
9          data.y * neueBildHoehe / originalBildHoehe + (windowHeight -
10             neueBildHoehe) / 2;
11  }

```

Das Aktualisieren der Richtung des Sprites ist hingegen leichter. Dort spielt es keine Rolle, wie groß das Endgerät ist. Beim Drücken von A oder D auf der Tastatur, wird einfach ein Event getriggert, dass die Richtung des Sprites bei jedem verbundenen Spieler ändert.

```

1  // A
2  if (keyWentDown(65)) {
3      socket.emit('updateDirection', 'left');
4  }
5  // D
6  if (keyWentDown(68)) {
7      socket.emit('updateDirection', 'right');
8  }

```

Im Server wird die Information nun einfach an alle Clients verteilt:

```

1  socket.on('updateDirection',updateDirection);
2
3  function updateDirection(data) {
4      let dataWithId = {
5          id: socket.id,
6          direction: ""
7      }
8      if (data == "left") {
9          dataWithId.direction = "left";
10         socket.broadcast.emit('updateDirection', dataWithId);
11     } else if (data == "right") {
12         dataWithId.direction = "right";
13         socket.broadcast.emit('updateDirection', dataWithId);
14     }
15  }

```

Wird vom Client nun das <updateDirection>-Event empfangen, wird einfach die Orientierung des Sprites geändert.

### Synchronisieren von Projektilen oder Items

Wenn ich ein Item aufsammle, soll dieses natürlich für die anderen Spieler nicht mehr verfügbar sein. Genauso soll, wenn ich eine Attacke ausführe, diese bei allen anderen Spielern auch ausgeführt werden. Die Logik dazu ist relativ simpel. Jedes Item bekommt

beim Erstellen eine eigene unique ID. Sammelt nun irgendein Spieler dieses Item auf, wird an den Server eine Benachrichtigung gesendet, dass jemand dieses Item aufgesammelt hat. Dieser verteilt diese Information einfach an alle Spieler, und das Item mit der richtigen ID wird bei jedem gelöscht. Bei den Projektilen ist es ähnlich. Wenn ein Spieler eine Attacke ausführt, wird auch das dem Server mitgeteilt. Der verteilt nun wieder die Information an alle Spieler, und bei jedem wird ein neues Projektil mit der gleichen ID erstellt. Wenn irgendeiner der Spieler eine Kollision zwischen ihrem Player-Sprite und einem Projektil feststellt, wird das dem Server mitgeteilt und dieser teilt wieder allen anderen Spieler mit, dass das Projektil bei jedem gelöscht werden muss. Der Server hat einen Array mit Item-Objekten gespeichert. Vereinfacht sieht die Logik so auf der Server-Seite aus:

```

1   socket.on('deleteItem', deleteItem);
2   socket.on('attack', syncAttacks);
3   socket.on('deleteAttack', deleteAttack);
4
5   function deleteItem(data) {
6       items.forEach(i => {
7           if (i == data.id) {
8               // remove item from item array
9               items.splice(items.indexOf(i), 1);
10              socket.broadcast.emit('deleteItem', data);
11           }
12       });
13   }
14
15
16 // data consist of type of attack and attack ID
17 function syncAttacks(data) {
18     socket.broadcast.emit('attack', data);
19 }
20
21 // data consist of type of attack and attack ID
22 function deleteAttack(data) {
23     socket.broadcast.emit("deleteAttack", data);
24 }
```

Im Frontend werden einfach die bestimmten Events am richtigen Ort ausgelöst. Zum Beispiel, wenn eine Bombe erstellt wird:

```

1   funciton addBomb() {
2       // bomb gets created here
3       let data = {
4           playerId: socket.id,
5           type: "bomb",
6           x: bomb.position.x,
7           y: bomb.position.y,
8       }
9       socket.emit("attack",data);
10 }
```

Natürlich befindet sich auch die Logik, um die vom Server ausgelösten Events zu empfangen und darauf zu reagieren, am Client.

```

1   socket.on('deleteItem', syncItems);
2   socket.on('attack', addAttack);
3   socket.on('deleteAttack', deleteAttack);
4
5   // remove item from the item array
6   function syncItems(data) {
7       items[data.index].remove();
8       items.splice(items.indexOf(items[data.index]), 1);
9   }
10
```

```

11      // add a projectile, depending on the type of attack
12      function addAttack(data) {
13          switch (data.type) {
14              case "default": addDefaultAttack(data);
15              break;
16              case "bomb": addBomb(data);
17              break;
18              case "blackHole": addBlackHole(data);
19              break;
20              case "piano": addPiano(data);
21              break;
22              case "mine": addMine(data);
23              break;
24              case "small": addSmall(data);
25              break;
26          }
27      }
28
29      // delete the right projectile, depending on the type and the ID of the
30      // projectile
31      function deleteAttack(data) {
32
33          switch (data.type) {
34              case "default":
35                  projectiles.forEach(p => {
36                      if (p.id === data.id) {
37                          p.remove();
38                      }
39                  });
39                  break;
40              case "bomb":
41                  bombs.forEach(b => {
42                      if (b.id === data.id) {
43                          b.remove();
44                      }
45                  });
45                  break;
46              // ... and so on
47          }
48      }
49  }

```

Somit ist schon die meiste Logik des SocketIO-Servers gegeben. Was aber noch wichtiges fehlt, ist wenn jemand aus der Spielumgebung fällt, muss immer auf Gewinner überprüft und Kills abgespeichert werden.

## Tode und Kills

Wie der Spieler in Scribble-Fight gewinnt, wann er 'stirbt' und wie er Kills sammelt wird alles auf der Client-Seite in Kapitel 5.1.1 beschrieben. Doch natürlich muss der Client auch mit dem Server kommunizieren, damit dieser all diese Events auch an alle anderen Spieler mitteilen kann. Stirbt ein Spieler, teilt er dies dem Server mit. Dieser speichert die Information ab und überprüft, ob der Spieler noch weiter mitspielen darf. Ist er jedoch schon zum dritten Mal gestorben, ist dies nicht mehr der Fall und der Spieler wird aus der Spieler-Map vom Server gelöscht. Ist ab dem Zeitpunkt nur mehr ein einziger Spieler in dem Map-Objekt gespeichert, hat dieser gewonnen.

```

1      function death(data) {
2          players.get(data.deadPlayer).death++;
3          if (players.get(data.deadPlayer).death >= 3) {
4              // delete the player from player-map
5              players.delete(data.deadPlayer);
6              let transferData = {
7                  id: data.deadPlayer
8              }
9              // send the information that someone died to everyone
10             io.emit('death', transferData);
11             // disconnect the socket connection

```

```

12         socket.disconnect();
13     }
14     // if player-map only has one player left, he has won
15     if (players.size <= 1) {
16         socket.broadcast.emit("win");
17     }
18 }
```

Der Client muss natürlich jetzt auf beide Events, also falls er gerade vom Spiel ausgeschieden wurde, oder gerade gewonnen hat, reagieren können. Bekommt er die Benachrichtigung, dass gerade jemand ausgeschieden wurde, wird überprüft, ob es sich um ihn selbst handelt. Falls ja, wird die draw-Methode gestoppt, ihm wird mitgeilt, dass er verloren hat und eine kurze Übersicht von seinen Spiel-Statistiken wird angezeigt. Die Übersicht beinhaltet:

- Den Schaden, den er an alle anderen Spieler ausgeteilt hat
- Anzahl an Kills, die erzielt worden sind
- Der eigene Knockback, also ein Indikator wie oft der Spieler getroffen worden ist

Falls es sich nicht um ihn handelt, passiert nichts.

```

1     socket.on('death',someoneDied);
2
3     // data consists of id of dead player
4     function someoneDied(data) {
5         players[data.id].sprite.remove();
6         if (data.id == myPlayer.id) {
7             alert("You died!\n"
8                 + "Your kills: " + myPlayer.kills + "\n"
9                 + "Your damage: " + myPlayer.dmgDealt + "\n"
10                + "Your knockback: " + myPlayer.knockback);
11
12         // stop the draw-function
13         noLoop();
14     }
15 }
```

Bekommt der Spieler nun aber die Benachrichtigung, dass er gewonnen hat, wird diese Nachricht ausgegeben und auch hier wird wieder eine kurze Übersicht der Spiel-Statistiken gezeigt. Diesmal aber inklusive den Toden, da sie dieses Mal nicht mit Sicherheit drei sind.

```

1     socket.on('win',win);
2
3     function win() {
4         alert("You won!\n"
5             + "Your kills: " + myPlayer.kills + "\n"
6             + "Your damage: " + myPlayer.dmgDealt
7             + "\n" + "Your knockback: " + myPlayer.knockback
8             + "\n" + "Your deaths: " + myPlayer.death);
9     }

```

Wird im Client bestimmt, dass gerade ein Kill erzielt worden ist, speichert der Server diese Information ab, damit sie immer abrufbar ist.

```

1     socket.on('kill',kill);
2
3     // data consists of the id of the player who should get the kill
4     function kill(data) {
5         // player could leave before getting the kill
6         if (players.get(data.damagedBy) != undefined) {
```

```

7           players.get(data.damagedBy).kill += 1;
8       }

```

Nun hat der Server schon fast die ganze Funktionalität, die er benötigt, damit das Web-Game funktioniert. Das Einzige, was fehlt, ist dass der Server allen Spielern immer mitteilen muss, wann ein neues Item erscheinen soll.

### Item-Spawn-Event

Items werden in Scribble-Fight alle 10 Sekunden gespawned, falls ein Spieler mit dem Server verbunden ist. Auf der Server-Seite ist das leicht umzusetzen. Mit der Methode `setInterval()` kann einfach ein Intervall definiert werden, in dem eine Funktion ausgeführt werden soll. Das Intervall wird auf 10 Sekunden gesetzt. Nun wird, solange ein Spieler vorhanden ist, alle 10 Sekunde ein Event ausgelöst, dass allen verbundenen Spielern mitteilt, wo, welche Art von Item erstellt werden soll. Wie die x-Koordinaten berechnet werden, die für die Item-Spawns in Frage kommen, wird in Kapitel 5.1.1 beschrieben. Auf der Server-Seite sieht der Code also so aus:

```

1   setInterval(() => {
2     // if players are connected with the server
3     if (players.size > 0) {
4       // get one of the available spawn points for items
5       let x = getItemSpawnPoint();
6       let data = {
7         id: randomId(),
8         x: x,
9         num: getRandomInt(5)
10      }
11      // too many items on the field
12      if (x != -1) {
13        items.push(data.id);
14      }
15      // sending the event to every connected player
16      io.emit('spawnItem', data);
17    }
18  }, 10000);

```

Der Client muss jetzt nur mehr beim Empfangen dieses Events ein Item am richtigen Ort erstellen. Wie Items am Client erstellt werden, findet kann in Kapitel 5.1.1 nachgelesen werden.

Vereinfacht sieht der Code am Client also so aus:

```

1   socket.on('spawnItem', createItem);
2
3   // data consists of item-ID, type of item and position of item
4   function createItem(data) {
5     // item gets created here
6   }

```

Das waren allen Funktionalitäten des Scribble-Fight-Web-Servers. Dieser läuft allerdings nicht mehr lokal, sondern auf einem Kubernetes-Cluster: Der Leocloud. Wie der Server deployed werden konnte, wird in dem nächsten Kapitel beschrieben.

## 5.2 Deployment [R]

Die Grundlage, um etwas auf einen Kubernetes-Cluster deploien zu können, ist ein Docker-Image. Näheres zu der Docker-Technologie wird in Kapitel 4.2.1 beschrieben.

### 5.2.1 Docker-Image [R]

Da es sich bei Scribble-Fight nur um einen Web-Server, der statische Files hostet, handelt, und keine Datenbank beinhaltet ist, ist das Docker-File relativ simpel.

```
FROM node:16

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY ./p5_backend/package*.json ./p5_backend/

WORKDIR /usr/src/app/p5_backend
RUN npm install

# Bundle app source
WORKDIR /usr/src/app
COPY ./p5_backend ./p5_backend
COPY ./p5_frontend ./p5_frontend

EXPOSE 3000
CMD [ "node", "./p5_backend/server.js" ]
```

Abbildung 22: Scribble-Fight Dockerfile

Die Zeilen Code werden im Folgenden genauer erläutert.

1        FROM node:16

Da Docker-Images von anderen Docker-Images erben können, wird nicht ein eigenes Base-Images verwendet, sondern das offizielle Node.js Image, das schon die Tools und Packages hat, die benötigt werden, um eine Node.js-Applikation auszuführen.

1        WORKDIR /usr/src/app

Es wird ein neues Verzeichnis erstellt, von wo aus gearbeitet wird. In diesem Verzeichnis werden alle folgenden Befehle ausgeführt. Der Vorteil davon ist, dass keine absoluten

File-Paths benutzt werden müssen, sondern relativ zu diesem Verzeichnis relativ-Paths benutzt werden können.

```
1      COPY ./p5_backend/package*.json ./p5_backend/
```

Bevor `npm install` ausgeführt werden kann, muss unser `package.json` und `package-lock.json` File auf das Image kopiert werden. Um das machen zu können, wird `COPY` benutzt. Dieser Befehl nimmt zwei Parameter: Source und Destination. Es werden `package.json` und `package-lock.json` in einen neuen Ordner des vorherig erstellen Arbeits-Verzeichnis kopiert.

```
1      WORKDIR /usr/src/app/p5_backend
2      RUN npm install
```

Es wird nun das Verzeichnis, in das das `package.json`-und `package-lock.json`-File kopiert worden sind, als Arbeits-Verzeichnis definiert. Da dort diese Files vorhanden sind, kann `npm install` ausgeführt werden und alle nötigen Dependencies werden heruntergeladen.

```
1      WORKDIR /usr/src/app
2      COPY ./p5_backend ./p5_backend
3      COPY ./p5_frontend ./p5_frontend
```

Nun wird wieder das ursprüngliche Verzeichnis als Arbeits-Verzeichnis definiert. Dorthin werden die Files, die der Server zum Funktionieren braucht und auch die Files für das Frontend, die der Server hostet, kopiert.

```
1      EXPOSE 3000
2      CMD [ "node", "./p5_backend/server.js" ]
```

Zuletzt wird nur mehr der Port, den der Scribble-Fight-Server akzeptiert, nach aussen freigegeben, und mit dem Befehl `CMD` der Server gestartet.

## 5.2.2 Leo-Cloud [R]

Die Leo-Cloud ist eine Cloud-Computing-Umgebung der HTL Leonding. Da nun ein funktionierendes Image des Web-Servers vorhanden ist, kann an dem Deployment an die Leo-Cloud gearbeitet werden.

Zuerst muss das Image, das für das Deployment verwendet werden soll, in das Docker-Registry der HTL Leonding geladen werden. Dazu kann der Befehl `docker login registry.cloud.htl-leonding.ac.at` benutzen, um sich mit den eigenen Anmelde-daten dort einzuloggen. Danach wird das Image einfach mit dem Befehl `docker push`

`registry.cloud.htl-leonding.ac.at/t.rafetseder/<image-name>/<image-version>` hochgeladen.

Um das Deployment umzusetzen, werden drei YAML-Files benötigt. YAML ist eine Daten-Serialisierungs-Sprache, die gerne benutzt wird, um Konfigurations-Files zu schreiben.

Diese drei Files sind:

- Deployment-File
- Service-File
- Ingress-File

In dem Deployment-File kann die Konfigurationen zum eigentlichen Deployment festgelegt werden, zum Beispiel welches Image verwendet werden soll, welche Anzahl an Pods (Linux-Containern) verwendet werden sollen oder welcher Port benutzt werden soll.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: scribble-fight-deployment
spec:
  selector:
    matchLabels:
      app: scribble-fight
  replicas: 1 # tells deployment to run 1 pods matching the template
  template:
    metadata:
      labels:
        app: scribble-fight
    spec:
      containers:
        - name: scribble-fight
          image: registry.cloud.htl-leonding.ac.at/t.rafetseder/scribble-fight/v20
          imagePullPolicy: Always
          resources:
            limits:
              memory: '1Gi'
              cpu: '0.3'
          ports:
            - containerPort: 3000
```

Abbildung 23: Scribble-Fight Deployment.yaml

Auf einem Kubernetes-Cluster laufen viele Pods. Diese müssen in einer Weise verbindet werden. Das ist der Job von Service-Files. Service abstrahiert die IP-Adresse eines Pods zu einem statischen Service-Name, sodass externe Requests zu multiplen Pods geleitet werden können.

