Matura Arbeit 2022

Entwicklung und Integrierung von Blockchain Smart Contracts in ein Computerspiel

Filip Cybulski

Referenten: Elisabeth Germann und Werner Winkelmann
KANTONSSCHULE KÜSNACHT

Contents

1 Vorwort

Vor circa zwei Jahren wurde mir ein Video über Bitcoin auf YouTube vorgeschlagen. Ich hatte immer wieder etwas von dieser digitalen Währung gehört, aber was es genau war, wusste ich nicht. Mein Interesse war geweckt und so klickte ich auf das Video. Ich erfuhr zum ersten Mal, dass es so etwas wie Kryptowährungen gab. Die Preise von Bitcoin waren rasant am Steigen, und da ich auch davon profitieren wollte, fragte ich meinen Vater, ob er mir einen Account erstellen würde. Nach einigem Anfängerglück wurden meine Hoffnungen von der Realität der Finanzmärkte getrübt. Das Interesse für Kryptowährungen blieb aber. Ein Jahr später, hatte ich gerade das Wahlfach Programmieren abgeschlossen. Ich hatte im Rahmen des Wahlfaches versucht, ein Krypto-Trading-Bot zu programmieren. Einige Wochen nach den Sportferien, bin ich dann mit Tomás im Gespräch auf die Idee gestossen gemeinsam ein Computerspiel zu programmieren. Da ich das Potenzial von Kryptowährungen und der dahinter liegenden Technologie besonders im Gaming Bereich sah, bildete sich langsam die Idee, mithilfe von Blockchain und Smart Contracts Transaktionen für unser Computerspiel zu integrieren, heraus. So könnten die Spielerinnen unseres Games um Kryptowährungen spielen, und so Geld gewinnen, beziehungsweise verlieren. Als dann die Zeit kam, sich für die Matura Arbeit ein Thema auszuwählen, beschlossen Tomás und ich die Chance zu nutzen und unser Computerspiel gemeinsam als Matura Arbeit zu entwickeln.

Filip Cybulski, Dezember 2022

2 Einleitung:

Die letzten Monate habe ich damit verbracht, automatisierte Transaktionen für ein Computerspiel zu entwickeln, um so selbst Amateur-Spielern zu ermöglichen, Kleingeld zu verdienen, oder zu verlieren. Bis dahin kannte ich kein Spiel, welches in Tournament-Runden, den Spielerinnen erlaubte, ihr eigenes Geld einzuzahlen und gegen andere Spieler anzutreten. Lotto existierte, online Glücksspiele auch, aber eins fehlte, ein Spiel, welches das Konzept von Lotto nahm, und den Glücksaspekt mit den Fähigkeiten der Spielerinnen ersetzte. Gemeinsam mit meinem Partner Tomás Brown Martinez, beschlossen wir, diese Herausforderung auf uns zu nehmen. Ich habe mich entschieden, die Matura Arbeit mit Tomás zu machen, da wir so gemeinsam ein grösseres Projekt gestalten konnten. Ausserdem entspricht die Zusammenarbeit in einem Team, viel eher der Realität im Berufsleben. Tomás hat das Programmieren des Computerspiels übernommen, während ich mich der Automatisierung von Transaktionen gewidmet habe.

Der vorliegende Text dient als Begleittext zur Maturarbeit «Die Entwicklung und Integrierung von Smart Contracts in ein Computerspiel». Der Text ist für Personen gedacht, welche sich genauer für den Prozess hinter der Entwicklung der Smart Contracts für unser Computerspiel interessieren. Ich schreibe dabei in einer Weise, die ein gewisses Vorwissen im Thema Programmieren sowie Blockchain voraussetzt. Um den Inhalt dieses Begleittextes auf das Interessanteste und Nötigste zu reduzieren, erkläre ich das Thema nicht von Grund aus aufbauend für einen kompletten Laien. Sollte man trotzdem als solcher Interesse daran haben, das Geschriebene im Vollen zu verstehen, verweise ich auf Quellen, die dazu dienen die Technologie sowie Terminologie zu erklären. Als Ziel setzte ich mir das Programmieren von Smart Contracts, welche automatisierte Kryptowährung-Transaktionen auf der Blockchain ermöglichen, sowie deren Integration in das von Tomás entwickelte Computerspiel. So würde ich die Technologie der Blockchain in der Praxis kennenlernen.

Zu Beginn des Hauptteils definiere ich einige Begriffe, welche den Leserinnen dabei helfen sollten, das Geschriebene zu verstehen. Danach zeige ich einige Vorteile der Blockchain gegenüber üblicheren Modellen für die Automatisation von Transaktionen. Ich offenbare dann das Endziel meiner Arbeit und stelle es visuell in einem Schema dar, um dem Leser eine genauere Vorstellung des verfolgten Ziels zu geben. Den Grossteil des Begleittextes formt der chronologische Arbeitsprozess. In diesem wird beschrieben, wie bei der Arbeit vorgegangen wurde. Dabei beschränke ich mich auf das Interessanteste und gehe jeweils bei wichtigeren Problemen ins genauere Detail. Später zeige ich die genaue Struktur des endgültigen Smart Contracts anhand von Ausschnitten aus dem Code. Einige weiteren Aspekte, auf welche im Hauptteil Fokus gelegt wird, sind der Jugendschutz, die Suchtprävention, sowie die Geldwäscherei. Dieser Abschnitt des Begleittextes überschneidet sich mit dem von Tomás, da für uns beide diese Themen von gleicher Wichtigkeit sind. Im Schlussteil wird retrospektive Selbstkritik betrieben, und es wird anhand von Beispielen aus der Arbeit gezeigt, wo Verbesserungspotenzial liegt, wo aber auch gut gearbeitet wurde. Ich gebe hier auch Tipps für angehende Maturanden und Maturandinnen, welche ihre Arbeit im Fach Informatik machen wollen.

Im Verlauf des Textes wird die Geschlechtsform jeweils zwischen männlich und weiblich abgetauscht.

3 Hauptteil:

3.1 Hilfreiche Begriffe zur Verstehung des Textes

- **Blockchain**[1]: Dezentralisierte Datenbank, welche unveränderbar ist
- **Smart Contracts**[2]: Automatisierte Kontrakte welche auf der Blockchain festgeschrieben sind
- **ERC-20 Token**[3]: Standardisierte Version für die Entwicklung von Tokens im Ethereum Ökosystem
- **Wallet**: Aus dem Englischen, für «Portemonnaie»; wird benutzt, um darin Kryptowährungen zu halten
- **BSC**: Abkürzung für Binance Smart Chain, eine Blockchain der Kryptowährungen-Börse Binance
- **Solidity**: Programmiersprache, welche zur Entwicklung von Smart Contracts benutzt wird
- **Testnetz**[4]: Ein Testnetz ist ein Blockchain Netzwerk welches ausschliesslich zum Testen benutzt wird, dies ist nötig, da es auf der Mainnet zu teuer wäre für Entwickler
- **BUSD**: Dies ist ein ERC-20 Token
- **ENTRToken, IGToken**: Dies sind ERC-20 Tokens, welche ich erstellt habe, sie erfüllen jeweils eine Funktion in meinem Smart Contract
- **Validator**: Jemand der für die Verifizierung von Transaktionen auf der Blockchain verantwortlich ist und für diese Arbeit mit Kryptowährungen entlohnt wird, welche die Benutzerinnen in Form von Transaktionskosten bezahlen
- **Ethereum Virtual Machine**: Die Ethereum Virtual Machine (EVM) ist eine Laufzeitumgebung für die Ausführung von Smart Contracts auf der Ethereum-Blockchain. Diese Software erlaubt der Blockchain dezentralisiert und automatisiert zu funktionieren.
- **Stable Coin**: Eine Kryptowährung welche den Wert einer Fiat Geld Währung wie zum Beispiel des US-Dollars widerspiegelt

3.2 Wieso Blockchain Smart Contracts die richtige Technologie zur Realisierung meiner Idee ist

Die Idee war es, ein Computerspiel zu programmieren, welches Spielerinnen erlauben würde, um Geld zu spielen. So könnten selbst leidenschaftliche Gamer in kleinen Spielrunden Geld verdienen, oder natürlich verlieren. Nun war es die Frage wie ich die Funktionalität, um Geld zu spielen, einbauen könnte. Ich wusste, dass es eine Technologie wie Smart Contracts gab, da ich mich ein wenig mit Kryptowährungen in der Vergangenheit beschäftigt hatte. Smart Contracts ermöglichen Transaktionen zu automatisieren, und das war das, was ich brauchte. Man könnte die Automatisation von Transaktionen auch mit regulärem Fiat Geld und anderer Technologie umsetzen, jedoch haben Smart Contracts verschiedenste Vorteile gegenüber Geldüberweisungen über Bankkonten, da die Technologie der Smart Contracts genau für Automatisation bestimmt ist.