Das Weiterleiten von Requests zu dem erwünschten Pod basiert auf der selector-spec am Service, der mit dem Metadaten-Label des Pods zusammenpassen muss.

```
apiVersion: v1
kind: Service
metadata:
  name: scribble-fight-service
spec:
  selector:
    app: scribble-fight
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
```

Abbildung 24: Scribble-Fight Service.yaml

Zusätzlich muss noch konfiguriert werden, wie Services in einem Cluster via IP-Adresse oder URL erreicht werden können. Dazu werden Ingress-Files verwendet. Ein Ingress ist eine Ansammlung von Regeln, die einkommende Verbindungen erlauben, Cluster-Services zu erreichen. Das Ingress-Spec-Field stellt einen Service-Namen und Service-Port bereit, der gemeinsam mit der URL-Route von dem Service exposed wird.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: scribble-fight-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: "/$1"
spec:
  rules:
    - host: student.cloud.htl-leonding.ac.at
      http:
        paths:
          - path: /t.rafetseder/scribble-fight/(.*)$
            pathType: Prefix
            backend:
              service:
                name: scribble-fight-service
                port:
                  number: 3000
```

Abbildung 25: Scribble-Fight Ingress.yaml

## 5.3 Map-Erkennung [H]

Nachdem der Spieler der Lobby beigetreten ist, wird es Zeit für ihn die Spielumgebung zu zeichnen und diese für die Abstimmung freizugeben. Das passiert, indem dieser

einen QR-Code (“Quick Response”), welcher sich in der Mitte der Lobby befindet, mit einem mobilen Endgerät, welches über eine Kamera verfügt, abgescannt. Somit wird er auf eine Webseite geleitet, welche wieder auf die Kamera zugreift. Auf dieser ist bereits eine Objekterkennung vorprogrammiert. Auf die Funktionsweise und auf die technische Umsetzung wird im folgenden ... eingegangen. Wenn der Spieler merkt, dass die Objekterkennung das Blatt erkannt hat, kann dieser ein Bild aufnehmen. Im nächsten Schritt ist es nochmals möglich die erkannten Kanten zu verschieben. Wenn sich der User für das Foto als Map entscheidet, so kann er den ausgewählten Teil in eine top-down-2d-Perspektive umwandeln. Kurzgesagt werden dabei die vier ausgewählten Ecken, welche über die Kanten des Blattes liegen, für eine Perspektiven-Transformation mit OpenCV2 herangezogen. Wenn der Spieler dann so weit ist, kann er dieses Bild beziehungsweise diese Map bestätigen und sendet sie somit zurück an die Lobby. Dann kann die Abstimmung beginnen.

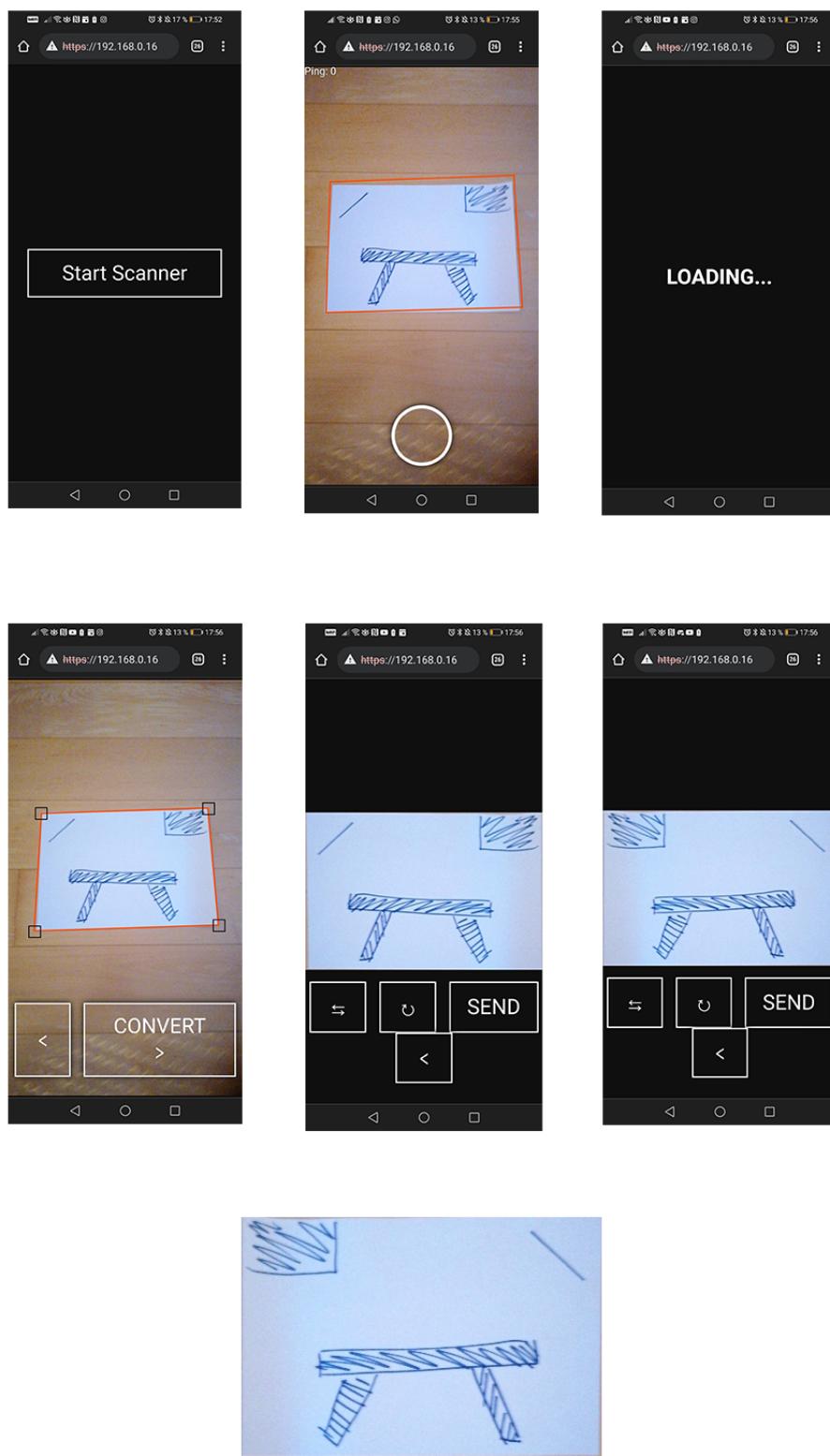


Abbildung 26: Screenshots der Maperkennungsanwendung

### 5.3.1 Objekterkennung [H]

Das Thema “Objekterkennung” ist ein extrem schnell wachsendes und sich verbesserndes Feld in der Welt der Bildverarbeitung beziehungsweise Bildanalyse.

Die Objekterkennung ist ein großer Teil unserer Arbeit. Hier wird aus dem live Footage

der Kamera das Blatt Papier und anschließend in einer fertigen Aufnahme die Map erkannt.

Die Objekterkennung wird zum Beispiel bei komplexen und großen Paketdiensten eingesetzt. Sollen Pakete nach Farbe gruppiert werden, so reicht bereits ein Farbsensor. Das ist in den meisten Fällen jedoch nicht genügend. Vielmals werden Pakete auf Strichcodes oder Adressen überprüft. Wenn also unterschieden werden muss, wo welches Paket landen soll, so ist eine Kamera nötig. Diese muss ein genügend gut auflösendes Bild aufnehmen, dass daraus Informationen gewonnen werden können. Diese Daten müssen zur Auswertung mathematisch beschrieben werden. Zur Übersetzung werden Verfahren, wie Kantenerkennung, Transformationen oder Größen- und Farberkennung, aber auch künstliche Intelligenzen eingesetzt. In dieser Arbeit wurden Kanten zur Objekterkennung herangezogen. Die Korrektheit des Ergebnisses ist abhängig von der Korrektheit der bereitgestellten Daten und wie genau das Objekt mathematisch beschrieben werden kann.

Ein weiteres Beispiel, wo Objekterkennung eingesetzt wird, ist in Fahrerassistenzsysteme oder autonomes Fahren. Bereits in der ersten Stufe des autonomen Fahrens (dem assistierten Fahren) werden zur Hilfe Kameras außerhalb des Fahrzeugs angebracht. In einem Auto, welches als Stufe eins autonomes Assistenzsystem eingestuft wird, werden diese Kameras für einen automatischen Spurhalteassistenten eingesetzt. Via Bildanalyse und Objekterkennung werden die Linien auf der Straße erkannt.

In höheren Stufen ist die Objekterkennung nicht mehr wegzudenken. Es müssen Verkehrsschilder und Personen auf der Straße erkannt werden um daraufhin entsprechend zu agieren.

Andere Beispiele:

- Gesichtserkennung, um das Smartphone zu entsperren.
- Qualitätskontrolle, zur Erkennung und automatischen Entfernung von kaputten oder beschädigten Teilen.
- Personenerkennung, um Menschenmassen zu analysieren.

## Beschreibung der Funktionalität [H]

Wie schon erwähnt kann man aus Bildern Daten erkennen. Viele Pixel bilden insgesamt ein Bild, welches wir visuell aufnehmen und in welchem wir alles Mögliche erkennen können. Jeder einzelne dieser Pixel in einem digitalen Bild enthält Informationen über

die Helligkeit und über die Farbe. Der Bildpunkt setzt sich zusammen aus den Farben Rot, Grün und Blau. Aus diesen drei können alle möglichen Farben abgebildet werden. Bevor ich darauf eingehe, wieso wir uns für die Konturenerkennung als Art der Bilderkennung entschieden haben, ist es noch entscheidend zu wissen, wie der Mensch Objekte erkennt.

In unserem Auge passiert die Farberkennung via den Zapfen. Auch hier wird die Wellenlänge des Lichts in ein Gemisch aus Rot, Grün und Blau aufgeteilt. Allerdings benötigen wir das reine Farbsehen nicht um Objekte zu erkennen. Uns reicht es schon, wenn wir nur Informationen über die Helligkeit empfinden. Somit können wir zum Beispiel auch in Schwarz-Weiß-Bildern oder in der Dunkelheit bei wenig Licht Objekte erkennen. Wir erkennen einen Unterschied zwischen Objekten, welche sich in einem dreidimensionalen Raum befinden leicht, indem das eine Objekt sich in der Helligkeit von dem anderen Objekt unterscheidet. Das heißt, dass wenn z.B. ein Würfel auf einem Boden liegt, dann erkennen wir den Würfel, weil dieser anders viel Licht reflektiert als der Boden. Und an der Stelle, wo der Würfel aufhört, erkennen wir eine Kontur. Hier ist der Unterschied zwischen Boden und Würfel erkennbar.

Und genau so passiert auch die Objekterkennung in unserem Spiel ScribbleFight. Hier wird zuerst aus der Live-View, welche vom Client zur Verfügung gestellt wird, das Farbbild in Graustufen konvertiert. Das passiert, indem für jeden Pixel ein “Durchschnittswert” der Helligkeit von allen drei Farbwerte (Rot, Grün, Blau), welche in einem 8-Bit System von 0 bis 255 geht, gebildet wird. Wie dies jedoch genauer funktioniert, wird im Kapitel Bildverarbeitungsalgorithmen 5.3.2 genauer erklärt.

Listing 22: Alle unnötigen Bilddaten entfernen

```

1     def check(img):
2
3     ...
4
5     # CONVERT IMAGE TO GRAY SCALE
6     imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
7     imgBlur = cv2.GaussianBlur(imgGray, (5, 5), 1)    # ADD GAUSSIAN BLUR
8     upperThres = 40
9     lowerThres = 40
10    imgThreshold = cv2.Canny(
11        imgBlur, upperThres, lowerThres)   # APPLY CANNY BLUR
12
13    ...

```

Nachdem das Bild in ein Schwarz-Weiß-Bild umgewandelt wurde, muss vor der Konturenerkennung noch ein Schwellwert-Bild eines unscharfen Bildes erzeugt werden. In der Untenstehenden Abbildung 27 das dritte Bild von links, oben. Das Schwellwert-Bild dient dazu, maximal viel Information aus dem Bild zu entfernen. Somit bleiben nur

die relevanten Informationen über, welche OpenCV2 braucht um Konturen bilden zu können.

Listing 23: Erhalten von Konturen

```

1      ...
2
3      contours, hierarchy = cv2.findContours(
4          imgThreshold, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE) # FIND ALL
5          CONTOURS
6
7      # FIND THE BIGGEST COUNTOUR
8      accuracy = 25/1000
9      area = 1000
10     biggest, maxArea = biggestContour(
11         contours, accuracy, area) # FIND THE BIGGEST CONTOUR
12
13     ...
14
15     edges = getEdges(oldBiggest, biggest, contours, img, biggestChanged)
16
17     return edges

```

In diesem Code-Block sieht man jetzt wie wir in unserer Diplomarbeit mit OpenCV2 die Konturen aus einem Schwellwert-Bild extrahiert haben und danach die größte Kontur herausgefunden haben. Wie die Funktion “biggestContour(….)” funktioniert ist in diesem Codeblock 24 beschrieben. Und danach wird in “getEdges(….)” ein Numpy-Array von den Eckpunkten erzeugt, in einer anderer Reihenfolge, welche für den Perspektiven Transform relevant ist, umgewandelt und dann als Return-Wert zurückgegeben.

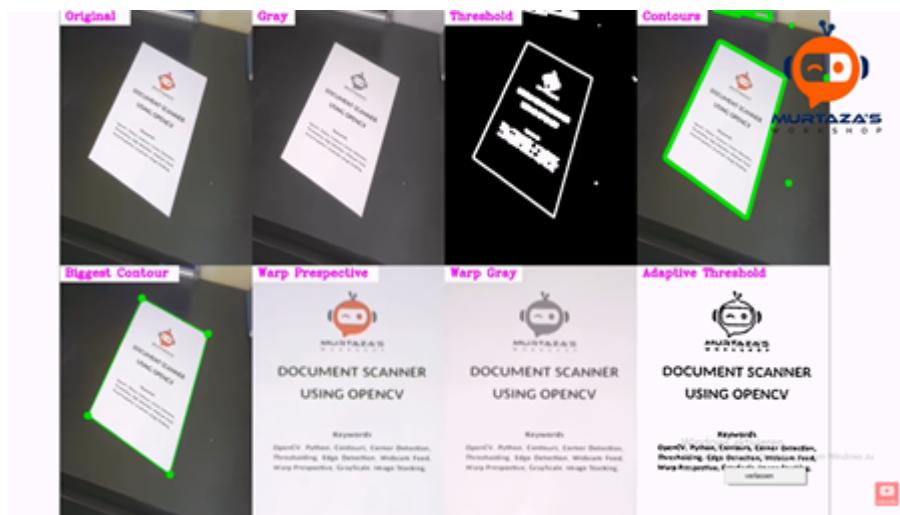


Abbildung 27: Documenten Scanner

Aus diesen Konturen, welche OpenCV2 erkennt, wird die größte, geschlossene Umrandung herausgesucht. Es wird davon ausgegangen, dass sich hier das fokussierte Objekt, also das Blatt Papier, befindet. Um nun die vier Eckpunkte lokalisieren zu können, wird eine vorgefertigte Funktion namens “approxPolyDP” angewendet. Diese

legt, mit den übergebenen Parametern ein Viereck über die Konturen. Danach werden aus diesem Viereck die 4 Eckpunkte extrahiert und als Array wieder an den Client zurückgesendet.

Listing 24: approxPolyDP

```

1  def biggestContour(contours, accuracy, areaVal): # FIND THE BIGGEST CONTOUR
2      biggest = np.array([])
3      max_area = 0
4
5      for i in contours:
6          area = cv2.contourArea(i)
7          if area > areaVal:
8              peri = cv2.arcLength(i, True)
9              # findet die vier Eckpunkte heraus und gibt diese zurueck
10             approx = cv2.approxPolyDP(i, accuracy * peri, True)
11             if area > max_area and len(approx) == 4: # IF == 4 THEN SQUARE
12                 biggest = approx
13                 max_area = area
14
15     return biggest, max_area

```

In der folgenden Funktion namens “getWrappedImg(… )”, wird das Bild, welches aufgenommen wurde in eine zwei-dimensionale top-down Ansicht umgewandelt. Allerdings geschieht dies nur da, wo zuerst die Eckpunkte beziehungsweise das Blatt Papier (also die Spielumgebung) gefunden wurden. Wie dies funktioniert ist im folgenden Codeblock 25 ersichtlich.

Zuerst wird aus dem übergebenen “snipset”-Parameter, welcher als Array vorliegt, die Daten ausgelesen und auf die Variablen “pt\_A” bis “pt\_D” geschrieben. Diese vier Buchstaben geben die Eckpunkte an. Aus diesen Eckpunkten wird im Folgenden dann ein Vektor erzeugt. Für weitere Berechnungen wird dann noch die Länge zwischen den wahrscheinlich parallel zueinanderliegenden Vektoren AD und BC berechnet. Dies ist wichtig, um im späteren Code herauszufinden, in welcher Orientierung das Blatt Papier zu der Kamera liegt.

Listing 25: Bild in Vogelperspektive umwandeln

```

1  def getWrappedImg(img, snipset):
2      snipset = squarify(snipset)
3
4      pt_A = snipset[0]
5      pt_B = snipset[1]
6      pt_C = snipset[2]
7      pt_D = snipset[3]
8
9      lineAB = np.array([pt_A, pt_B])
10     lineBC = np.array([pt_B, pt_C])
11     lineCD = np.array([pt_C, pt_D])
12     lineDA = np.array([pt_D, pt_A])
13
14     width_AD = np.sqrt(((pt_A[0] - pt_D[0]) ** 2) + ((pt_A[1] - pt_D[1]) ** 2))
15     width_BC = np.sqrt(((pt_B[0] - pt_C[0]) ** 2) + ((pt_B[1] - pt_C[1]) ** 2))

```

Im folgenden Code-Block passiert zuerst im Grunde genommen genau dasselbe, wie im zuvor erklärtem Code-Block 25. Hier wird die größere Länge der beiden Vektoren AB und

CD, welche wahrscheinlich in der Realität parallel zueinander liegen, errechnet. Dabei wird angenommen, dass diese beiden Längen die Höhe des Blatt Papiers sind.

```

16     maxHeight = max(int(width_AD), int(width_BC))
17     if maxHeight == width_AD:
18         lineA = lineDA
19     else:
20         lineA = lineBC
21
22     height_AB = np.sqrt(((pt_A[0] - pt_B[0]) ** 2) +
23                           ((pt_A[1] - pt_B[1]) ** 2))
24     height_CD = np.sqrt(((pt_C[0] - pt_D[0]) ** 2) +
25                           ((pt_C[1] - pt_D[1]) ** 2))
26     maxWidth = max(int(height_AB), int(height_CD))
27     if maxWidth == height_AB:
28         lineB = lineAB
29     else:
30         lineB = lineCD

```

Die nun errechnete maximale Höhe wird nun noch mit einem realistischen Faktor multipliziert, um noch ein akkurateeres Ergebnis der Perspektiventransformation erzielen zu können. Dieser “realistische Faktor” wird errechnet, indem der Winkel zwischen den beiden größten orthogonal zueinander liegenden Linien in folgende Formel miteinbezogen wird:

$$factor = \frac{90}{(180 - angle)}$$

Dieser errechnete Faktor staucht die Höhe eines Bildes, welches von der unteren Blattkante aufgenommen wurde und streckt die Höhe von jene, welche, wieso auch immer, von oben Blattkante aufgenommen wurden.

```

31     angle = ang(lineA, lineB)
32     if angle == 90:
33         factor = 1
34     if angle > 90:
35         factor = 90 / (180-angle)
36     if angle < 90:
37         factor = 90 / angle
38
39     maxHeight *= factor

```

Im letzten Teil der Funktion wird lediglich nur noch das Bild in die richtige Perspektive transformiert. Dies wird erzielt, indem man die beiden vorgefertigten Funktionen “cv2.getPerspectiveTransform(...)” und “cv2.warpPerspective(...)” verwendet. Als Return-Wert liefert die genannte Funktion ein Bild mit dem Datentypen OpenCV2, welches dann im späteren Verlauf noch umcodiert werden muss.

```

40     m = np.array([pt_A, pt_B, pt_C, pt_D])
41     m = rotateCW(m)
42     input_pts = np.float32(m)
43     output_pts = np.float32([[0, 0],
44                             [0, maxHeight - 1],
45                             [maxWidth - 1, maxHeight - 1],
46                             [maxWidth - 1, 0]])
47
48     M = cv2.getPerspectiveTransform(input_pts, output_pts)
49
50     dst = cv2.warpPerspective(
51         img, M, (int(maxWidth), int(maxHeight)), flags=cv2.INTER_LINEAR)
52
53     return dst

```

### 5.3.2 Open-CV2 [H]

#### Bildverarbeitungsalgorithmen

Bildverarbeitungsalgorithmen sind Grundbausteine für die Bild- beziehungsweise Map-Erkennung. Ohne diesen können keine, oder nur wenige Daten aus Bildern ausgelesen werden. Somit würde jedes Bild nur Informationen über die Helligkeit und die Farbe/Farbhelligkeit jedes Bildpunktes beinhalten. Deshalb ist es wichtig Bildverarbeitungsalgorithmen zu verwenden um mehr Informationen aus den Bildern, welche dem Python-Server zur Verfügung gestellt werden, zu erkennen und zu erhalten. In dieser Arbeit wurden folgende Methoden von OpenCV2 verwendet, welche das Erkennen des Blatt Papiers ermöglichen. Im Folgenden (5.3.2-5.3.2) wird näher auf die Funktionsweise der einzelnen Algorithmen eingegangen. Diese sind:

- cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY) (“Grayscaling”): Diese Funktion bewirkt eine Graustufenkonvertierung eines gegebenen Bilds.
- cv2.GaussianBlur(...) (“Blurring”): Mit dieser Funktion wird ein gegebenes Bild unscharf gezeichnet und somit unnötig genaue Informationen entfernt.
- cv2.adaptiveThreshold(...) (“Adaptive Thresholding”): Hier wird die Schwellwertbildung adaptiv festgelegt. Es wird das gegebene Bild in Raster unterteilt und für jeden dieser Raster ein Schwellwert gebildet. Somit werden Farbübergänge eliminiert und es bleibt ein reines Schwarz-Weiß-Bild mit harten Kanten über.
- cv2.canny(...) (“Canny” Kantenerkennung): Die Kantenerkennung liefert automatisch erkannte Kanten aus einem gegebenen Bild.
- cv2.approxPolyDP(...) (“ApproxPolyDP”): Diese Funktion vergröbert die erkannten Konturen um einen angegebenen Grad. Die erkannten Konturen werden auf wenige Punkte minimiert. Somit kann zum Beispiel über eine Kontur ein Vieleck gelegt werden.
- cv2.warpPerspective(...) (Perspektiven-Korrektur): Mit “wrapPerspective” kann ein Bild, welches in einer weiteren Dimension aufgenommen wurde und somit einer Perspektive unterlag Perspektiven-korrigiert werden.

Nur einige der verwendeten Bildbearbeitungsalgorithmen sind basierend auf Pixelmanipulation. Andere Algorithmen, wie zum Beispiel “ApproxPolyDP”, sind reines Auslesen von Bildwerten beziehungsweise -daten, welche durch Vorarbeit extrahiert wurden.

## Grayscale

Beim “grayscaleing” werden Informationen über die Lichtintensität aus den einzelnen Pixeln extrahiert und über eine Gewichtung der drei Farben Rot, Grün und Blau gleichmäßig gewertet. Somit entsteht ein Bild, welches rein Informationen über die Helligkeit/Lichtintensität enthält.

Die Lichtintensität ist die Menge an Licht(stärke) pro Pixel. Diese Lichtstärke kann in einem 8-Bit System von den Farben Rot, Grün und Blau von 0-255 reichen. Der Kontrast geht somit von ganz schwarz, was einer Lichtintensität von 0 entspricht (es wird aus diesem Pixel kein Licht emittiert), bis ganz weiß, was in einem 8-Bit System 255 entspricht.

Für eine Umwandlung eines Bilds in dessen Graustufen wird für jeden Pixel also die Lichtintensität der drei Farben (Rot, Grün und Blau) beziehungsweise der Farbkombination dieser drei Farben gemessen und über eine Gewichtung der drei Werte ein “Mittelwert” errechnet. Diese Gewichtung kann über verschiedene Faktoren erfolgen. Daher ist es nicht gegeben, dass auch OpenCV2 die folgende Formel zur Umrechnung eines Bildes, welches aus einem RGB(A)-Farbraum (Rot (R); Grün (G); Blau (B); Opazität (A)) in Graustufen konvertiert werden soll, verwendet. Die Formel dient deswegen nur als Exempel, um alle drei Werte auf eine lineare Gleichgültigkeit für das menschliche Auge zu setzen, um somit ein Graustufenbild zu erzeugen:

$$Y_{linear} = 0,2126 * R_{linear} + 0,7152 * G_{linear} + 0,0722 * B_{linear}$$

Aus dieser Formel ist erkennbar, dass die Farbe Grün mehr gewichtet wird als Rot und Blau. Dies resultiert daraus, dass das Auge die meiste Lichtinformation aus der Farbe Grün extrahieren kann. Somit wirkt auch diese Farbe am hellsten für das menschliche Auge. Deshalb wird sie am höchsten gewichtet, damit ein scheinbar ausgeglichener Wert der Lichtintensität der Farbkombination der drei Farben herauskommt. Dieser gewichtete Wert (Y), welcher auch als Luminanz bezeichnet wird, ist die Basis für die Graustufenkonvertierung.

Im folgendem Code-Beispiel 26 wird die Umsetzung einer Graustufenkonvertierung mittels OpenCV2 in der Programmiersprache Python gezeigt.

Listing 26: Graustufenkonvertierung

```

1      # OpenCV2 used as pixel manipulation library
2      import cv2
```

```

3
4      # read Image from Folder
5      image = cv2.imread('Path/to/your/image.jpg')
6      # Convert RGB image to (2) gray
7      gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
8
9      # display the original image (input) on the screen
10     cv2.imshow('Original image',image)
11     # display the grayscaled image (output of cv2 funktion cvtColor) on the screen
12     cv2.imshow('Gray image', gray)
13
14    cv2.waitKey(0) # waits until a key is pressed
15    cv2.destroyAllWindows() # destroys the window showing image

```

Dieses Stück Programmcode führt zu folgendem Ergebnis:



Abbildung 28: Input



Abbildung 29: Output

## Blurring

Soll ein Bild “geblurred” (engl. für verwischen/verschwommen) werden, so ist das Ziel eine Unschärfe. Ähnlich, wie das “Grayscaleing” ist auch das unscharf-Zeichnen eines Bildes reine Pixelmanipulation. Hier wird jedoch nicht ein einzelner Pixel zur Berechnung der Unschärfe herangezogen, sondern eine zweidimensionale Matrix an umliegenden Bildpunkten. Somit wird jeder Pixel basierend auf dessen benachbarten Bildzellen manipuliert.

Um eine Unschärfe zu erzielen, muss eine mathematische Operation der Faltung einer spezialisierten Matrix, genannt Kernel, auf die Matrix des Bildes angewandt werden. Dabei wird diese Faltungsoperation auch “Konvolution” genannt.

Mathematisch gesehen ist Konvolution, also die Faltung zweier Matrizen, A mit der Größe  $a \times b$  und B mit der Größe  $c \times d$ , eine  $(a + c - 1) \times (b + d - 1)$  Matrix C mit folgenden Einträgen:

$$C_{rs} = \sum_{i+k=r+1, j+l=s+1}^{r=a+c-1, s=b+d-1} A_{ij} B_{kl}$$

Aus dieser Formel ist ersichtlich, dass das Falten beziehungsweise die Konvolution zweier Matrizen A und B eine neue Matrix C, in welcher die Einträgen durch die Summe des Produkts der Einträge der Matrix A mit den am gleich liegenden zweidimensionalen

Indexe entsprechenden Einträgen der anderen Matrix B gebildet werden, erzeugt. Alle diese Produkte werden im zweidimensionalen, also entlang den Zeilen und Spalten, berechnet.

Der zuvor genannte (Begriff) Kernel kann als Matrix beschrieben werden. Diese dient dazu ein Bild in dessen Aussehen zu verändern. Somit ist der Begriff nicht nur auf das Weichzeichnen von Bildern begrenzt. Ein Kernel kann zum Beispiel auch dazu verwendet werden, um Kanten in Bildern zu schärfen.

Da der Kernel eine Matrix mit einem Mittelpunkt ist, muss die Kernel Größe einer ungeraden Zahl, zum Beispiel  $[5 * 5]$ , entsprechen.

In dieser Diplomarbeit wurde häufig der “Gauß’scher” Weichzeichnungsfilter verwendet. Dieser Filter gibt jedem Bildpunkt, welche um einen betrachteten Pixel liegen, eine Gewichtung.

Diese Gewichtung nimmt mit der Distanz zu dem betrachteten Pixel ab. Somit hat theoretisch jeder Pixel in der Bild-Matrix eine eigene Gewichtung. Da diese jedoch mit der Distanz so stark abnehmen werden diese nicht mehr berücksichtigt. Die theoretische Formel zur Berechnung der Gewichte der Pixelwerte rund um einen fokussierten Bildpunkt sieht wie folgt aus:

$$G(x, y) = \frac{1}{2 * \pi * \sigma^2} * e^{-\frac{x^2+y^2}{2*\sigma^2}}$$

Wobei x und y der horizontale und vertikale Abstand zum untersuchten Bildpunkt ist. Sigma ist die Standardabweichung (je höher der Wert von Sigma, desto stärker ist auch der Weichzeichnungseffekt).

In der Realität werden jedoch die Werte dieser “Gaus’schen” Funktion mittels der Werte im Kernel genähert. Dabei macht es wenig Sinn, dass die Kernel-Matrix groß gewählt wird, da der Wert eines Eintrags im Kernel mit der Distanz zum Mittelpunkt abnimmt. Dabei wird oft das  $\sigma$  der Funktion in der Programmierung ignoriert und vom Programm auf Basis der angegebenen Größe des Kernels selbst entschieden. Somit sieht eine Näherung an die “Gaus’sche” Formel durch eine Kernel Matrix wie folgt aus:

$$\frac{1}{16} * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

OpenCV2 stellt jedoch bereits Funktionen zur Verfügung, welche das Kalkulieren des Kernels abnimmt. Im folgendem Codeblock (27) wird gezeigt, wie diese Funktion in dieser Arbeit verwendet wurde.

Listing 27: Gaussian Blur