[1] (Blockchain facts, "https://www.investopedia.com/terms/b/blockchain.asp")
[2] (What Are Smart Contracts on the Blockchain and How They Work?, "https://www.investopedia.com/terms/s/smart-contracts.asp")
[3] (What Are ERC-20 Tokens on the Ethereum Network?, "https://www.investopedia.com/news/what-erc20-and-what-does-it-mean-ethereum/")
[4] (Was ist Mainnet und Testnet?, "https://portalcripto.com.br/de/Was-ist-Mainnet--und-Testnet-Kryptow%C3%A4hrungsentwicklung%3F/")

- Ein erster Vorteil ist, dass keine Infrastruktur für Accounts entwickelt werden muss. Ein Spieler, der unser Game spielen möchte, braucht keinen Account zu erstellen, sondern kann sich mit seinem Wallet anonym verbinden.
- Zweitens muss ich keine Software von Drittanbietern nutzen, um die Überweisung per Karte möglich zu machen. Ich kann die dezentralisierte Technologie der Blockchain verwenden. Das spart mir die Kosten, welche ich für diese Software bezahlen müsste.
- Ein drittes Argument für die Auswahl von Blockchain für meinen Zweck, ist die einfachere Automatisation. Ich brauche mich auf keine Server zu verlassen, denn die Blockchain ist so konzipiert worden, dass sie ständig von so genannten Validatoren betrieben wird. Diese stellen ihre Computer zur Verfügung, um das Netzwerk zu unterhalten, und erhalten eine Entlohnung, welche die Benutzer in Form von Transaktionskosten bezahlen.

3.3 Das Endziel

Das Ziel meiner praktischen Arbeit war es, Smart Contracts zu entwickeln, mit deren Hilfe Transaktionen in einem Computerspiel automatisiert werden können. Dabei müssen meine Smart Contracts auf das Computerspiel abgestimmt sein. In diesem Abschnitt erkläre ich, welche Funktion meine Smart Contracts im Spiel übernehmen, damit das Lesen des darauffolgenden Arbeitsprozesses klarer wird. Die Funktion der Smart Contracts im Spiel wird im darunterliegenden Schema visuell dargestellt.

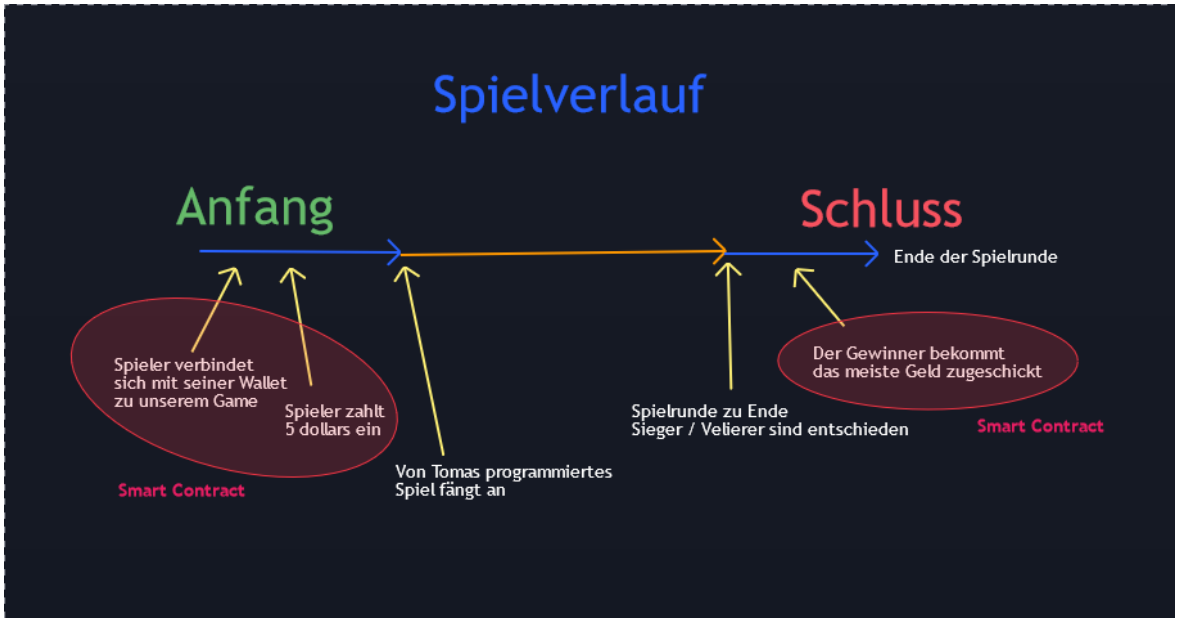

Abbildung 1: Schema der Funktionalität der Smart Contracts im Computerspiel

Die zwei roten Ovale stellen die Smart Contracts dar.

- Am Anfang einer Spielrunde verbindet sich eine Spielerin mit seiner Wallet zu unserem Spiel.
- Danach zahlt die Spielerin 5 Dollars in einen Fund ein. Dies machen auch alle anderen Mitspieler. Sobald das Geld angekommen ist, wird die Spielerin in eine Wartelobby eingelassen, aus welcher das Spiel gestartet wird.
- Nach Abschluss der Spielrunde, ist die Siegerin klar entschieden. Nun wird automatisch auf die Wallet der Siegerin der Grossteil des Geldes aus dem Fund überwiesen. Die Verliererinnen verlieren dabei den Grossteil ihres Geldes.