```

1   import cv2
2   import numpy
3
4   # read image
5   src = cv2.imread('/path/to/image.png', cv2.IMREAD_UNCHANGED)
6
7   # apply gaussian blur on src image
8   # (10,10) is the Kernel size
9   dst = cv2.GaussianBlur(src,(10,10),cv2.BORDER_DEFAULT)
10
11  # display input and output image
12  cv2.imshow("Gaussian Smoothing",numpy.hstack((src, dst)))
13  cv2.waitKey(0) # waits until a key is pressed
14  cv2.destroyAllWindows() # destroys the window showing image

```

Führt man diesen Programmcode aus, so erzeugt dieser folgenden Effekt:



Abbildung 30: Input



Abbildung 31: Output

## Thresholding

Das “Image-Thresholding” also die Schwellwertbildung eines Bilds ist basierend auf die “Image-Segmentation”. Diese erzeugt eine Unterteilung eines Bildes in mehrere Segmente.

Das “Image-Thresholding” ist eine einfache Form der Segmentierung. Das “Thresholding” basiert auf gesetzte Schwellwerte, welches unterschiedlich “intensive” Pixel trennen soll. Dadurch ist das Produkt ein binäres Bild, welches aus Nullen und Einsen besteht (= entweder hell oder dunkel).

“Thresholding” ist, wie auch die Graustufenkonvertierung (Kapitel “Grayscaling” 5.3.2), das Weichzeichnen von Bildern (Kapitel “Blurring” 5.3.2) und viele andere Methoden der Bildverarbeitung, eine Art der Pixelmanipulation. Die Pixel sollen beim “Image-Thresholding” beziehungsweise bei der Schwellwertbildung so angepasst werden,

dass Bilder und deren Pixel einfacher zu analysieren sind. In dieser Arbeit legte das Schwellwert-Bild den Grundbaustein zur Konturen-/Kantenerkennung.

Ziel ist es also, aus einem Farb- beziehungsweise Graustufenbild eine Segmentierung zu erzeugen. Durch dieses Verfahren wird also, wie zuvor erwähnt, ein binäres Bild erzeugt, welches nur noch Schwarz- und Weißwerte beinhaltet. Dies wird zum Beispiel dafür verwendet, dass ein Hintergrund vom Vordergrund getrennt wird. Somit werden sogenannte “Regions Of Interest” erkannt. “Region Of Interest” bedeutet “Bereich von Interesse”. Hierzu werden spezielle Bereiche aus einem Histogramm, also einer Messkurve, ausgewertet beziehungsweise ausgewählt und nach dem zuvor definierten Schwellwert in Gruppen unterteilt. Somit gibt die “Region Of Interest” die fokussierten/für die Auswertung interessanten Bereiche eines Bildes wieder. Dem Hintergrund wird in einem binären Bild der Wert 0 (null) zugewiesen und dem Vordergrund der Wert 1. Diese Werte werden in einem Bild als 0 (null) - schwarz und als 1 – Weiß dargestellt.

Durch das “Thresholding” gibt es die Möglichkeit ein Histogramm zu erstellen, um die Helligkeitswerte eines Bildes zu analysieren. Wenn man aus diesem Graustufenbild (32) ein Histogramm (33) erzeugt, sieht dies wie folgt aus:



Abbildung 32: Input

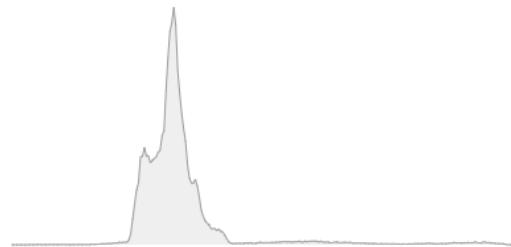


Abbildung 33: Helligkeitsverteilung

Das Histogramm veranschaulicht die Helligkeitsverteilung aller Pixel von den Werten 0 (null) (schwarz), welcher der Wert ganz Links auf der X-Achse im Histogramm ist, bis 255 (weiß), welches dem äußersten X-Wert entspricht. Auf der Y-Achse befindet sich die Häufigkeit der vorkommenden Pixel mit der entsprechenden Helligkeit. Man kann also erkennen und analysieren, dass überwiegend Hintergrund (dunkle Helligkeitswerte) in diesem Bild existiert. Damit aus diesem Histogramm auch ein Schwellwert-Bild (das binäre Bild) erzeugt werden kann, wird ein zuvor definierter Schwellwert zur Verarbeitung der Pixel herangezogen. Somit werden global (!) nur alle Pixel über einem gewissen Helligkeitswert in den Wert 1 (Vordergrund) umgewandelt. Alle anderen bleiben Schwarz.

Sieht man sich nun folgendes Bild (35) an, kann man durch die Schwellwertbildung erkennen, dass sich das interessante Objekt in der Bildmitte befindet:



Abbildung 34: Input



Abbildung 35: Output

Für dieses Bild wurde ein globaler Schwellwert verwendet. Dies ist jedoch keine gute Entscheidung, wenn der Helligkeitseinfall im Bild unterschiedlich stark ist. In unserer Arbeit wurde daher ein adaptiver Schwellwert verwendet. Dieser wird somit durch einen Algorithmus für einen Pixel basierend auf einem kleinen Bereich um ihn herum gebildet. Dadurch werden unterschiedliche Schwellwerte in unterschiedlichen Regionen des Bildes gebildet, um eine ungleichmäßige Beleuchtung zu korrigieren.

Programmiertechnisch wurde dies wie folgt umgesetzt:

Listing 28: Adaptive Gaussian Thresholding

```

1      # Python program to illustrate
2      # adaptive thresholding type on an image
3
4      # organizing imports
5      import cv2
6      import numpy as np
7
8      # path to input image is specified and
9      # image is loaded with imread command
10     image1 = cv2.imread(
11         'path/to/img.png')
12
13     # cv2.cvtColor is applied over the
14     # image input with applied parameters
15     # to convert the image in grayscale
16     img = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
17
18     # applying gaussian thresholding
19     # technique on the input image
20     thresh = cv2.adaptiveThreshold(
21         img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 5, 4)
22
23     cv2.imshow('Adaptive Gaussian', thresh)
24
25
26     # De-allocate any associated memory usage
27     if cv2.waitKey(0) & 0xff == 27:
28         cv2.destroyAllWindows()

```

Dieser Code erzeugt aus einem Input-Bild (36) folgendes Schwellwert-Bild (37), in welchem sehr genau erkennbar ist, was den Vordergrund bildet.



Abbildung 36: Input

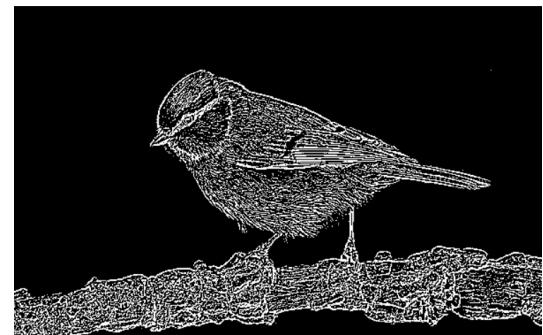


Abbildung 37: Output

### Detecting Contours

Kantenerkennung ist wie die Graustufenkonvertierung (5.3.2) oder das Weichzeichnen (5.3.2) ein Bildverarbeitungsalgorithmus, welcher dazu verwendet wird, um Diskontinuitäten zu identifizieren. Hierzu werden scharfe Änderungen der Bildfarbe/der Bildintensität herausgefiltert. Punkte, wo dieser Wert stark variiert, wird als Kante abgestempelt.

Das Kanten erkennen schafft eine Basis für die Bildverarbeitung und Form Erkennung der Bild- und Objekterkennung. Für die Konturenerkennung sind jedoch nur geschlossene Kanten von Bedeutung.

Die Kantenerkennung funktioniert wie folgt. Für die “Edge-Detection” (Kantenerkennung) ist wie beim Weichzeichnungsalgorithmus (5.3.2) eine Konvolution nötig; vor allem, um hochauflösende Bilder zu verarbeiten. Oft wird dazu der sogenannte “Sobel”-Operator herangezogen. Dieser erzeugt für die Kantenerkennung zwei Kernel, welche in x- und y-Richtung unterteilt sind. Der Kernel in x-Richtung sieht wie folgt aus:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Hier sind negative Nummern links und positive Nummern rechts. Wegen der 0 (null) in der Mitte des Kernels wird nur geprüft, ob eine vertikale Linie/Kontur nach unten existiert. Daher wird mit diesem Kernel nur in x-Richtung auf eine Kante überprüft. Die mittigen Pixel auf der y-Achse des Kernels werden durch den “Sobel”-Operator höher gewichtet als die Randpixel.

Das Ziel ist dabei einen Unterschied zwischen der Farb-/Lichtintensität des linksseitigen Bereichs des Bilds (der Pixel auf der linken Seite) und der Intensität des rechtsseitigen Bereichs des Bilds (der Pixel auf der rechten Seite) herauszufinden.

Anhand eines Beispiels wird diese Operation klarer:

$$\begin{bmatrix} 0 & 0 & 255 & 255 \\ 0 & 0 & 255 & 255 \\ 0 & 0 & 255 & 255 \\ 0 & 0 & 255 & 255 \end{bmatrix}$$

Hier existiert eine für einen Menschen klar erkennbare Kante; jedoch tut sich bei dem Erkennen die Maschine etwas schwerer. Dazu muss der zuvor genannte Kernel über die Pixel gelegt werden. So erhält man durch die Summe der Multiplikationen der einzelnen Werte des Kernels mit den entsprechenden Werten der Matrix des Bildes. Diese ergibt in diesem Beispiel 1025. Ist dieser Wert über einen zuvor definierten Schwellwert, so existiert an dieser Position eine Kante.

Für die Konturenerkennung wird ein Bereich von geschlossenen Kanten herangezogen. Diese geschlossenen Kanten werden erkannt, indem eine Kante in einem zuvor definierten Kernel rund um eine weitere Kante existiert. Führt dieser Pfad wieder zurück zur Ausgangskante, so kann mit Sicherheit gesagt werden, dass es sich um eine Geschlossene Kontur handelt.

Listing 29: Adaptive Gaussian Thresholding

```

1  import cv2
2  import numpy as np
3
4
5  # Let's load a simple image with 3 black squares
6  image = cv2.imread(
7      'path/to/image.png')
8  cv2.waitKey(0)
9
10 # Grayscale
11 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
12
13 # Find Canny edges
14 edged = cv2.Canny(gray, 30, 200)
15 cv2.waitKey(0)
16
17 # Finding Contours
18 # Use a copy of the image e.g. edged.copy()
19 # since findContours alters the image
20 contours, hierarchy = cv2.cv2.findContours(
21     edged, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
22
23 cv2.imshow('Canny Edges After Contouring', edged)
24 cv2.waitKey(0)
25
26 print("Number of Contours found = " + str(len(contours)))
27

```

```

28 # Draw all contours
29 # -1 signifies drawing all contours
30 cv2.drawContours(image, contours, -1, (255, 255, 0), 1)
31
32 cv2.imshow('Contours', image)
33 cv2.waitKey(0)
34 cv2.destroyAllWindows()

```

Dieser Programmcode erzeugt folgendes Bild:



Abbildung 38: Input



Abbildung 39: Output

### ApproxPolyDP

Um nun aus den gewonnenen Konturen ein Blatt Papier zu erkennen, müssen die vier Eckpunkte des Blattes herausgefunden werden für eine Perspektiventransformation. Diese vier Punkte werden über die Open-Computer-Vision-Funktion “approxPolyDP(...)” (“approxymation of Polygon Points” englisch für Annäherung von Polygonen) gewonnen. Die Funktion gibt aus einem übergebenen Konturen-Array (Array = Ansammlung an Daten; in diesem Fall eine Sammlung an Positionsdaten der Kontur-Eckpunkte), eine Annäherung der Eckpunkte zurück. In dem Feld werden die Punkte der Konturen als Vektoren angegeben. Die Annäherung an diese Punkte erfolgt mit einer übergebenen Genauigkeit, welche als Parameter der Funktion spezifiziert wird.

Diese Kontur Näherung, welche den “Ramer–Douglas–Peucker” Algorithmus (abgekürzt RDP.; Abbildung 40) verwendet, hat das Ziel eine Kontur, also eine Polygon-Line, zu vereinfachen. Dies passiert, indem der Algorithmus die Scheitelpunkte der Kontur bei einem gegebenen Schwellwert reduziert.

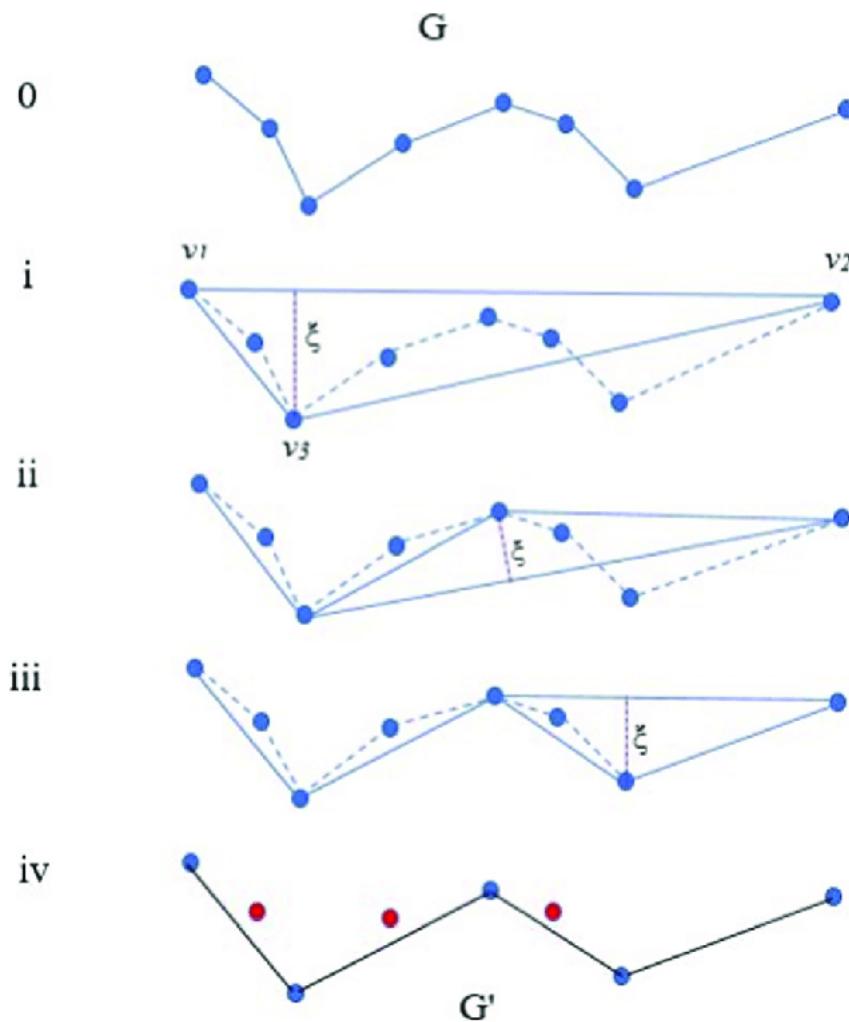


Abbildung 40: Der Ramer–Douglas–Peucker Algorithmus

Wenn die Start- und Endpunkte einer Kurve gegeben sind, findet der Algorithmus zuerst den am weitest entfernten Punkt (Endpunkt). Dieser nimmt dann mit einem gegebenen Schwellwert  $x$  einen Bruchteil, welcher zu der Anzahl  $x$  an Polygonpunkten führt, der originalen Kontur her. Der gefragte Punkt, welcher sich an der Stelle des Bruchteils befindet, wird als neuer Konturpunkt herangezogen. Wenn andere Konturpunkte dazwischen liegen, so werden diese für den neuen, vereinfachten Konturen-Array ignoriert beziehungsweise eliminiert. Somit werden bestimmte Vertices (Eckpunkte) systematisch eliminiert.

Über bleibt eine Annäherung an eine Ausgangskurve. Somit bleibt am Ende genügend Information über, um zum Beispiel ein Blatt Papier erkennen zu können, was das Hauptziel von diesem Teil der Arbeit (Maperkennung) war. Dabei werden so viel wie möglich Daten aus den Konturen entfernt und die Verarbeitung kann schneller passieren.

Dieses Verfahren wird auch als Kurvenglättung bezeichnet.

Außerdem werden maximal vier Punkte für eine Perspektiventransformation herangezogen.

Das heißt, dass die Scheitelpunkte einer Kurve reduziert werden und dabei trotzdem eine große Menge beziehungsweise Genauigkeit an Information erhalten bleibt. Somit bleibt ein Großteil der Form beibehalten.

Somit werden die vier Eckpunkte des Blatt Papiers in dieser Arbeit herausgefunden. Diese vier Eckpunkte sind eine sehr genaue Annäherung an die Eckpunkte, so wie diese in der Realität vorzufinden sind.

Wie dies mit Python und OpenCV2 umgesetzt wurde, wird im folgenden Codebeispiel ersichtlich:

Listing 30: ApproxPolyDP Beispiel

```

1      #importing the module cv2
2      import cv2
3
4      #reading the image whose shape is to be detected using imread() function
5      imageread = cv2.imread('Path/to/image.png')
6
7      #converting the input image to grayscale image using cvtColor() function
8      imagegray = cv2.cvtColor(imageread, cv2.COLOR_BGR2GRAY)
9
10     #using threshold() function to convert the grayscale image to binary image
11     _, imagethreshold = cv2.threshold(imagegray, 245, 255, cv2.THRESH_BINARY_INV)
12
13     #finding the contours in the given image using findContours() function
14     imagecontours, _ = cv2.findContours(imagethreshold, cv2.RETR_TREE,
15                                         cv2.CHAIN_APPROX_SIMPLE)
16
17     #for each of the contours detected, the shape of the contours is approximated
18     #using approxPolyDP() function and the contours are drawn in the image
19     #using drawContours() function
20     for count in imagecontours:
21         epsilon = 0.01 * cv2.arcLength(count, True)
22         approximations = cv2.approxPolyDP(count, epsilon, True)
23         cv2.drawContours(imageread, [approximations], 0, (0), 3)
24
25     #displaying the resulting image as the output on the screen
26     cv2.imshow("Resulting_image", imageread)
27     cv2.waitKey(0)
```

Dieser Programmcode führt zu folgendem Ergebnis:

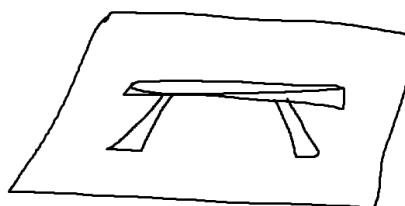


Abbildung 41: Input

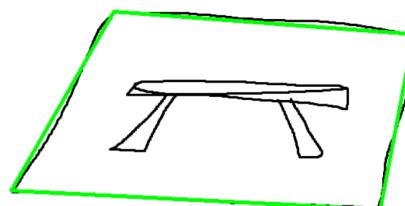


Abbildung 42: Erster Output des Programmcodes

## Wrap Perspective

“Warp Perspective” ist Englisch und bedeutet “Perspektiventransformation”. Der Bildverarbeitungsalgorithmus wird zur Perspektivenveränderung herangezogen. Dieser Algorithmus ist der letzte Schritt der Bildverarbeitung, welcher nötig ist, um das Blatt Papier, auf welchem eine Map gezeichnet ist, aus einem Bild zu erkennen.

Durch die Perspektiventransformation werden Winkel, Parallelität und Längen verändert, wodurch es zu einer Stauchung beziehungsweise Streckung eines Bildes kommt.

Der Algorithmus zur Transformation der Perspektive kann wie folgt beschrieben werden:

$$\begin{bmatrix} ti\_x' \\ ti\_y' \\ ti \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ c_1 & c_2 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Auf der rechten Seite der Gleichung befindet sich die Transformationsmatrix als erster Faktor und der Input  $x$  und  $y$  als Vektor und 1 als zweiter Faktor.

Das Produkt daraus, also die linke Seite der Gleichung, gibt den Skalierungsfaktor an. Das heißt der Perspektiven-transformierte Punkte mit den Koordinaten  $x$  und  $y$  wird zu einem neuen Punkt  $P'$  mit den Koordinaten  $x'$  und  $y'$ .

Die Transformationsmatrix ist eine Kombination aus folgenden Werten:

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \rightarrow \text{Transformation hinsichtlich Rotation, Skalierung, etc.}$$

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \rightarrow \text{Verschiebungsvektor}$$

$$\begin{bmatrix} c_1 & c_2 \end{bmatrix} \rightarrow \text{Projektionsvektor}$$

Die acht Unbekannten in der Transformationsmatrix geben die Freiheitsgrade im Raum an.

Um die Matrix kalkulieren zu können müssen zuerst die vier Punkte der Eckpunktnäherung durch “approxPolyDp” herangezogen werden. Diese Funktion wurde im Kapitel 5.3.2 erklärt. Die vier Eckpunkte, welche in die Vogelperspektive transformiert wurden, sollen ein neues Rechteck darstellen. Das heißt, dass die vier Punkte, welche zuvor im

dreidimensionalen Raum die vier Ecken des Blatt Papiers darstellten, werden nun in ein Rechteck perspektiventransformiert. Dadurch wird das Bild in eine Perspektive, welche einen Blick von “oben” ermöglicht, umgewandelt, und richtig sakliert.

Somit hat man acht Unbekannte und acht Gleichungen wodurch die Gleichungen in der Matrix-Form gelöst werden können. In OpenCV2 passiert diese Matrixkalkulation via der Funktion “cv2.getPerspectiveTransform(points, points’’). Der erste Parameter stellt die Koordinaten im Ausgangsbild dar und der zweite stellt die Koordinaten im Ausgabebild dar.

Danach wird “cv2.wrapPerspective(… )” angewendet, um die Perspektive anhand der gegebenen Matrix zu verändern. Dafür wird aus dem Ausgangsbild ein Teil weggeschnitten und über bleibt der für das “use-case” (Anwendungsfall) interessante Teil. Es wird nur jenes Stück aus dem Bild verwendet, welches in den vier gewählten Eckpunkten liegt. Dieses wird in die Vogelperspektive umgewandelt.

Folgender Programmcode zeigt die Matrixrechnung anhand der Bibliothek OpenCV2 und der dazugehörigen Funktionen:

Listing 31: Perspektiventransformation Beispiel

```

1  #importing the module cv2 and numpy
2  import cv2
3  import numpy as np
4
5  #reading the image whose perspective is to be changed
6  imageread = cv2.imread('C:/Users/admin/Desktop/plane.jpg')
7
8  #specifying the points in the source image which is to be transformed to the
   corresponding points in the destination image
9  points1 = np.float32([[0, 100], [700, 260], [0, 700], [700, 400]])
10 points2 = np.float32([[0, 200], [600, 0], [0, 700], [1000, 700]])
11
12 #applying getPerspectiveTransform() function to transform the perspective of
   the given source image to the corresponding points in the destination image
13 resultimage = cv2.getPerspectiveTransform(points1, points2)
14
15 #applying warpPerspective() function to fit the size of the resulting image
   from getPerspectiveTransform() function to the size of source image
16 finalimage = cv2.warpPerspective(imageread, resultimage, (500, 600))
17
18 #displaying the original image and the transformed image as the output on the
   screen
19 cv2.imshow('Source_image', imageread)
20 cv2.imshow('Destination_image', finalimage)
21 cv2.waitKey(0)
22 cv2.destroyAllWindows()
```

Dieser Programmcode führt zu folgendem Ergebnis:



Abbildung 43: Input

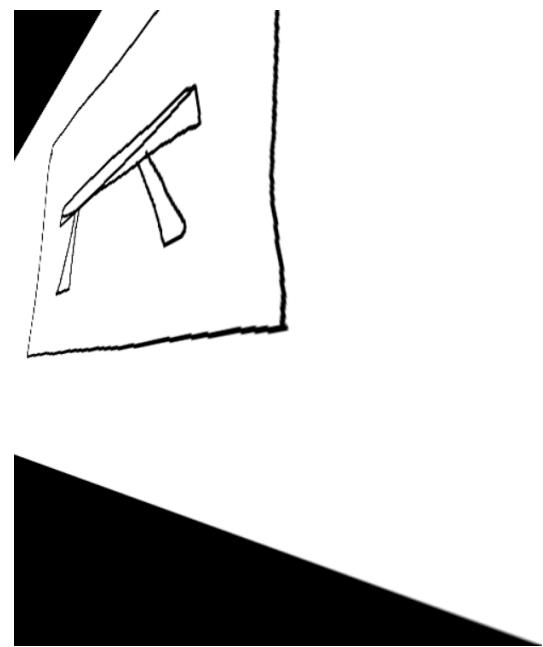


Abbildung 44: perspektiventransformiert

### **Umwandlung der Bilder in für das Spiel brauchbare Daten [H]**

Hat man nun das Bild Perspektiven transformiert, so kann man dies mit dem Send-Button (Wie in Abbildung 26 ersichtlich) bestätigen. Somit wird das Bild an den Python-Server ein letztes Mal zurückgesendet und es wird eine spielbare Map daraus generiert. Wie dies Funktioniert wird in den folgenden Codeblöcken ersichtlich.

Zuerst wird sich wieder darum gekümmert, dass die Daten so stark minimiert werden, dass sie trotzdem noch brauchbar sind, aber so viel wie möglich Informationen beinhalten. Das heißt: so wenig wie möglich Bildmaterial soll so viel wie möglich Information enthalten. Doch davor wird das übergebene Bild in den RGBA-Farbraum umgewandelt und in ein OpenCV2-Bild konvertiert. Im RGBA Farbraum ist es zusätzlich noch möglich neben den Farbwerten auch Informationen über die Transparenz des Bildes abzuspeichern. Dann startet wieder dasselbe Prozedere, wie bei der Perspektiven Transformation.

Zuerst wird das Bild in Graustufen konvertiert. Wie dies funktioniert ist bereits im Unterpunkt 5.3.2 erklärt. Dann wird es unscharf gezeichnet und danach ein adaptiver Threshold (ein Schwellwert-Bild) daraus generiert.

Im nächsten Schritt wird die vorgefertigte Funktion “Bitwise\_not” angewendet. Hier wird das Bild in nur schwarze und weiße Pixel umgewandelt. Sie wandelt also alle Pixel

entweder in schwarze oder weiße Bildpunkte um und generiert daraus wieder ein neues OpenCV2 Bild. Auch dieses Bild wird wieder unscharf gezeichnet.

Was dann noch für das Spiel wichtig ist, ist, dass das Bild in einen Quader umgewandelt wird. Das heißt, dass das Schwellwert-Bild so umgewandelt wird, dass alle Seiten (Höhe und Breite) gleich lang sind. Die leeren Flächen, die daraus entstehen werden mit transparenten Pixeln gefüllt.

Die Zeile zwanzig dient dazu das Bild testweise in einem Ordner abzuspeichern.

Listing 32: Bild in Spielbare Map umwandeln

```

1      def getPlayableArray(img):
2          np.set_printoptions(threshold=sys.maxsize)
3
4          alpha_img = cv2.cvtColor(img, cv2.COLOR_BGR2BGRA) # rgba
5          imgWarpGray = cv2.cvtColor(alpha_img, cv2.COLOR_BGR2GRAY)
6          blurred = cv2.GaussianBlur(imgWarpGray, (7, 7), 0)
7          imgAdaptiveThre = cv2.adaptiveThreshold(
8              blurred, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV,
9              7, 2)
10         imgAdaptiveThre = cv2.bitwise_not(imgAdaptiveThre)
11         imgAdaptiveThre = cv2.medianBlur(imgAdaptiveThre, 3)
12
13         # make image square
14         imgAdaptiveThre = np.array(makeSquare(
15             cv2.cvtColor(imgAdaptiveThre, cv2.COLOR_BGR2BGRA)))
16
17         img = cv2.cvtColor(imgAdaptiveThre, cv2.COLOR_BGR2BGRA)
18
19         # pippoRGBA2 = Image.fromarray(np.array(img).astype('uint8'), mode='RGBA')
20         # pippoRGBA2.show()
21         cv2.imwrite('./source/prototypes/streamFusion/output/imgAdaptiveThre.png',
22                     imgAdaptiveThre)

```

Im nächsten Schritt wird der Array berechnet, welcher als Grundlage für die spielbare Map dient und aus dem aufgenommenen Bild extrahiert wird. Der Array beinhaltet nach der Verarbeitung schwarze und transparente Bildpunkte. Die schwarzen Pixel symbolisieren jene Plätze, auf welchen sich der Spieler schlussendlich bewegen kann. Die transparenten Pixel sind jene, welche als Hintergrund erkannt wurden und somit nicht für den Spieler begehbar sein sollten. Wie sich jedoch der Spieler auf diesen Punkten bewegen kann und wie der Array in die Spiellogik implementiert wird, wird im Kapitel “Erstellung der Spielumgebung 5.1.1” von Rafetseder Tobias näher erklärt.

Jedoch wird zuerst eben das transformierte Bild in den Array umgewandelt. Dafür wird aus diesem adaptiven Threshold-Bild (Schwellwert-Bild) ein Faktor ausgerechnet, mit welchem die Bildpunkte zusammengefasst werden müssen, um insgesamt nicht mehr als  $\approx 3025$  Pixel zu überschreiten. Die dabei verwendete Formel wird im Folgenden an einem einfachen Beispiel erklärt. Diese lautet:

$$n \approx \text{math.ceil}\left(\sqrt{\frac{\text{rows} * \text{columns}}{\text{meshes}}}\right)$$

Setzt man nun eine Bildbreite und eine Bildhöhe in “rows” und “columns” ein, so erhält man den ganzzahligen Faktor, mit welchem die angegebene Bildbreite und Höhe multipliziert werden muss, um maximal  $\approx 3025$  Pixel zu erreichen. Angenommen also, das Bild ist 1920x1080 Pixel groß, so würde das eine Anzahl von 2.073.600 Pixel liefern. Setzt man diese Werte nun in die Formel ein, so erhält man den Faktor 27. Dividiert man nun die beiden Bildmaße mit diesem Wert, so erhält man ein Bild, welches 72x40 groß ist und somit eine Anzahl von 2880 Pixel liefert und somit weit unter den geforderten 3025 Pixeln liegt.

Der Grund, warum der Array eine maximale Größe von 3025 Einträgen hat, ist, weil p5.js, die Bibliothek, welche wir verwenden, um die Spielumgebung aufzubauen nur ein Maximum von 1000 Meshes generieren kann, bevor das Spiel zu ruckeln beginnt. Und wir gehen davon aus, dass man nie mehr als ein Drittel seines Blattes vollzeichnen wird.

Dann wird in einer zweidimensionalen Schleife das Bild durchgegangen und jeweils alle n Pixel ein Mittelwert errechnen über die Farbwerte des Bildes errechnet und somit der Hintergrund vom Vordergrund getrennt. Alle n Pixel werden zusammengefasst. Wenn in diesem Code also mehr als 95% der Pixel Hintergrund darstellen, so wird der Pixel auch als Hintergrund gewertet (es wird also in den Map-Array ein transparenter Pixel an der Stelle x/n und y/n gepusht). Ansonsten wird ein Schwarzer Pixel (als Map erkannt) dem Map-Array hinzugefügt. An dem Punkt mit den Koordinaten x/n und y/n befindet sich also der neue Bildpunkt, da das Bild ja komprimiert werden soll.

Dieser Map-Array wird dann in ein File abgespeichert. Dieses File dient rein dazu, um den Array debuggen zu können. Darunter wird ein Bild, welches aus diesem Map-Array generiert worden ist, im angegebenen Pfad gespeichert.