3.4 Der chronologische Arbeitsprozess:

3.4.1 Recherche: Was ist Blockchain? Was sind Smart Contracts?

Ich fing meinen praktischen Teil der Arbeit mit einer Recherche an. Mein Vorwissen Bestand lediglich daraus, dass man mit Smart Contracts Funktionen und, entscheidend für unser Projekt, Transaktionen automatisieren konnte. Ich fand heraus, dass Smart Contracts, Kontrakte sind, welche dezentralisiert sind. Dies bedeutet, dass es kein Einzelunternehmen hat, welches die volle Kontrolle über die Daten hat, vielmehr wird die Entscheidungsfindung und Kontrolle auf ein Netzwerk mit vielen Mitgliedern übertragen. So muss nicht auf Vertrauen in eine zentralisierte Organisation gesetzt werden. Smart Contracts ermöglichen das Abschliessen von Abkommen zwischen Einzelpersonen, ohne eine Drittpartei zu haben welche überwachen und kontrollieren muss. So kann man zum Beispiel bei einem Kauf einer Kryptowährung[5], wie zum Beispiel Ethereum, auf einer dezentralisierten Börse, kurz DEX, sicher sein, dass man nicht betrogen wird.[6] Alles wird automatisch Peer-to-Peer, abgewickelt und automatisiert. Eine solche Technologie, kann viele Vorteile haben. So muss zum Beispiel ein Unternehmen nicht mit weiteren Anbietern arbeiten, um eine Bezahlmöglichkeit einzubauen. Was ich noch erfuhr, war, dass diese Smart Contracts auf der Blockchain festgehalten werden. Die Blockchain ist eine Datenbank, welche alle Transaktionen und Ereignisse festhält.[7] Wenn eine Wallet einer anderen Wallet Ethereum schickt, ist die Bestätigung dieser Transaktion für immer auf der Blockchain verfestigt. Jeder kann, wenn er die Adresse einer gewissen Wallet kennt, ihre ganze Aktivität verfolgen. Das gleiche Prinzip wird bei dem Kreieren eines Smart Contracts angewandt. Sobald ein Smart Contract auf die Blockchain hochgeladen ist, wird er unveränderbar. Diese Unveränderbarkeit, wie auch der von manchen Entwicklern Open Source zugängliche Kode der Smart Contracts erlaubt ein System, indem nicht auf Vertrauen in ein Unternehmen gesetzt werden muss. Dazu muss gesagt werden, dass der Code von Smart Contracts nicht immer Open Source ist. Das kann jede Entwicklerin selbst entscheiden. Aber jeder Smart Contract ist Open Target, dass bedeutet, dass der Code zwar nicht öffentlich einsehbar ist, aber mithilfe von Dekompilierern, zurück in die Solidity Programmiersprache dekompiliert, und so einsehbar gemacht werden kann. Nach dieser Recherche, wusste ich, dass Smart Contracts perfekt für das Umsetzten meines Ziels sein würden.

3.4.2 Entscheid für eine Blockchain

Der nächste Schritt war der Entscheid für eine Blockchain. Es gibt sehr viele verschiedene Blockchains, die jeweils spezifische Vorteile, respektive Nachteile haben. Ich wusste, dass man auf der Ethereum Blockchain Smart Contracts programmieren könnte. Jedoch wusste ich aus Erfahrung, dass die Transaktionskosten bei belastetem Netzwerk sehr hoch, circa zwischen 8 und 30 Dollars sind. Deshalb wollte ich eine andere Blockchain suchen, welche sich für mein Endziel besser eignete. Denn ich wollte, dass unser Spiel selbst bei stark belasteten Netzwerken für die Spielerinnen attraktiv wäre. Ich entschied mich für die Binance Smart Chain. Dies ist eine Blockchain von einer der grössten Kryptowährungen Börse der Welt, Binance. Der hauptsächliche Grund war, dass die Transaktionskosten niemals 0.5 Dollars überschritten haben, während ich diese Chain für den Kauf von Kryptowährungen benutzt habe. Was auch für diese Chain sprach, war, dass sie auch Ethereum Virtual Machine kompatibel ist. Was das bedeutete, verstand ich zu der Zeit noch nicht, aber es sollte sehr hilfreich sein, denn es erlaubte mir mit nur wenigen Änderungen im Code, die Ethereum

[5] (Kryptowährung, "https://de.wikipedia.org/wiki/Kryptow%C3%A4hrung")
[6] (What are decentralized exchanges, and how do DEXs work?, "https://cointelegraph.com/defi-101/what-are-decentralized-exchanges-and-how-do-dexs-work")
[7] (Blockchain facts, "https://www.investopedia.com/terms/b/blockchain.asp")

Kontrakte auf BSC hochzuladen. So beschloss ich, als Endziel die Smart Contracts, sollte das Spiel jemals für Benutzer spielbar sein, auf der BSC laufen zu lassen.

3.4.3 Mein erster Smart Contract

Ich folgte einem Tutorial auf YouTube und programmierte meinen ersten Smart Contract. Smart Contracts auf der EVM werden meistens in der Solidity Programmiersprache programmiert.[8] Um Smart Contracts zu testen und benutzen zu können, muss man sie immer auf die Blockchain hochladen. Dazu benutzte ich am Anfang die online Entwicklungsumgebung Remix IDE.[9] Diese erlaubt Smart Contracts einfach hochzuladen und die Funktionen, welche man programmiert hat, gleich in einer simplen Benutzer Oberfläche zu bedienen. Das verschnellert den Prozess um ein Vielfaches und erlaubt Anfängern schnell und einfach eigene Smart Contracts zu entwickeln.

```
1   pragma solidity 0.8.7;
2
3   contract Counter{
4       uint public count = 0;
5
6
7       function getCount() view public returns(uint) {
8           return count;
9       }
10
11
12      function increment() public{
13          count += 1;
14      }
15
16      function decrement() public{
17          count -= 1;
18      }
19  }
```

Abbildung 2: Der Kode von meinem ersten Smart Contract

Mein erster Smart Contract definierte eine integer Variabel, sowie zwei Funktionen, welche erlaubten diese Variabel per Betätigung der Funktion um 1 zu erhöhen, beziehungsweise zu vertiefen. Eine dritte Funktion gibt den momentanen Wert der Variabel zurück.

3.4.4 Tutorial für Ethereum Smart Contracts Entwicklung

Nach einigem Experimentieren wusste ich immer noch nicht wie ich anfangen sollte, meine konkreten Smart Contracts zu schreiben, deshalb entschied ich mich einem langen Tutorial[10] zu folgen. Es war jedoch auf der Ethereum Blockchain, doch da ich nichts zur Entwicklung von Smart Contracts auf der Binance Smart Chain fand, entschied ich auf Ethereum anzufangen und später nach Möglichkeiten umzusteigen. Ich fing an, die Grundkenntnisse von Solidity zu lernen. Um den Einstieg simpel zu halten, wurde im Tutorial auch Remix benutzt. Ich schrieb einen Speicher-Smart Contract, welcher erlaubte einer Integer Variabel, einem String zuzuweisen. Als nächstes lernte ich die geschriebenen Solidity Smart Contracts auf einem Testnetz hochzuladen. Ein Testnetz ist eine Blockchain welche wie die Mainnet funktioniert, aber auf welcher man mit Test-Ethereum die Transaktionen bezahlt. Das Testen der Smart Contracts auf dem Testnetz ist nötig, da es sonst teuer wäre Smart Contracts zu

[8] (Solidity documentation, "https://docs.soliditylang.org/en/v0.8.17/")
[9] (Remix IDE, "https://remix.ethereum.org/")
[10] (Solidity, Blockchain, and Smart Contract Course, "https://youtu.be/M576WGiDBdQ")

schreiben und sie auf der Mainnet immer und immer wieder auszuprobieren und zu testen. Um auf dem Testnetz testen zu können, braucht man eine Wallet. Ich benutzte die MetaMask Wallet, welche eine Browserextension ist, die man für den Browser installieren kann. Auf sogenannten Faucets kann man sich das Test-Ethereum holen. Zu diesem Punkt tauchte wieder die Frage auf, auf welcher Plattform unser Game laufen sollte. Ich entschied, dass es am besten browserbasiert sein sollte, da sich Spielerinnen im Browser einfacher mit ihrer Wallet zu unserem Spiel verbinden könnten. Als Software, welche man herunterlädt, würde es zwar auch gehen, man bräuchte sich dann aber mit QR-Codes und mit einer Wallet auf dem Handy zu verbinden. Der nächste Smart Contract den ich programmierte, hatte die Funktion «fund». Diese Funktion brachte mir erste Ideen wie ich meine Spiel-Contracts machen könnte. Die «fund» Funktion erlaubt das Einzahlen in einen Smart Contract. Das bedeutet, dass ein Benutzer einem Smart Contract Ethereum schicken kann. Das war mein erster Anschluss an eine Idee für meine eigenen Smart Contracts.

3.4.5 Installierung von Visual Studio Code, sowie weiteres Lernen

Das Tutorial, dem ich folgte, wechselte zu einem Zeitpunkt auf Visual Studio Code.[11] Visual Studio Code ist ein Text Editor, welcher eine weite Bandbreite an Erweiterungsmöglichkeiten hat. Ich installierte die Solidity Programmiersprachen-Erweiterung für Visual Studio Code, sowie Python auf meinem Computer und dann war ich bereit in Visual Studio Code zu programmieren. Ich installierte zudem für beide Programmiersprachen Formatierer. Diese ermöglichen automatisches Formatieren und erleichtern so den Überblick über den geschriebenen Kode. Später installierte ich noch Brownie, das ist ein auf Python basiertes Framework, welches das Hochladen und Testen von Smart Contracts vereinfacht.[12] Ich lernte in den nächsten Tagen wie ich in Python mithilfe des Brownie-Frameworks einen Smart Contracts Hochlader bauen konnte. Der Solidity Code bleibt gleich, egal ob man die Smart Contracts auf Remix schreibt oder im Visual Studio Code. Was man im Visual Studio Code selbst programmieren muss, ist die Hochlade-Funktionalität. Mit Anleitung des Tutorials programmierte ich ein Lotto und lernte Smart Contracts zu testen. Nachdem ich ein wenig Erfahrung mit dem Schreiben, Testen und Hochladen von Smart Contracts gesammelt hatte, entschied ich mich mit meinen eigenen Smart Contract anzufangen. Ich ging wieder zurück auf Remix um sich nur um den Solidity Code kümmern zu müssen. Als erstes schrieb ich eine Funktion welche erlaubte von einer Wallet auf eine andere Ethereum zu schicken. Danach baute ich die Funktion «fund» ein, welche ermöglichte, dass eine Wallet in den Smart Contract Ethereum hineinschicken kann. Und dazu schrieb ich auch eine Auszahl-Funktion, in welche ich die Möglichkeit einbaute, zu einer beliebigen Wallet auszuzahlen. Ich hatte somit meine erste Funktionalität erstellt. Ich versuchte diese Smart Contracts aus Visual Studio Code heraus auf das Testnetz hochzuladen. Mithilfe einer Etherscan API konnte ich meine Contracts im Etherscan Blockchain-Explorer kompilieren und dort in einer Benutzer Oberfläche benutzen. Etherscan ermöglicht zugleich auch die Verifikation der Smart Contracts. Dieser Prozess ist wichtig, um die Richtigkeit der Smart Contracts zu bestätigen. Während dieses Vorgangs prüft eine Software, ob der kompilierte Code in den Smart Contracts, derselbe ist wie der Quellkode, welchen man Open Source zur Verfügung stellt. Zur gleichen Zeit spielte ich mit den Gedanken wie ich meine Smart Contracts später mit Tomás seinem Game in Unity verbinden könnte. Ich entschied, dass es am einfachsten wäre, die Benutzer Oberfläche für die Contracts in Unity selbst zu designen, da wir so alles in Unity hätten. Ob es einen Weg gäbe, um genau das zu tun, würde sich nach einer Recherche herausstellen.

[11] (Visual Studio Code, "https://code.visualstudio.com/")
[12] (Brownie documentation, "https://eth-brownie.readthedocs.io/en/stable/")

3.4.6 Funktionalität zum Senden von ERC-20 Tokens

Da wir in unserem Computerspiel immer gleich viel Geld für den Eintritt verlangen würden, wollte ich eine Funktion schreiben, welche ausrechnete, wie viel Ethereum 5 Dollars in einem gegebenen Moment entspricht. Um das zu tun, braucht man eine Adresse, welche den Zugriff auf eine Datenbank mit den letzten Preisen ermöglicht. So rechnete ich aus, wie viel Ethereum 5 Dollars in einem gewissen Moment entsprachen. Das funktionierte. Jedoch wollte ich es noch einfacher für die Benutzerinnen machen, indem die Spieler nicht Ethereum einzahlen müssten, sondern einen Stablecoin. Ein Stablecoin ist ein Token, welcher den Preis einer gewissen Fiat-Geld Währung widerspiegelt. Zum Beispiel USDT, dies ist ein ERC-20 Token, welcher im Wert immer genau dem US-Dollar entspricht. Durch das Einzahlen mit solchen Tokens, könnte man sich die Ausrechnung der Menge von Ethereum sparen, sowie eventuelle Unstimmigkeiten, wie zum Beispiel bei hoher Volatilität auf dem Markt entstehen könnten, vermeiden. Nun also tauchte die Frage auf, wie ich genau die «deposit» und Auszahl-Funktionen mit USDT anstelle von Ethereum machen könnte. Da USDT und jegliche ERC-20 Tokens eigene Funktionsweisen haben, wie man sie verschickt und in einen Smart Contract hineinsendet, tauchten viele Fehler auf, als ich die deposit Funktion mit USDT versuchte. Ich verstand lange nicht, wieso immer ein Fehler auftauchte, wenn ich einem Smart Contract Test-USDT schicken wollte, doch dann fand ich es nach viel Frustration endlich heraus. Der Benutzer muss nämlich beim Smart Contract des Tokens, welcher geschickt werden möchte, eine Genehmigungs-Funktion betätigen. Dabei erlaubt die Benutzerin unserem Contract, ihre ERC-20 Tokens zu bewegen.

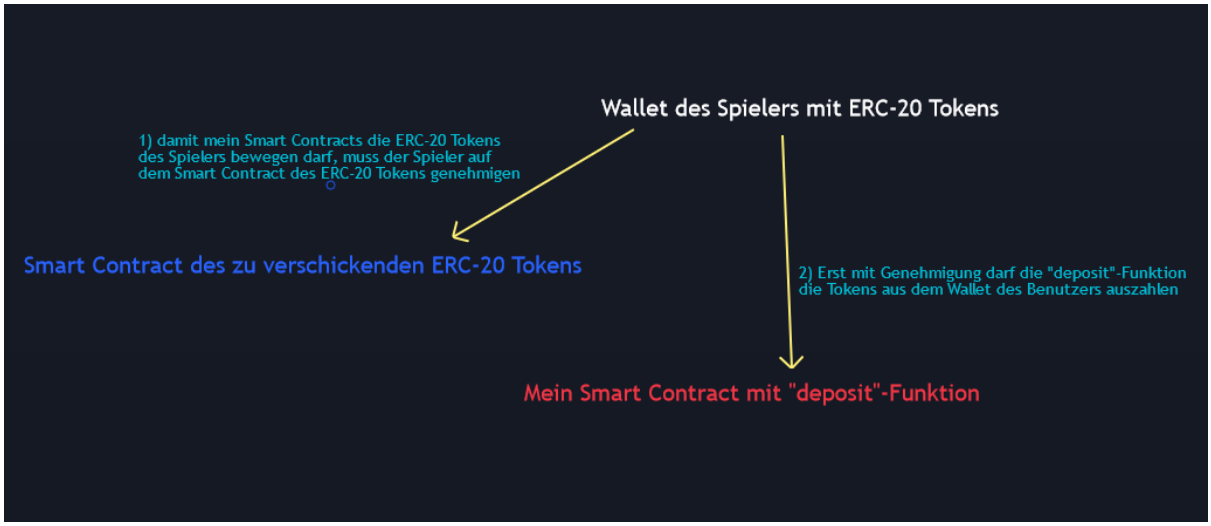

Abbildung 3: Visualisierung der Genehmigungsfunktion, im Smart Contract des ERC-20 Tokens

Ich habe einige Stunden lang versucht die Genehmigungsfunktion des fremden ERC-20 Smart Contracts aus dem eigenen Contract zu aktivieren, bis ich realisierte, dass das nicht geht. Endlich verstand ich auch, wieso ich früher, als ich auf dezentralisierten Börsen Kryptowährungen gekauft habe, immer diese Genehmigungs-Funktionen betätigen musste. Ich verstand, dass ich die Genehmigungs-Funktion später in meine Benutzeroberfläche hineinbauen müsste, und dass jeder Spieler diese selbst betätigen muss. Inzwischen hatte ich den Grossteil meines Solidity Codes beisammen, es war Zeit die Smart Contracts in Unity zu integrieren. Sobald ich das hätte, würde das Verbinden mit Tomás Computerspiel nicht mehr kompliziert sein. Ich würde danach nur mein Unity Projekt in seines hineinkopieren und die Szenen in der richtigen Reihenfolge miteinander verbinden müssen.