```

21     iar = np.asarray(img).tolist()
22
23     rows = len(iar)
24     columns = len(iar[0])
25
26     meshes = 3025
27     # percent = perc(rows * columns)
28     percent = 95
29     n = math.ceil(np.sqrt(rows * columns / meshes))
30     x = 0
31     y = 0
32
33     newImg = []
34
35     while y < rows:
36         newImg.append([])
37         while x < columns:
38
39             i = 0
40             j = 0
41             bg = 0
42             while i < n:
43                 while j < n:
44                     if (y + j) < rows and (x + i) < columns:

```

```

45             if np.all(iar[y + j][x + i][:3] == [255, 255, 255], 0):
46                 bg += 1
47             else:
48                 bg += 1
49                 j += 1
50             j = 0
51             i += 1
52
53             bgPercent = bg / (n**2)
54             if (bgPercent < (percent / 100)):
55                 newImg[int(y / n)].append([0, 0, 0, 255])
56             else:
57                 newImg[int(y / n)].append([255, 255, 255, 0])
58
59             x += n
60             x = 0
61             y += n
62
63             iar = np.asarray(newImg).tolist()
64             with open('./source/prototypes/streamFusion/output/mapArray.txt', 'w') as
65                 f:
66                     f.writelines(repr(iar))
67
68             # pippoRGBA2 = Image.fromarray(np.array(newImg).astype('uint8'),
69             #                             mode='RGBA')
70             # pippoRGBA2.show()
71             cv2.imwrite(
72                 './source/prototypes/streamFusion/output/newImg.png', np.array(newImg))
73
74     return newImg

```

### 5.3.3 Kommunikation mit der Lobby via Flask und Flask SocketIO [W]

## 5.4 KI [H]

Die Vorstellung von Künstlichen Intelligenzen gibt es schon seit Mitte des zwanzigsten Jahrhunderts. Bei einer Konferenz von Wissenschaftlern am Dartmouth College im US-Bundesstaat New Hampshire im Jahr 1956 wird das erste Mal bewiesen, dass es möglich ist mit Mathematik menschliche Intelligenz zu simulieren.

Der Gedanke, dass eine von Menschenhand erschaffene Intelligenz, welche keine biologische Abstammung hat und den Menschen bei der Arbeit unterstützen, oder diese ganz abnehmen kann, wird seit diesem Ereignis angestrebt. Dabei ist das Feld, in welchem “Artificial Intelligence” (AI) beziehungsweise Künstliche Intelligenz (KI) eingesetzt werden kann, scheinbar unbegrenzt.

Das Einzige, das gegeben sein muss, ist, dass in einem komplexen System, in welcher die Künstliche Intelligenz operiert, Daten extrahiert und gesammelt werden können. Weiters sind Ressourcen, wie Hardware oder Zeit, ein wichtiger Aspekt. Die intelligente Maschine kann mehrere Millionen Schritte pro Tag ausführen und mit mehreren “Gehirnen” (Mehrere Instanzen der Lernumfelds) gleichzeitig lernen. Trotzdem ist dies limitiert, durch die Geschwindigkeit und der Rechenleistung, die zu diesem Zeitpunkt aufgebracht

werden kann, mit welcher die Aktionen ausgeführt werden und somit zu einem “Lernen” führen.

Die zuvor genannte Lerntechnik “Reinforcement Learning” (Kapitel 4.3.5), funktioniert ähnlich wie die Lern-Art von biologischen Organismen. Die Idee stammt davon, dass simple, Lebensformen, welche nicht mehr als Instinkte zum Futter sammeln, Prädatoren ausweichen und im Generellen zum Überleben haben, sollten von einer Maschine “leicht” simuliert werden können. Ein Beispiel hierfür wäre ein Fadenwurm. Das ZNS (Zentrale Nervensystem) baut sich aus dreihundertzwei Nervenzellen zusammen. Weitere chemische Verbindungsstellen und Kanäle erhöhen die Komplexität dieses Lebewesens nur wenig. Dies ist mit modernen Computern bereits simulierbar.

Dazu müssen jedoch gewisse mathematische Modelle angewandt werden, um dieses Ergebnis zu erreichen und den Organismus zu simulieren.

Dass und wie es für eine Maschine möglich ist zu lernen wurde bereits im Kapitel 4.3.6 näher erläutert.

Um diese mathematischen Modelle jedoch perfekt auf ein gewünschtes Ergebnis abstimmen zu können, wird Hyperparameter-“Fine-Tuning” angewendet. Dabei werden jene Parameter, welche zu einem Lernen beitragen, so angepasst, dass das Modell eine perfekte Lernkurve erzielt.

Im Zuge dieser Arbeit wurde klar, dass Künstliche Intelligenz kein Thema ist, welches leicht zu begreifen ist.

Diese zuvor erwähnten mathematischen Modelle wurden bereits in einer Python Bibliothek namens “Stable\\_baselines3” implementiert. Dieses Framework wird im Folgenden (Kapitel: 5.4.1) genauer erklärt.

Die zur Verfügung gestellten Algorithmen sind jedoch nicht perfekt auf unseren Anwendungsfall abgestimmt. Trotzdem wurden Ergebnisse erzielt (auch, wenn diese nicht bemerkenswert sind).

Die Logik der Welt, in welcher sich der Agent (Reinforcement-Learning) bewegt, wurde mittels Open-AI-Gym umgesetzt. Dabei handelt es sich um eine Hilfestellung, welche es ermöglicht objektorientiert eine Künstliche Intelligenz mit Python umzusetzen. Die Syntax beziehungsweise Vorgehensweise, welche verwendet wird, um eine Künstliche Intelligenz, welche mit der Methode des verstärkten Lernens trainiert, mittels Python

zu implementieren, wird in den folgenden Kapiteln genauer beschrieben. Dabei wird auch genauer auf die genannten Bibliotheken eingegangen.

### 5.4.1 Verwendete Bibliotheken [H]

“OpenAI-Gym” ist dafür da Reinforcement-Learning Algorithmen auf einem Beginner Niveau beizubringen und diese hauptsächlich zum Ausprobieren und Vergleichen zur Verfügung zu stellen.

Dabei erstellt “OpenAI-Gym” viele vorab entwickelte Videospiele zur Verfügung. Diese Spiele sind hauptsächlich Atari-Games, wie zum Beispiel Tetris, Pac-Man, oder Breakout. Atari ist eine Spielkonsole aus den 1980‘ Jahren.

Diese Bibliothek stellt jedoch nicht nur zweidimensionale Umwelten zu Verfügung. Auch einige dreidimensionale Umgebungen, wie “Humanoid”, bei welchem man einer Figur das Gehen beibringen muss, sind zur Auswahl.

Somit ist es für jeden möglich sich spielerisch mit dem Thema der Künstlichen Intelligenz auseinanderzusetzen. Es ist bereits mit einigen, wenigen Code-Zeilen möglich die “Library” einzubinden und ein simples Programm umzusetzen.

Listing 33: Simples OpenAI-Gym Programm

```

1      # import dependencies
2      import gym
3
4      # create CartPole environment
5      env = gym.make('CartPole-v0')
6
7      # reset the environment
8      env.reset()
9
10     # walk 1000 steps
11     for _ in range(1000):
12         # display progress visually
13         env.render()
14         # take a random action
15         env.step(env.action_space.sample())
16
17     # close the environment after it is done
18     env.close()
```

In diesem Environment (Spielumgebung) wird versucht ein Stab mithilfe einer beweglichen Plattform zu balancieren. Die Plattform kann sich dabei nur in einer Dimension (auf der X-Achse) bewegen. Mit diesem Beispielcode (33) wird jedoch noch keine Künstliche Intelligenz trainiert. Hier werden zunächst zufällige Aktionen in dem Spiel ausgeführt, was selbstverständlich noch nicht zu einem gewünschten Ergebnis führt. Die Schwierigkeit dabei ist, dass sich die Plattform nicht nach links bewegt, wenn der

Stab nach links fällt, oder umgekehrt, sondern den Stab wirklich balanciert. Diesen Ablauf der Bewegung findet der Algorithmus binnen weniger Sekunden heraus.

Es ist jedoch möglich dieses Programm so zu erweitern, dass dieses auch dazulernt.

Außerdem kann man mit “OpenAI-Gym” ein eigenes Spiel erstellen. Dies funktioniert objektorientiert. Dabei müssen folgende Funktionen eingebunden werden:

1. `__init__(self)`: diese Funktion wird aufgerufen, wenn ein Objekt der Klasse erstellt wird. Diese Methode wird auch “Konstruktor” genannt. Hier werden alle Parameter des Objekts initialisiert.
2. `step(self)`: `Step(...)` ist das Herzstück der Klasse. Hier wird jeder Schritt der Künstlichen Intelligenz definiert. Dazu wird zunächst eine Aktion definiert, welche die Maschine ausführen soll. Danach wird observiert, was sich in der Umgebung verändert hat und daraus ein Schluss gezogen, ob die Aktion zu einem guten, schlechten oder gar keinem Ergebnis führte; es wird also die Belohnung errechnet. Der Rückgabewert besteht dann aus den Observationen, der Belohnung, dem Status der Episode, welche die Künstliche Intelligenz zurzeit erlernt und einer beliebigen Zusatzinformation.
3. `render(self)`: `Render(...)` kann optional implementiert werden. Diese Funktion dient dazu dem Anwender den Lernfortschritt visuell darzustellen.
4. `reset()`: die vierte und letzte Funktion ist `Reset(...)`. Diese wird dazu verwendet um nach einer Episode die Umgebung und alle dazugehörigen Variablen auf einen gewünschten Stand zurückzusetzen. Als return Wert wird der aktuelle Stand der Künstlichen Intelligenz (die “Observation”) zurückgegeben.

Für die Beschreibung einzelner Begriffe empfiehlt es sich folgendes Kapitel zu lesen: “Künstliche Intelligenz allgemein” (4.3.6)

Die ScribbleFight-KI wurde wie folgt in Python implementiert:

Zunächst werden die Dependencies importiert.

Listing 34: ScribbleFight-KI OpenAI-Gym Environment

```

1      # Import dependencies
2      import gym
3      from gym.spaces import MultiDiscrete, Box
4      from KI_v01.env.game import Game

```

Die Klasse CustomEnv, welche von “gym.Env” erbt wird wie folgt implementiert:

```

5      class CustomEnv(gym.Env):
6
7          ...
8          This class is a custom open-ai gym environment
9          It has helper functions which are implemented below
10         ...
11

```

```

12         #metadata = {'render.modes' : ['human']}
13     def __init__(self):
14         # in the init function a new ScribbleFight instance is created
15         self.pygame = Game()
16
17         # the actionspace is defined by a MultiDiscrete set of random numbers
18         '''DISCRETE[0]: ANGLE+, ANGLE-, idle
19         DISCRETE[1]: "SPACE", "E", "Q", "R", "C", "F", "LEFTCLICK", idle
20         DISCRETE[2]: "A", "D", "idle'',
21         self.action_space = MultiDiscrete([3, 8, 3])
22
23         # the observationspace has the size of the visual copy Array (165x165),
24         # which is the vision of the AI
25         '''random output when observation_space is sampled:
26         [[0,0,0,...],
27          [0,1,0,...],
28          ....,
29          [0,1,3,...],
30          [0,1,5,...]]'',
31         self.observation_space = Box(0, 10, (165, 165), int)
32
33     def reset(self):
34         # the reset function resets the game
35         self.pygame.reset()
36         obs = self.pygame.observe()
37         return obs
38
39     def step(self, actions):
40         # in the step function a step is made
41         self.pygame.action(actions)
42         obs = self.pygame.observe()
43         # checks if observations are valid
44         # if true then there are no observations made
45         '''
46         comparison = obs == Box(0, 0, (165, 165), int).sample()
47         equal_arrays = comparison.all()
48         print(equal_arrays)
49         '''
50         reward = self.pygame.evaluate()
51         done = self.pygame.is_done()
52
53         # info = self.pygame.info() # not really needed
54         return obs, reward, done, {}
55
56     def render(self, mode="human", close=False):
57         # this function renders the game
58         self.pygame.view()

```

Eine weitere Bibliothek basierend auf Python, welche beim Erstellen der Reinforcement-Learning KI verwendet wurde, ist “Stable-baselines-3”. Diese bietet auf eine zuverlässige Art und Weise die Implementierung von mathematischen Reinforcement-Learning Modellen. Dabei werden viele Algorithmen, wie zum Beispiel “A2C”, “DQN”, oder “PPO”, zur Verfügung gestellt. Während dieser Arbeit wurden die Algorithmen “A2C” und “PPO” verglichen. In dem Fall der “Scribble-Fight” Künstlichen Intelligenz, welche ziemlich komplex ist, hat die Recherche ergeben, dass diese zwei Arten, Modelle zu trainieren, am besten geeignet sind.

Eingebunden wird die Bibliothek wie folgt:

Listing 35: Stable-Baselines3 Beispiel

```

1 import gym
2
3 from stable_baselines3 import PPO
4 from stable_baselines3.common.env_util import make_vec_env

```

OpenAI-Gym kann verwendet werden, um ein Spielumgebung zu erschaffen. In diesem Beispiel wird wieder (wie in 33) die “CartPole”-Welt verwendet. Damit die Effizienz des Lernens erhöht wird, kann durch das Erstellen der Umgebung mittels “vec\_env” eine Anzahl an Umgebungen gleichzeitig kreiert werden. Diese Anzahl wird als Parameter übergeben. Im folgenden Fall werden vier Umwelten gleichzeitig erstellt. All diese lernen auf dasselbe Modell. Dies wird auch “Competitive Self-Play” genannt. Danach wird dieses Environment an den PPO-Algorithmus übergeben. PPO steht für “Proximal Policy Optimization”. Danach lernt die Maschine für 25.000 Schritte. Die Anzahl der Schritte haben nichts mit der Episodendauer zu tun. Ein Schritt ist entweder eine Bewegung nach links oder rechts.

```

6
7 # Parallel environments
8 env = make_vec_env("CartPole-v1", n_envs=4)
9
10 model = PPO("MlpPolicy", env, verbose=1)
11 model.learn(total_timesteps=25000)

```

Nach dem Trainieren kann das Modell gespeichert und wiederverwendet werden. Dies passiert mittels “PPO.load(… )”. Somit wird aus dem Gelernten ein Modell erstellt, welches verwendet werden kann, um vorhersagen zu treffen, darüber, wie sich die Plattform bewegen muss, um die Stange aufrecht zu erhalten.

```

12
13 model.save("ppo_cartpole")
14
15 del model # remove to demonstrate saving and loading
16
17 model = PPO.load("ppo_cartpole")
18
19 obs = env.reset()
20 while True:
21     action, _states = model.predict(obs)
22     obs, rewards, dones, info = env.step(action)
23     env.render()

```

Auf die Frage, wie die “Reinforcement-Learning-Algorithmen” funktionieren wird in dieser Arbeit nicht eingegangen.

## 5.4.2 Die ScribbleFight-KI [H]

Im Laufe der Ausarbeitung der Diplomarbeit und der damit zusammenhängenden Forschung änderte sich oft die Vorstellung darüber, wie das Endprodukt (KI) auszusehen hat, beziehungsweise, wie dieses aussehen kann. Limitierungen, welche vor der Forschung noch nicht bekannt und bewusst waren, begrenzten die Lernfähigkeit der KI. Auf Probleme, Lösungen und Ergebnisse wird jedoch in diesem Kapitel noch näher eingegangen.

Das Ziel bestand darin eine Künstliche Intelligenz zu erschaffen, welche auf einem, für einen durchschnittlich guten, menschlichen Spieler, herausfordernden Niveau spielt. Dabei sollte diese:

- Alle möglichen Attacken erlernen.
- erlernen, in welchem Winkel ein Schuss abgefeuert werden muss, um einen Gegner zu treffen.
- lernen, wann eine Attacke, oder ein Schuss abgefeuert werden muss, um einen Gegner zu treffen.
- erlernen, wie ein Agent auf der Plattform bleibt, beziehungsweise was und wo diese ist.
- gegnerische Kugeln beziehungsweise Attacken erkennen und ausweichen.

Im Laufe der Ausarbeitung der Künstlichen Intelligenz wurde jedoch klar, dass dies Ziele sind, welche im Zuge einer Diplomarbeit, in diesem Ausmaß, nicht erreichbar waren. Die Künstliche Intelligenz lernte jedoch schnurstracks und am schnellsten Wege die Plattform zu verlassen.

Damit die Maschine lernen kann, stellt das Scribble-Fight-Frontend, welches mit Webtechnologien, wie HTML und JavaScript umgesetzt wurde, Informationen über den Spieler beziehungsweise Agenten zur Verfügung. Diese werden in einem “myPlayer”-Objekt gespeichert. Das Objekt sieht wie folgt aus:

```
> myPlayer
< Player {sprite: Sprite, id: 'Q8VvTOH4DopO_-ZiAABN', knockback: 1, death:
  1, kills: 0, ...} ⓘ
  damagedBy: null
  death: 1
  direction: "right"
  dmgDealt: 0
  id: "Q8VvTOH4DopO_-ZiAABN"
  item: [bomb: Item]
  kills: 0
  knockback: 1
  sprite: Sprite {position: Vector, previousPosition: Vector, newPosition:
  [[Prototype]]: Object}
```

Abbildung 45: “myPlayer”-Objekt

Für die Künstliche Intelligenz sind Daten, wie “death” (Tode), “dmgDealt” (verteilten Schaden), “kills” (Anzahl an Spieler, welche von diesem Agenten von der Plattform gestoßen wurden) und “knockback” (Rückstoß, welchen man erleidet) von Bedeutung.

Damit die Werte des Objekts ausgelesen werden können wurde in Python folgende Funktion implementiert:

Listing 36: Informationen über den Spieler

```

1 def getStats(driver):
2     dmgDealt, knockback, deaths, kills = 0, 0, 0, 0
3
4     dmgDealt, knockback, deaths, kills = driver.execute_script(
5         'return [myPlayer.dmgDealt, myPlayer.knockback, myPlayer.death,
6             myPlayer.kills]')
7
8     return dmgDealt, knockback, deaths, kills

```

Die Funktion extrahiert den für die Künstliche Intelligenz relevanten Inhalt und gibt diese als return Wert wieder zurück. Dadurch wird die Maschine bei jedem Schritt über den Status des Spielers informiert.

### Beschreibung der Funktionalität [H]

Ein weiterer, zentraler Punkt der Ausarbeitung ist, wie die “ScribbleFight”-KI zu sehen hat. Dies passiert mit einem zweidimensionalen Array, welcher “visCopy” im Frontend genannt wird. Die Einträge in diesem Feld sind die Zahlen von null bis zehn. Dieser wird erstellt, basierend auf den Inhalt des Spiels, welchen auch ein menschlicher Spieler sehen würde. Somit ist es für die KI nicht möglich, Daten zu manipulieren, auf welche ein Gegenspieler keinen Zugriff hätte. Dabei wird das Sichtfeld wie folgt codiert:

- 0: Hintergrund
- 1: Plattform
- 2: eigener Agent
- 3: Gegner
- 4: eigenes Projektil
- 5: gegnerisches Projektil
- 6: Bombe
- 7: Schwarzes Loch
- 8: Verkleinerung
- 9: Miene
- 10: Piano

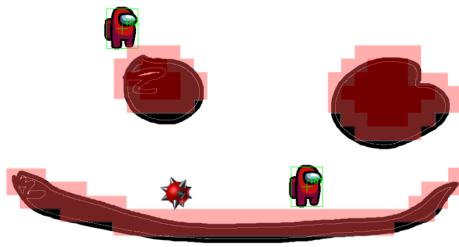


Abbildung 46: Spieler Sicht

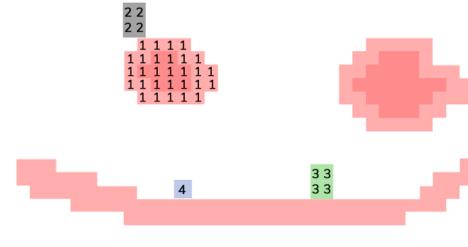


Abbildung 47: KI Sicht

Der ‘‘Pixelclump’’ ist ein zweidimensionaler Array von Pixelwerten (schwarz oder transparent, weiß) und wird von der Maperkennung (5.3) als Output zurückgeliefert. Dieses zweidimensionale Feld dient als Basis für den Untergrund, auf welchem sich die Spieler bewegen können. Um das Spiel in eine, für die Künstliche Intelligenz sichtbare Welt umzuwandeln, wird zunächst der ‘‘Pixelclump’’, um das Dreifache vergrößert. Gleichzeitig werden die Pixelwerte durch die oben angeführten Zahlen (von null bis zehn; Bedeutung: 5.4.2) ausgetauscht. Durch die dreifache Skalierung der Spielumgebung von 55x55 auf 165x165 Werten, ist es möglich, dass die Künstliche Intelligenz dreimal genauer sieht, als ein Bildpunkt der Map aufgetragen wird. Somit kann genauer agiert werden und Spieler und Attacken besser erkannt werden (in der Theorie).

Die Aufskalierung und Umwandlung des Map-Arrays in den ‘‘Visual’’-Array passiert in der Funktion ‘‘getVisualMap(...)'':

Listing 37: Umwandlung des Pixel-Clumps in eine für die KI sichtbare Darstellung

```

1  /**
2   * Converts standard map, which is 55x55
3   * into visual map which is 165x165 for better accuracy
4   *
5   * @param {given array which represents map} paramarr
6   * @returns new visual array
7  */
8  function getVisualMap(paramarr) {
9    let visual = []
10   let factor = 3
11   for (let i = 0; i < paramarr.length * factor; i++) {
12     visual.push([])
13     for (let j = 0; j < paramarr.length * factor; j++) {
14       if (paramarr[int(i / factor)][int(j / factor)][3] > 0) {
15         visual[i].push(1)
16       } else {
17         visual[i].push(0)
18       }
19     }
20   }
21   return visual
22 }
```

Mit jeder Bewegung und Aktualisierung von Status wird der ‘‘visual’’-Array in einen ‘‘visCopy’’-Array eingetragen. Dabei werden alle Entitäten neu hinzugefügt. Das Feld wird jedoch durch die Kopie nur an den Positionen verändert, wo auch eine Änderung passiert

ist. Dadurch wird dieses nicht immer neu erstellt, und die Effizienz wird gesteigert. Außerdem, dadurch, dass Javascript zur Kalkulation komplexer Berechnungen mehrere Threads zur Verfügung hat, kann das Verarbeiten dieses Arrays in einer Zeit passieren, welche die Bilder pro Sekunde des Spiels nicht verringert und somit für den Spieler nicht bemerkbar. Lässt man sich nun die bearbeitete Kopie des “visual”-Arrays ausgeben, so sieht man die Sicht der “ScribbleFight” Künstlichen Intelligenz. In der folgenden Abbildung wurden die Ziffern von null bis zehn in Farben umgewandelt. Das Bild 48 ist die Sicht des Spielers und das Bild 49 ist die Sicht der KI.

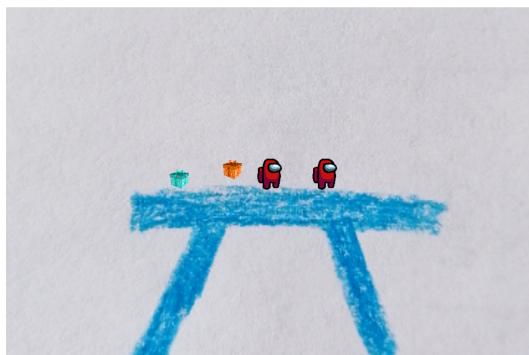


Abbildung 48: Spieler Sicht

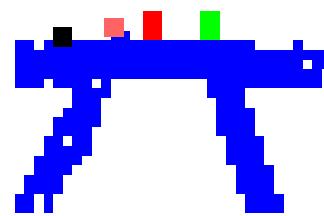


Abbildung 49: KI Sicht

### 5.4.3 ScribbleFight Reinforcement Learning Implementierung [H]

Wie bereits in dem Kapitel “Reinforcement Learning” (9) erklärt, basiert diese Art des Lernens auf unpräparierte Daten. Das heißt, dass kein Trainingsset an Daten existiert, welches zum Lernen und Vergleichen verwendet wird, sondern ein Agent in eine ihm unbenannte Umwelt gesetzt wird und rein über ausprobieren lernt. Das einzige Ziel besteht darin, dass eine maximale Höhe an Belohnung erreicht werden soll.

Im folgenden wird der Programmcode (38), welcher diese Forderungen umsetzt, beschrieben.

Bevor die KI implementiert werden kann, muss eine Referenz auf das Browserspiel erzeugt werden. Dies passiert mit der Bibliothek namens “Selenium”. Selenium ermöglicht es mit einem Python-Code ein Browser-Fenster zu öffnen, in welchem diverse Keyboard-Events und vieles mehr ausgeführt werden können. Dafür wird zunächst die Optionen, welche beim Öffnen ein Browserfenster berücksichtigt werden erstellt. Die Optionen

besagen zum Beispiel, dass sich das Fenster an die Bildschirmgröße mit “start-maximized” anpasst. Die “Url” auf welche navigiert werden soll ist “`http://localhost:3000`”, da hier das Spiel auf einem Node-Server, als lokale Instanz auf das Spiel, zur Verfügung gestellt wird.

Listing 38: Implementation der Logik der Reinforcement Learning KI

```

1  '''DEPENDENCIES'''
2  # selenium webdriver
3  from webdriver_manager.chrome import ChromeDriverManager
4  from selenium import webdriver
5
6  # options
7  from KI_v01.env.options.actions import *
8  from KI_v01.env.options.observations import *
9
10 # openai gym
11 from gym.spaces import Box
12
13
14 '''VARIABLES'''
15 FPS = 10
16
17 # selenium webdriver options
18 options = webdriver.ChromeOptions()
19 options.add_argument("start-maximized")
20 options.add_argument("disable-infobars")
21 port = 3000
22 url = 'http://localhost:%s' % (port)

```

Im folgenden Programmcode wird eine Klasse namens “ScribbleFight” erstellt.

In der `__init__` Funktion, also dem Konstruktor der Klasse “ScribbleFight” werden alle Variablen und Konstanten initialisiert und somit auf einen Startwert gesetzt. Dabei werden alle, für das Spiel relevanten Parameter auf einen “Null-Wert” gesetzt, da noch keine Daten gewonnen wurden. Auch Wahrheitsoperatoren, wie der “readystate”, also ob das Browserfenster gestartet wurde oder nicht werden hier gesetzt. Nach dem Kreieren wird das Fenster und somit das Spiel gestartet.

```

25  class ScribbleFight:
26
27      '''SCRIBBLE FIGHT GAME'''
28
29      def __init__(self):
30          self.driver = None
31          self.playerId = 0
32          self.obs = Box(0, 0, (165, 165), int).sample()
33          self.dmgDealt = 0
34          self.knockback = 0
35          self.kills = 0
36          self.deaths = 0
37          self.just_died = False
38          self.readystate = False
39          self.startBrowser()

```

In der Funktion “startBrowser” wird ein neuer webdriver erstellt, über welchem später die Künstliche Intelligenz das Spiel steuert. In einer Schleife wird überprüft, ob das Fenster bereits geöffnet wurde.

```

41      def startBrowser(self):
42          self.driver = webdriver.Chrome(options=options,
43                                         executable_path=ChromeDriverManager().install())
44          self.driver.get(url)

```

```

45
46     while not self.readyState:
47         self.readyState = self.isReady()
48
49     def isReady(self):
50         try:
51             self.playerId = self.driver.execute_script('return myPlayer.id;')
52             return True
53         except:
54             return False

```

Die folgenden drei Funktionen sind “update”, “move” und “action”.

Update verwendet die zuvor erklärte “getStats” Methode (Code-Block 36) um den aktuellen Stand des Spielers wiederherzustellen, beziehungsweise die Informationen und den Status des Spielers zu aktualisieren.

Die Funktion “action” führt, basierend auf einem Übergabewert, eine Aktion aus. Diese Aktion kann eine Zahl zwischen null und sieben sein. Null stellt dabei zum Beispiel die Handlung “jump”, also Springen, dar. Diese wird als “Aktionsstring” zurückübergeben.

In der Methode “move” wird ebenfalls ein Aktionsstring generiert. Jedoch wird hier eine links oder rechts Bewegung angestrebt, basierend auf Ziffern zwischen null und zwei. Die Aktion null steht für einen Schritt nach links, eins steht für einen Schritt nach rechts und zwei für “keine Bewegung”.

```

56     def update(self):
57         # get stats
58         self.dmgDealt, self.knockback, self.deaths, self.kills =
59             getStats(self.driver)
60
61     def action(self, action, angle):
62         # down ('s') action not implemented
63         actionString = ''
64         if action == 0:
65             actionString = 'jump()'
66         ...
67         if action == 7:
68             # idle state
69             pass
70         return actionString
71
72     def move(self, action):
73         actionString = ''
74         if action == 0:
75             actionString = 'moveLeft(); mirrorSpriteLeft();'
76         if action == 2:
77             # idle state
78             pass
79         return actionString

```

“observe” ist dafür da, Observationen aus dem Spiel zu extrahieren.

In der Methode wird die Funktion “readFromClient” aufgerufen, welche den “visCopy”-Array aus dem Frontend zurückgibt. Wie dieser Array funktioniert, was dieser ist und wie er erzeugt wird, wird im Kapitel “Beschreibung der Funktionalität” 5.3.1 erklärt.

Weiters wird überprüft, ob das Feld existiert. Sollte “readFromClient” “false” als Rückgabewert liefern, so wird ein neuer Array in derselben Dimension (zweidimensional) und Größenordnung (165x165) erstellt; jedoch mit lauter Nullen, welche für ein leeres Feld stehen. Liefert “readFromClient” “visCopy” als Rückgabewert zurück, so wird die Variable “obs” (Observation) mit diesem belegt.

```
78     def observe(self):
79         visCopy = readFromClient(self.driver, 'return visCopy;')
80         if visCopy:
81             self.obs = visCopy
82         if not visCopy:
83             self.obs = Box(0, 0, (165, 165), int).sample()
84     ...
```

Wie auch in der Klasse “ScribbleFight” (38) beinhaltet “Game” zunächst eine “`__init__`”-Funktion, welche als Konstruktor dient. Dieser setzt Startwerte der Parameter der Funktion. Beispielsweise wird als erstes eine Instanz auf ein Objekt der Klasse “ScribbleFight” erstellt. Variablen, welche “`previous_`” als Präfix haben, dienen dazu den Unterschied zu einem vorherigen Schritt beziehungsweise Status herauszufinden. Die Implementation wird genauer in folgenden Codeblöcken beschrieben.