3.4.7 Die Integration in Unity mithilfe von Chainsafe Gaming SDK

Um mit der Integration anzufangen, schaute ich mir als Erstes einige Tutorials an. Ich fand eines welches Moralis[13] verwendete, um zu integrieren. Moralis ist eine Software, welche Entwicklern APIs anbietet um die Integration von Blockchain und anderer Web3-Technologie zu vereinfachen. Bei einem grösseren Projekt wäre eine Software wie Moralis auf jeden Fall hilfreich und nötig, jedoch wollte ich mich nicht von einer Drittanbieter-Software abhängig machen. Ich recherchierte weiter herum und fand die Chainsafe Gaming SDK (Chainsafe SDK docs, n.d.). Chainsafe Gaming SDK ist ein Software-Entwicklungs-Kit, welcher Entwicklern erlaubt, Blockchain und Smart Contracts in Unity zu integrieren. Was meine Entscheidung ausserdem beeinflusst hat, ist, dass Chainsafe SDK kostenlos ist, während Moralis kostenpflichtig ist. Mithilfe der Chainsafe SDK erstellte ich in Unity ein Skript für die Genehmigung, sowie für die Einzahlung der ERC-20 Tokens. Um meinen Smart Contract dort zu integrieren, brauchte ich ein ABI meines Smart Contracts. Aus Visual Studio Code heraus lud ich meinen Smart Contract auf das BSC-Testnetz und hatte sie somit auf Bscscan, einem Blockchain Explorer der Binance Smart Chain. Dort konnte ich das JSON-File meines Smart Contracts kopieren und sie in die von Chainsafe SDK zur Verfügung gestellten Beispielkodes hineinbringen. Um zu testen, ob meine Integration funktionierte, brauchte ich einen «WebGL-build» meines Unity Projektes machen. Ein «WebGl-build» ist eine Kompilation eines Unity Projektes, welches im Browser laufen kann. Dann konnte ich mich mit meiner MetaMask Wallet mit den Smart Contracts verbinden und sie von dort heraus betätigen. Es funktionierte, ich hatte eine erste Funktion meines Smart Contracts in Unity integriert.

```
8  public class WebGLApprove20My : MonoBehaviour
9  {
10     [SerializeField]
       1 reference
11     private string contract = "0xbCe98d116cA02A87a2E6c8EDf9597CEd50f3B0a2";
12     [SerializeField]
       1 reference
13     private string spender = "0x7d5A6F6313633Ba8DdbbE893f893761C90dab0eA";
14     [SerializeField]
       1 reference
15     private string amount = "10000000000000000000";
       1 reference
16     private readonly string abi = "[{\"inputs\":[{\"internalType\":\"uint256\",\"name\":\"initialSupply\",\"type\":\"uint256\"}],\'
17
       0 references
18     async public void Approve()
19     {
20         // smart contract method to call
21         string method = "approve";
22         // array of arguments for contract
23         string[] obj = { spender, amount };
24         string args = JsonConvert.SerializeObject(obj);
25         // value in wei
26         string value = "0";
27         // gas limit OPTIONAL
28         string gasLimit = "";
29         // gas price OPTIONAL
30         string gasPrice = "";
31         // connects to user's browser wallet (metamask) to send a transaction
32         try
33         {
34             string response = await Web3GL.SendContract(method, abi, contract, args, value, gasLimit, gasPrice);
35             Debug.Log(response);
36         }
37         catch (Exception e)
38         {
39             Debug.LogException(e, this);
40         }
41     }
```

Abbildung 4: Kode zur Integration der Genehmigungsfunktion in Unity

Im ersten Abschnitt des Codes auf Linien 11, 13 und 15 werden gewisse Variablen zugeschrieben. Die Adresse des Smart Contracts der Genehmigungsfunktion ist auf Linie 11. Es ist also der Smart Contract des ERC-20 Tokens. Auf Linie 13 wird die Adresse von meinem Smart Contract der Variabel «spender»

[13] (Moralis documentation, "https://docs.moralis.io/")

zugeschrieben. Und auf Linie 15 wird die zu genehmigende Menge an Tokens definiert. Man muss dabei wissen, dass die Solidity Programmiersprache nur mit integer Zahlen funktioniert und man eigentlich 18 Nullen wegdenken muss, um auf die eigentliche Menge zu kommen.

In der «Approve»-void wird danach der Name der Methode gefragt. Das ist der Name der Funktion, welche im Smart Contract abgerufen werden sollte. In diesem Fall «approve». Auf Linie 24 werden die zwei Input Variablen angegeben, welche für die Funktion benötigt werden. Und auf Linie 34 wird dann die Funktion ausgeführt.

3.4.8 Zwei interessante Probleme

Um dem Sieger am Ende jeder Spielrunde das Geld zurückzuschicken, brauchte ich die entsprechende Wallet-Adresse. Meine erste Idee bestand daraus, dass ich eine Liste mit Spielerinnen in meinen Smart Contract einprogrammierte. Mithilfe dieser Liste könnte am Schluss entschieden werden, ob ein Spieler wirklich eingezahlt hat, und danach würde die Liste gelöscht werden. Hier entstand aber ein Problem, und zwar wollte ich, dass mehrere Runden gleichzeitig laufen könnten. Wenn ich aber in einem Contract eine Liste mit den Spielerinnen machen würde, welche in den Smart Contract Geld hineinschicken, dann durchmischen sich die Listen, wenn parallel mehrere Runden laufen und man könnte nicht die richtigen Listen mit den richtigen Spielern zum entsprechenden Zeitpunkt löschen. Nach einigem Nachdenken habe ich entschieden, dass ich Spielerinnen, welche in einer Runde gemeinsam sind, nicht in einer Liste gruppieren würde, da ich aus einem Skript in Unity heraus die Adresse der verbundenen Wallet abrufen konnte. Der Smart Contract würde dann einfach wie eine grosse Schüssel funktionieren, in welche Spieler Geld einzahlen. Mithilfe verschiedener Auszahl-Funktionen, könnten dann Spielerinnen jeweils den ihnen zustehenden Beitrag wieder Auszahlen. Ich müsste dann aber eine andere Funktionalität zur Bestätigung des Sieges einer Wallet machen, da die Verifizierung das Spieler wirklich eingezahlt haben, nicht durch eine Speicherung dieser Adressen in einer Liste geschehen würde. Dies war das zweite interessante Problem.

Abbildung 5: Funktionsweise des "InGame"-Tokens

Eine weitere Schwierigkeit stellte das Bestätigen und Verifizieren, dass ein Spieler wirklich eine gewisse Platzierung eingenommen hat. Es ist nämlich so, dass ohne eine solche Bestätigung jede Benutzerin, wenn sie nur die Adresse meines Smart Contracts hat, sich Tokens auszahlen könnte welche von anderen Spielerinnen eingezahlt wurden. Um das zu verhindern, brauchte ich eine Möglichkeit zu finden, damit wirklich nur Spieler, welche auch zu Beginn eingezahlt haben, die «Auszahl»-Funktion betätigen könnten. Ich wollte zuerst eine «OnlyOwner»-Funktion machen, welche nur von dem Ersteller des Smart Contracts, also von mir, betätigt werden kann. Jedoch konnte ich kein einziges Tutorial finden, welches das automatische Betätigen einer solchen Funktion

aus Unity heraus ermöglichen würde. Chainsafe SDK hatte keine Möglichkeit eingebaut, um im Hintergrund Funktionen automatisch mithilfe eines Private Keys aktivieren zu lassen. Lediglich, um mithilfe des Private Keys eine Transaktion zu schicken. Und da kam ich auf die Idee, dass ich meinen eigenen ERC-20 Token machen könnte. Und dieser wird im Hintergrund, sobald die Gewinnerin in die entsprechende Szene kommt, automatisch zu ihrer Wallet geschickt. Danach kann die Spielerin selbstständig die Auszahl-Funktion betätigen. Diese prüft zuerst, ob ein Spieler eine genügende Menge des «InGame»-Tokens hat, und tauscht nach Sicherstellung diese «InGame»-Tokens mit USDT oder BUSD aus.