```
85 class Game:
86     '''AI INTERFACE TO SCRIBBLEFIGHT GAME INSTANCE'''
87
88     def __init__(self):
89         self.scribble_fight = ScribbleFight()
90         self.min_game_length = 30 * FPS # 1 min
91         self.nothingChanged = 0 # player didn't accomplish anything in this
92             timespan
93         self.just_won = False # the winning condition has yet to be implemented
94         self.previous_damage_dealt = 0
95         self.previous_knockback = 0
96         self.previous_kills = 0
97         self.previous_deaths = 0
98         self.angle = 0 # angle in which the ai can shoot a bullet
```

In der “action”-Funktion wird zunächst überprüft, ob das Spiel läuft. Danach wird ein “String” (Zeichenkette) namens “actionString” angelegt. Dieser wird leer initialisiert. Darunter wird der “actionString” basierend auf einem eindimensionalen Array, welcher drei Einträge hat, befüllt. Der erste Eintrag beinhaltet die Information, über die Richtung, in welche das Programm den Agenten beziehungsweise Spieler steuern möchte. Der nächste Eintrag bestimmt, welche Attacke die Künstliche Intelligenz ausführen möchte. An der letzten Position des Arrays wird bestimmt, wie die Künstliche Intelligenz den Abschusswinkel, in welchem sie das Projektil abfeuern möchte, anpassen soll. Ist dieser Wert des Arrays null, so wird der Winkel um fünf Grad gegen den Uhrzeigersinn angepasst. Umgekehrt passiert dies bei dem Wert eins. Zwei bewirkt, dass der Winkel gleich bleibt. Nachdem in der Funktion der “actionString” erstellt wird, wird dieser an die Methode “execAction” der “ScribbleFight”-Instanz weitergegeben. “execAction”

bewirkt die Ausführung der angegebenen Handlungen. Im Anschluss wird der Status des “ScribbleFight” Objekts und dessen Parameter aktualisiert.

```

101     def action(self, actions):
102         # if current browser window has wrong url
103         # then dont execute actions
104         if (self.scribble_fight.isPlaying() is False):
105             return
106
107         # take set of actions
108         '''DISCRETE[0]: "A", "D", idle
109         DISCRETE[1]: "SPACE", "E", "Q", "R", "C", "F", "LEFTCLICK", idle
110         DISCRETE[2]: ANGLE+, ANGLE-, idle'''
111         actionString = ''
112
113         actionString += self.scribble_fight.move(actions[0])
114         actionString += self.scribble_fight.action(actions[1], self.angle)
115         if actions[2] == 0:
116             self.angle += 5
117             if self.angle > 360:
118                 self.angle -= 360
119         if actions[2] == 1:
120             self.angle -= 5
121             if self.angle < 0:
122                 self.angle += 360
123         if actions[2] == 2:
124             # idle state
125             pass
126
127         if actionString:
128             self.scribble_fight.execAction(actionString)
129
130         # update and save in-game stats
131         self.scribble_fight.update()

```

Nachdem die vordefinierte Handlung der KI ausgeführt wurde, ist beim verstärkten Lernen der nächste Schritt, die Umwelt zu observieren. Der wohl wichtigste Schritt passiert danach. In der Funktion “evaluate” wird der Reward beziehungsweise die Belohnung berechnet. Dieser wurde wie folgt vergeben:

- hat der Agent der Künstlichen Intelligenz Schaden ausgeteilt (einem gegnerischem Spieler Schaden hinzugefügt), so wird der Belohnung eins addiert.
- erleidet der Agent Schaden, so wird der Belohnung eins subtrahiert.
- hat der Agent einen Spieler von der Plattform gestoßen beziehungsweise diesem so viel Schaden zugefügt, dass er stirbt, so wird als Belohnung eintausend Punkte vergeben.
- stirbt der Agent selbst, so wird eine negative Belohnung von 990 vergeben. Der Unterschied von zehn Punkten in der Entlohnung zwischen “Gegner töten” und “selbst sterben”, wurde in der Hoffnung festgelegt, dass sich der Agent für einen “Kill” opfert.

Nach der Evaluation der Belohnung wird überprüft, ob sich im Spiel etwas geändert hat. Wenn der Agent keinen “Reward” (null) erzielt hat, was bedeutet, dass er weder Schaden zugefügt, noch erlitten hat, keinen von der Plattform gestoßen hat und nicht selbst gestorben ist, so wird der Wert der Variable “nothingChanged” um eins erhöht.

Erreicht diese Variable eine gewisse Zahl erreicht hat, welche zuvor als Episodendauer definiert wurde (“min\_game\_length”), so wird in der Funktion der Wahrheitswert “true” (wahr) zurückgegeben, was so viel bedeutet, wie dass eine Episode beendet wurde. Die Variable “min\_game\_length” wurde als  $30 * \text{der Bilder pro Sekunde}$  definiert. Dadurch wird eine Episode, in welcher sich für 30 Sekunden nichts ändert, abgebrochen. Eine Episode ist auch vorbei, wenn der Agent gewonnen hat, oder gestorben ist.

```

138     def observe(self):
139         # get computer vision
140         self.scribble_fight.observe()
141         return self.scribble_fight.obs
142         # return None
143
144     def evaluate(self):
145         reward = 0
146         dmgDealt = self.scribble_fight.dmgDealt
147         knockback = self.scribble_fight.knockback
148         kills = self.scribble_fight.kills
149         deaths = self.scribble_fight.deaths
150
151         # evaluate reward
152         # if reward is calculated then update previous_x variables
153         # since they will be used for future comparison and reward calculation
154         if dmgDealt > self.previous_damage_dealt:
155             reward += 1
156         if knockback > self.previous_knockback:
157             reward -= 1
158         if kills > self.previous_kills:
159             reward += 1000
160         if deaths > self.previous_deaths:
161             # the reason why less reward is given when dying is because
162             # I want the ai to be a killer machine
163             reward -= 990
164             self.scribble_fight.just_died = True
165
166         self.previous_damage_dealt = dmgDealt
167         self.previous_knockback = knockback
168         self.previous_kills = kills
169         self.previous_deaths = deaths
170
171         if reward == 0:
172             self.nothingChanged += 1
173         else:
174             self.nothingChanged = 0
175
176     return reward
177
178     def is_done(self):
179         # returns if game won or nothing changed or agent died
180         return self.scribble_fight.just_died or self.just_won or
181             self.nothingChanged == self.min_game_length

```

In der Funktion “reset” werden die Variablen der Klasse “Game” zurückgesetzt.

“info” alle gesammelten Informationen über den Spieler zurück. Diese Funktion wird allerdings nicht verwendet und wurde nur zur Vollständigkeit implementiert.

“view” ist eine Methode, welche OpenAI-Gym abverlangt. In dieser das Spiel für den Programmierer der Künstlichen Intelligenz visuell angezeigt. Da jedoch “ScribbleFight” ein Frontend besitzt ist die Implementierung dieser Funktion sinnbefreit.

```

182     def reset(self):
183         # reset position and observation
184         if not self.scribble_fight.just_died:
185             self.scribble_fight.resetPlayer()
186         # reset in-game variables
187         self.scribble_fight.update()
188         # reset variables

```

```

189         # they should always be 0 because a reset only accures after death or as
190         # initialisation
191         self.previous_damage_dealt = self.scribble_fight.dmgDealt # should always
192         # be 0
193         self.previous_knockback = self.scribble_fight.knockback # should always
194         # be 1
195         self.previous_kills = self.scribble_fight.kills # should always be 0
196         self.previous_deaths = self.scribble_fight.deaths
197         self.nothingChanged = 0
198         self.angle = 0
199
200     def info(self):
201         # return all infos about player
202         # not really needed tho
203         return self.scribble_fight.dmgDealt, self.scribble_fight.knockback,
204             self.scribble_fight.kills, self.scribble_fight.deaths
205
206     def view(self):
207         pass

```

#### 5.4.4 Evaluation der ScribbleFight KI [H]

Im Laufe der Arbeit war der Teil “ScribbleFight KI” stark mit Forschung verbunden. Dazu wurde das Programm, mit welchem die Maschine hätte lernen sollen, 25 mal ausgeführt. Dabei wurden natürlich auch 25 Modelle erlernt. Allerdings wurden nur in 8 Fällen länger als ein Tag trainiert, da zu Testzwecken das Training abgebrochen wurde. Im folgenden werden die zwei Reinforcement Learning Algorithmen “A2C” und “PPO” verglichen und die besten Ergebnisse der Modelle präsentiert.

#### A2C [H]

A2C ist einer der zwei Algorithmen, mit welchem die ScribbleFight Künstliche Intelligenz trainiert wurde. A2C steht für “Advantage Actor Critic”. Dabei ist dies ein “Reinforcement Learning” Algorithmus, welcher zwei Arten von verstärktem Lernen miteinander kombiniert. Einerseits das “Policy-Based Reinforcement Learning”, in welchem der Agent direkt ein Verhalten erlernt, was bedeutet dass dieser lernt eine Eingabe direkt auf die bestmögliche Ausgabe umzuwandeln. Andererseits das “Value-Based Reinforcement Learning”. Hier erlernt das Modell Handlungen so auszuwählen, dass diese der Vorhersage der Eingabe gleicht.

Die mit OpenAI-Gym und Stable Baselines3 erlernten Modelle können via einem “TensorBoard” analysiert werden.

Die Graphen, welche aus den Modellen entsprungen sind zeigen, dass der A2C Algorithmus für die Künstliche Intelligenz nicht geeignet ist. Beispielsweise ist in folgender Grafik erkennbar, dass die Entropie, also Zufälligkeit, mit welcher die Künstliche Intelligenz Entscheidungen trifft zwar anfänglich stark abfällt, jedoch später wieder steigt.

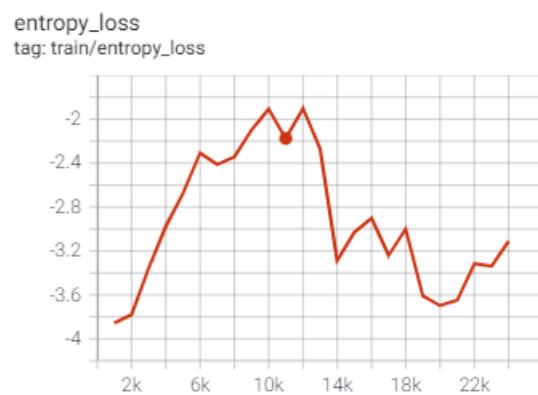


Abbildung 50: Entropie Verlust

## PPO [H]

# 6 Evaluation des Projektverlaufs

## 6.1 Meilensteine

Ist es uns gut dabei gegangen diese einzuhalten?

## 6.2 Gelerntes

Haben wir bei der DA was gelernt? nein

## 6.3 Was würden wir anders machen?

JOA Was würden wir anders machen?

Aufzählungen:

- Itemize Level 1
  - Itemize Level 2
    - Itemize Level 3 (vermeiden)
- 1. Enumerate Level 1
  - a. Enumerate Level 2
    - i. Enumerate Level 3 (vermeiden)

**Desc** Level 1

**Desc** Level 2 (vermeiden)

**Desc** Level 3 (vermeiden)



# **Literaturverzeichnis**

[1] „p5.js erklärt.” Online verfügbar: <https://p5js.org/>

# Abbildungsverzeichnis

1	Brawlhalla Spielumgebung . . . . .	5
2	Aufbau index.html . . . . .	7
3	Simples sketch.js Beispiel . . . . .	8
4	Package.json des Scribble-Fight Backends . . . . .	10
5	Simpler Web Server mit Express . . . . .	10
6	Socket.IO Client Beispiel . . . . .	11
7	Socket.IO Server Beispiel . . . . .	12
8	Veranschaulichung Docker Architektur . . . . .	14
9	Veranschaulichung bestärkendes Lernen . . . . .	18
10	Neuron . . . . .	19
11	Eingabe → Berechnung → Ausgabe . . . . .	19
12	Fehlerkurve . . . . .	21
13	Hexapawn Spielumgebung . . . . .	22
14	Hexapawn mögliche Züge . . . . .	23
15	Einfacher Sprite . . . . .	25
16	Player-Klasse . . . . .	26
17	Item-Klasse . . . . .	27
18	Originale Zeichnung . . . . .	37
19	Bilddaten-Array bildlich dargestellt . . . . .	37
20	Spielumgebung . . . . .	38
21	Pixel-Unterschied . . . . .	45
22	Scribble-Fight Dockerfile . . . . .	51
23	Scribble-Fight Deployment.yaml . . . . .	53
24	Scribble-Fight Service.yaml . . . . .	54
25	Scribble-Fight Ingress.yaml . . . . .	54
26	Screenshots der Maperkennungsanwendung . . . . .	56
27	Documenten Scanner . . . . .	59
28	Input . . . . .	64
29	Output . . . . .	64
30	Input . . . . .	66
31	Output . . . . .	66
32	Input . . . . .	67
33	Helligkeitsverteilung . . . . .	67
34	Input . . . . .	68
35	Output . . . . .	68
36	Input . . . . .	69
37	Output . . . . .	69
38	Input . . . . .	71
39	Output . . . . .	71
40	Der Ramer–Douglas–Peucker Algorithmus . . . . .	72
41	Input . . . . .	73
42	Erster Output des Programmcodes . . . . .	73

---

43	Input . . . . .	76
44	perspektiventransformiert . . . . .	76
45	“myPlayer”-Objekt . . . . .	85
46	Spieler Sicht . . . . .	87
47	KI Sicht . . . . .	87
48	Spieler Sicht . . . . .	88
49	KI Sicht . . . . .	88
50	Entropie Verlust . . . . .	95

# **Tabellenverzeichnis**

# Quellcodeverzeichnis

1	OpenCV Demo . . . . .	16
2	PIL Demo . . . . .	16
3	Keyboard-Access . . . . .	28
4	Bomb Item Physics . . . . .	28
5	Attraction . . . . .	29
6	Black Hole Item Physics . . . . .	29
7	Piano-Item Physics . . . . .	30
8	Mine-Item Physics . . . . .	30
9	Size-Reduction . . . . .	31
10	Default-Attacke . . . . .	31
11	Jumping . . . . .	32
12	Links/Rechts-Movement . . . . .	32
13	Bewegung auf der Spielumgebung . . . . .	33
14	Knockback-Bewegung . . . . .	33
15	Sprite Richtungswechsel . . . . .	34
16	Überprüfung nach Toden . . . . .	35
17	Fatal Hit . . . . .	35
18	Vereinfachte Darstellung eines Bilddaten-Arrays . . . . .	36
19	Erstellen eines Items . . . . .	39
20	Item-Physik . . . . .	39
21	Bestimmung gültiger X-Koordinaten . . . . .	40
22	Alle unnötigen Bilddaten entfernen . . . . .	58
23	Erhalten von Konturen . . . . .	59
24	approxPolyDP . . . . .	60
25	Bild in Vogelperspektive umwandeln . . . . .	60
26	Graustufenkonvertierung . . . . .	63
27	Gaussian Blur . . . . .	66
28	Adaptive Gaussian Thresholding . . . . .	68
29	Adaptive Gaussian Thresholding . . . . .	70
30	ApproxPolyDP Beispiel . . . . .	73
31	Perspektiventransformation Beispiel . . . . .	75
32	Bild in Spielbare Map umwandeln . . . . .	77
33	Simples OpenAI-Gym Programm . . . . .	81
34	ScribbleFight-KI OpenAI-Gym Environment . . . . .	82
35	Stable-Baselines3 Beispiel . . . . .	83
36	Informationen über den Spieler . . . . .	86
37	Umwandlung des Pixel-Clumps in eine für die KI sichtbare Darstellung . . . . .	87
38	Implementation der Logik der Reinforcement Learning KI . . . . .	89

# **Anhang**