3.5 Die genaue Struktur der endgültigen Smart Contracts:

Der endgültige Smart Contract besteht aus zwei hauptsächlichen Funktionen. Der «depositTokens» und der «withdrawal» Funktion. Im Code meines Smart Contracts definiere ich zunächst das ERC-20 Token Interface. In diesem Interface werden Funktionen definiert, welche oft im Umgang mit ERC-20 Tokens benutzt werden.

3.5.1 Die «Deposit»-Funktion

```
44      function depositTokens(uint256 amount) public {
45          // To be able to call this function, testBUSd needs to be approved as well as the Smart Contract needs to have some ENTR to
46          uint256 playersUsdcBalance = BUSD.balanceOf(address(msg.sender));
47          require(amount > 0);
48          //Checking whether the wallet has enough tokens
49          require(playersUsdcBalance >= amount * 10**18);
50
51          //BUSD get transferred to the contract, while ENTRToken gets transferred from contract to player
52          BUSD.transferFrom(msg.sender, address(this), amount * 10**18);
53          ENTRToken.transfer(msg.sender, amount * 10**18);
54
55          // update staking balance
56          depositedBalance[msg.sender] =
57              depositedBalance[msg.sender] +
58              amount *
59              10**18;
60
61          // Here the total amount of times a wallet has called the deposit function is tracked
62          // And the player is added to the total players list, but only when he plays for his first time
63          uint256 result = timesDeposited(msg.sender) + 1;
64          playerTimesDeposited[msg.sender] = result;
65          if (result == 1) {
66              allPlayers.push(msg.sender);
67          }
68      }
```

Abbildung 6: Die "Deposit"-Funktion

Die «deposit»-Funktion ermöglicht das Einzahlen von ERC-20 Tokens in meinen Smart Contract. Die Funktion nimmt einen Input, nämlich eine uint256 welche als «Menge» definiert wird. Auf Linie 46 definiere ich die Menge von BUSD welche eine Spielerin auf ihrer Wallet hat. «Msg.sender» ist immer die Adresse der Wallet, welche gerade die Funktion betätigt. Ich prüfe auf Linie 47, dass auf jeden Fall mehr als 0 eingezahlt wird, und danach wird geprüft, ob der Spieler genug Tokens auf seiner Wallet hat. Die Menge, welche geschickt wird, muss vor dem Betätigen dieser Funktion vom Besitzer auf dem Smart Contract des ERC-20 Tokens genehmigt werden, erst danach hat ein fremder Smart Contract Zugriff auf die ERC-20 Tokens des Benutzers. Danach werden auf Linien 52 und 53 BUSD-Tokens von der Wallet des Spielers zum Smart Contract gesendet, während vom Smart Contract zur Benutzerin der «ENTRToken» transferiert wird. Diesen ERC-20 Token habe ich erstellt, um zu verifizieren, dass ein Spieler wirklich eingezahlt hat, und ihn erst dann in die Spiellobby zu lassen.

Der endgültige Smart Contract ist abgestimmt auf die Möglichkeiten, welche ich bei der Integrierung in Unity hatte. So sende ich zum Beispiel in meinem Smart Contract während des Einzahlens einen eigenen Token auf die Wallet des Spielers. Wenn man nun den Smart Contract alleinstehend

betrachtet, dann ergibt dies keinen Sinn. Integriert in Unity macht es dann aber wiederum Sinn. Es ermöglicht die Kontrolle, ob eine Spielerin wirklich eingezahlt hat. In Unity habe ich ein Skript erstellt, welches prüft, ob die Menge von Tokens auf dem Wallet der Spielerin einen gewissen Wert überschreiten. Sobald diese Bedingung erfüllt ist, wird die Spielerin in die Wartelobby hineingelassen.

3.5.2 Die Auszahl-Funktion

Um das Auszahlen nur möglich zu machen für Spieler welche wirklich gewonnen haben, musste ich eine Möglichkeit der Bestätigung durch uns hineinbauen. Das war nötig, da sich ansonsten jeder Benutzer Tokens mithilfe einer Netzwerk-Suchmaschine herauszahlen könnte, da mein Smart Contract Open Source ist. Ich entschied mich einen eigenen ERC-20 Token zu erstellen. In einem Skript im Back-End unseres Games habe ich eine Funktion geschrieben, welche mithilfe des Private Keys die Transaktionen aus meinem Admin Wallet erlaubt. Das Admin Wallet ist keine spezielle Art von Wallets, sondern ein von mir nur für den Bestätigungszweck ausgewähltes Wallet. Wenn ein Spieler in die entsprechende Szene kommt, werden dem Spieler im Back-End unsere «InGame»-Tokens geschickt.

```
71      function withdrawalPlayers(uint256 _amount) public {
72          /*This function allows the withdrawal of BUSD from the Smart Contract, to check whether the player
73          is allowed to withdraw their tokens, the balance of the InGame-Token (IGToken) is checked
74          */
75
76          uint256 dexBalance = BUSD.balanceOf(address(this));
77          uint256 playersBalance = IGToken.balanceOf(address(msg.sender));
78
79          require(playersBalance >= _amount, "Not enough tokens in wallet");
80          require(_amount <= dexBalance, "Not enough tokens in reserve");
81
82          BUSD.transfer(msg.sender, _amount);
83          ENTRToken.transferFrom(msg.sender, address(this), _amount);
84          IGToken.transferFrom(msg.sender, admin, _amount);
85
86          depositedBalance[msg.sender] = 0;
87      }
```

Abbildung 7: Die Auszahl Funktion

In der Auszahl-Funktion wird geprüft, ob der Spieler, welcher die Auszahl-Funktion drückt, genug von diesen «InGame»-Tokens hat. Wenn dann die Funktion aufgerufen wird, werden gleichzeitig die «InGame»-Tokens von der Spielerin zurück zum Admin Wallet geschickt, sowie die BUSD-Tokens aus dem Smart Contract heraus zum Wallet des Spielers geschickt. Diese Absicherung mit dem eigenen Token, ist nötig, damit kein Benutzer im Etherscan unseren Contract finden kann und sich von dort Kryptowährungen auszahlen kann. Denn mein Smart-Contract ist Open Source, das heisst jeder der eine Wallet hat, kann sich mit ihm verbinden und ihn benutzen, wenn er die Adresse des Smart Contracts kennt.

3.5.3 Zusätzliche Variablen

Ich habe zusätzlich noch zwei weitere Variablen eingebaut. Diese sind nicht unbedingt notwendig, sie ermöglichen mir aber, Daten zu sammeln über unsere Spieler und Spielerinnen. Die erste Variabel ist ein Mapping. Ein Mapping ist eine Struktur, welche eine Variabel einer anderen zuweist. Ich habe das Mapping so programmiert, dass ich eine Wallet Adresse einer Integer Zahl zuweise. Diese Integer zeigt die Anzahl der Aufrufung der «deposit»-Funktion, welche eine Wallet, gemacht hat. So kann ich verfolgen, wie viele Male eine gewisse Wallet gespielt hat. Die weitere Variabel welche definiert habe, wird auch zur Sammlung der Daten benutzt. Es ist eine Liste, welche die Adressen aller Wallets, die jemals die «deposit»-Funktion aufgerufen haben beinhaltet. Diese beiden Variablen, tragen nichts zur

eigentlichen Funktionalität des Spiels bei, sind aber für mich interessant und könnten einen späteren Ausbau der Suchtprävention ermöglichen.

3.6 Wie wir den Jugendschutz online sicherstellen:

Da mein Teil der Arbeit mit der Integrierung der Spielmöglichkeit, um Geld zu tun hat, schreibe ich hier noch etwas zum Jugendschutz online. In Europa muss man auf jeder Finanzplattform die Identität verifizieren und die ID einscannen. Diese Plattformen kann man erst benutzen, wenn man 18 Jahre alt ist. Deshalb haben wir uns entschieden diese Verantwortung abzugeben. Wir verlassen uns darauf, dass jegliche Plattformen, auf welchen man in der Schweiz Kryptowährungen kaufen kann, diesen Prozess der Bestätigung der Identität eingebaut haben. Somit erwarten wir, dass unser Spiel, welches nur spielbar ist, wenn man in Besitz von Kryptowährungen ist, nur von volljährigen Personen gespielt werden kann. Es ist uns nämlich wichtig, dass keine minderjährigen Personen durch den Verlust von Kryptowährungen in unserem Spiel in schwierige Situationen geraten. Um diesen Jugendschutz in Zukunft zu hundert Prozent sicherzustellen, könnte man eine Accounterstellungs-Funktionalität, mit Identitätsverifizierung, einbauen. Dies würde natürlich die Einfachheit der Spielerfahrung reduzieren.

3.7 Wie wir die Geldwäscherei vermeiden, oder vielleicht doch nicht:

Das Ziel war es ein System zu entwickeln, in dem Geldwäscherei nicht möglich ist. Ich hatte dieses Problem schnell bei Seite geschoben, denn schlussendlich ist die Blockchain transparent und hält jede Transaktion fest. Ausserdem wird bei Geldwäscherei mithilfe von Kryptowährungen das Geld meistens in dem Moment gewaschen, in welchem man Fiat Geld in Kryptowährung umtauscht. Das grundlegende Problem der Geldwäscherei haben wir so an andere Unternehmen ausgelagert. Jedoch habe ich in letzter Zeit lange nachgedacht, ob man mit dem Game, welches wir kreiert haben, nicht doch Kryptowährungen waschen kann. Und ich bin zu dem Schluss gekommen, dass ja theoretisch kann dieses System dazu benutzt werden, um Geld zu waschen. Aber nur Geld welches schon in Form von Kryptowährungen gestohlen wurde. Das Problem hierbei ist, dass mehrere Benutzer gleichzeitig ihre Tokens in einen Fund einzahlen, dabei durchmischen sich die Währungen. Wenn nun die Benutzerin, die das Geld wäscht, wieder die gleiche Menge gewinnt, dann ist das Geld sauber, denn die Herkunft der exakten Tokens geht verloren. Trotzdem ist das Risiko, dass mit unserem Spiel bedeutende Mengen an Kryptowährungen gewaschen werden, sehr gering, da nur jeweils maximal mit 5 Dollars eingezahlt werden kann. Bevor wir das Spiel online stellen würden, müsste man natürlich an dieser Stelle etwas abändern, wie auch die entsprechenden Rechtsgrundlagen durchlesen.

3.8 Suchtprävention bei Videospielen:

In unserem Game können unsere Spieler um Geld spielen. Bei Videospielen besteht immer ein Risiko, dass man eine Sucht entwickelt. Wenn Geld eine Rolle spielt, führt eine Sucht zu noch grösserem Schaden. Deshalb finde ich es wichtig, dass ich diesen Aspekt in meiner schriftlichen Arbeit erwähne. Für Casinos sowie Lottos gibt es Regulierungen. Für online Spiele wie unser, gibt es keine festen Regulierungen, weil das Internet Grenzen übergreifend ist. Deshalb ist es wichtig, dass Spielhersteller selbst die Regulierungen treffen, und ein System zustande bringen, welches Spielern hilft, eine Sucht vorzubeugen. Leider sind sehr viele Computerspiele genauso ausgelegt, dass sie möglichst schnell süchtig machen. Das ist sehr problematisch und zerstört das Leben vieler Menschen. Wir wollen mit unserem Game nicht dazu beitragen und werden versuchen, unsere Website, sollte unser Game jemals wirklich online gestellt werden, so auszurichten, dass unsere Spielerinnen sich selbst bewusst sind, welche Risiken sie eingehen. Um in Zukunft ein System zur Suchtprävention einzubauen, habe

ich eine Variabel definiert, welche trackt, wie viele Male ein gewisser Spieler gespielt hat. So könnte man zum Beispiel ein Limit an Spielrunden innerhalb einer gewissen Zeitperiode erstellen.

4 Schlussteil:

4.1 Was ich gelernt habe:

Im Rahmen der Matur Arbeit habe ich Smart Contracts für ein Multiplayer-Spiel entwickelt. Ich habe sehr vieles gelernt während dieser Arbeit. Ich habe die Technologie der Blockchain in Praxis kennengelernt. Aber auch, wie ich die meisten Schwierigkeiten, welche aus dem Nichts auftauchen bewältigen kann. Nachdem ich viele Probleme und Engpässe bewältigt habe, habe ich realisiert, dass es immer irgendeinen Ausweg gibt, um ein Problem zu lösen. Ich habe gelernt, wie ich bei dem Auftauchen solcher Probleme selbstständig vorgehe. Ich hatte mehrmals solche Schwierigkeiten, dass ich im Moment, in welchem sie auftauchten, wirklich nicht wusste, wie ich sie lösen sollte. Doch nach einiger Recherche, oder Nachdenken wie ich das Problem umgehen kann, konnte ich jedes Problem lösen. Ich denke, dass genau diese Fähigkeiten flexibel unterschiedliche Probleme zu meistern, sehr wichtig ist. Ich habe auch gelernt Solidity als Programmiersprache zu benutzen, um Smart Contracts damit zu schreiben. Ich kann eigene Smart Contracts entwickeln, Variablen wie auch Funktionen definieren, entwickelte Smart Contracts auf die Blockchain hochladen und verifizieren lassen und sie schlussendlich mithilfe von Unity in eine Benutzer Oberfläche hineinbringen. Das Großartige ist, dass ich nicht nur Tutorials nachgemacht, sondern, dass ich mithilfe von Tutorials die Programmiersprache gelernt habe, sodass ich anschliessend frei das Gelernte in meinem eigenen Projekt umsetzten konnte.

4.2 Meine grösste Schwierigkeit:

Ich denke, dass bei meinem Projekt nicht der Code an sich sehr kompliziert ist, sondern dass der Prozess die eigentliche Herausforderung darstellte. Denn mein Smart Contract hat lediglich 95 Linien Code. Die Schwierigkeit war es, herauszufinden wie man es genau machen konnte, sowie die Integration des Smart Contracts in Unity. Es brauchte viel Zeit, viele Fehler und sehr viel Ausprobieren. Weil mein Projekt recht spezifisch war, gab es kein Tutorial, welches genau mein Projekt zeigte. Vielmehr musste ich aus den Tutorials lernen, den Code gut verstehen, und danach kombinierend anwenden. Die grösste Herausforderung für mich war das Anfangen. Ich wusste nicht, wo ich anfangen sollte, denn ich hatte weder eine Ahnung wie man solche Smart Contracts schrieb, noch wusste ich in welcher Programmiersprache oder in welcher Entwicklungsumgebung. Alle Tutorials, welche ich im Internet gefunden habe, waren unterschiedlich. In einem Tutorial wurde Python benutzt und im anderen JavaScript, um das Framework zum Hochladen zu programmieren. Jeder Programmierer machte es ein wenig anders, deshalb war es schwierig eine richtige, vollständige Weise zu finden. Wie ich diese Schwierigkeit umging, war, dass ich mich auf ein 15-stündiges Tutorial einliess, welches alles was man braucht, um Smart Contracts auf Ethereum zu programmieren in sich hatte. So konnte ich wenigstens in die richtige Richtung gehen, auch wenn es nicht genau das abdeckte, was ich machen wollte.

4.3 Was ich anders gemacht hätte:

Ich denke, was ich vor allem anders gemacht hätte, ist, von Anfang an ein längeres Tutorial zu verfolgen, anstatt kurze anzuschauen, denn dann lernt man es vollständig und nicht in Stücken. Ich hätte früher angefangen an eigenen Smart Contracts zu schreiben. Denn als ich nach einem Monat Tutorial Verfolgen und Nachmachen probiert habe meine eigenen Smart Contracts zu schreiben, wusste ich nicht wie anzufangen. Ich hatte einfach den Code abgeschrieben, ohne ihn zu verstehen.

Ich denke man sollte schnell weggehen vom Tutorial, nachdem man ein gewisses Basisverstehen hat. Und dann auch sofort das eigene Projekt versuchen zu entwickeln, denn so bringt man sich selbst zum Nachdenken. Man bemerkt wirklich erst ob man etwas verstanden hat, wenn man es selbst flexibel anwenden muss.

Was ich bezüglich des Produktes anders gemacht hätte, ist vielleicht eine andere Möglichkeit zu finden, um die Smart Contracts zu integrieren. Denn diese Version sieht zwar recht gut aus und funktioniert auch. Aber es werden zur Bestätigung der Transaktionen unnötigerweise ERC-20 Tokens, welche ich erstellt habe, herumgeschickt. Wenn man aus Unity heraus eine Möglichkeit hätte, Events auf der Blockchain abzuhören, oder Funktionen mithilfe des Private Keys zu bestätigen, dann könnte man den Smart Contract vereinfachen und alles professioneller gestalten können. Wie ich es gemacht habe, ist ein kreativer Weg um das Problem von Chainsafe SDK's Unvollständigkeit zu umgehen.

4.4 Die Arbeit zu zweit, Vorteil und Schwierigkeiten

Grundsätzlich fand ich es wirklich hilfreich meine Matura Arbeit mit einem Arbeitspartner absolviert zu haben. Ich hatte immer einen Arbeitskollegen, der im gleichen Boot wie ich steckte. Der grösste Vorteil denke ich ist, dass man sich gegenseitig motivieren kann, weiterzuarbeiten und sein Bestes zu geben. Man konnte die eigenen Ideen mit dem Partner besprechen, und war bei Entscheidung nicht nur auf sich selbst gestellt. Gleichzeitig entstehen besonders bei einer Arbeit, bei welcher am Ende beide Teilaufgaben zu einem Produkt zusammenkommen, einige Herausforderungen. So muss man bei der Zeitplanung darauf achten, dass man es in einer solchen Weise synchronisiert, dass beide zum ungefähr gleichen Zeitpunkt mit ihren Teilaufgaben fertig sind. Kommunikation ist dabei sehr wichtig. Ist man hinter dem Zeitplan, muss man ein Ziel vielleicht ein wenig abändern. Man kann sich dabei die ganze Arbeit nicht nur für sich selbst passend zeitlich einteilen, sondern muss Rücksicht auf den Arbeitskollegen geben.

Es gibt also einige Herausforderungen, wenn man zu zweit arbeitet, aber ich denke die Vorteile sind um einiges schwerwiegender als die Herausforderungen. Ich würde jedem, der ein interessantes, vielleicht grösseres Projekt machen möchte, raten, sich zu überlegen, ob man nicht vielleicht eine Klassenkameradin oder Klassenkameraden hat, welcher ähnliche Interessen hat. Denn zu zweit kann man ein ordentlich grösseres und unter Umständen interessanteres Projekt gestalten.

4.5 Ziele

Ich habe all meine hauptsächlichen Teilaufgaben erfüllt. Zwischendurch musste ich einen Umweg nehmen, ich bin aber trotzdem am Ziel angelangt. Zum Beispiel war es mein Ziel, die Smart Contracts auf der Binance Smart Chain zu entwickeln, weil diese aus mehreren Gründen (Siehe: Entscheid für eine Blockchain) anderen Blockchains überlegen war. Ein Problem, welches auftauchte, war, dass ich nur wenige Tutorials für die Entwicklung der Smart Contracts auf der BSC finden konnte. Weil ich keinen Weg sah, um dieses Problem zu umgehen, entschied ich mich in einem Moment, dass ich die Smart Contracts vorerst auf der Ethereum Blockchain programmieren würde, und sie eventuell, wenn ich am Ende Zeit hätte, auf die BSC umprogrammierte. Es stellte sich heraus, dass die Entwicklung der Smart Contracts auf der BSC genau gleich wie auf der Ethereum Blockchain ist, da beide Blockchains von der Ethereum Virtual Machine gesteuert werden. Ich musste nur an wenigen Stellen den Code etwas abändern, um meinen Contrakt auf die BSC zu laden.

4.6 Tipps für Programmierprojekte

Die Schwierigkeit bei einem Programmierprojekt ist, dass Unvorhergesehenes sehr oft passiert. Es tauchen verschiedene Probleme auf, welche man lösen muss, um weiterzukommen. Deshalb reicht es nicht aus, spät anzufangen. Man muss genug Pufferzeit einplanen, um genau diese spontanen Probleme zu beseitigen. Ich persönlich habe ziemlich früh mit dem Entwickeln angefangen, weil ich nicht genau wusste wie viel Zeit ich benötigen würde. Das hat sich als gute Idee herausgestellt, da oft Probleme auftauchten, für die ich mehrere Tage Zeit brauchte, um sie zu lösen. Man muss sich ausserdem bewusst sein, dass man das Arbeitsvolumen nicht perfekt einschätzen kann. Es ist eine Arbeit, bei welcher man bereit sein muss, oft allein für mehrere Stunden an einem Problem herumzuknobeln. Man kann es sich vorstellen wie eine knifflige Matheaufgabe, bei welcher man lange dransitzt, ohne auf die Lösungen schauen zu dürfen, und erst zur nächsten Aufgabe gehen kann, nachdem man die Vorherige gelöst hat. Man kann versuchen Hilfe zu holen, im Internet. Aber auch da geht es oft ums Ausprobieren. Jeder der also mit den Gedanken herumspielt, eine Arbeit im Fach Informatik anzufangen, sollte sich dieser Aspekte bewusst sein. Ich will aber damit nicht alle wegschrecken, denn gleichzeitig denke ich ist es eine Arbeit, in welcher man sehr vieles lernt. Unter anderem wie man Probleme löst welche hoffnungslos erscheinen. Dies ist eine Fähigkeit, welche man in jegliche weiteren Bereiche des Lebens mitnehmen kann. Eine Programmierarbeit ist auch sehr erfüllend. Denn je kniffliger ein Problem vorerst erscheint, desto mehr freut man sich, wenn es endlich nach vielen Stunden harter Arbeit funktioniert.

5 Quellenverzeichnis

Blockchain facts, "https://www.investopedia.com/terms/b/blockchain.asp". (kein Datum). Von Investopedia: https://www.investopedia.com/terms/b/blockchain.asp abgerufen

Brownie documentation, "https://eth-brownie.readthedocs.io/en/stable/". (kein Datum). Von eth-brownie: https://eth-brownie.readthedocs.io/en/stable/ abgerufen

Chainsafe SDK docs. (kein Datum). Von chainsafe.io: https://docs.gaming.chainsafe.io/ abgerufen

Kryptowährung, "https://de.wikipedia.org/wiki/Kryptow%C3%A4hrung". (kein Datum). Von Wikipedia: https://de.wikipedia.org/wiki/Kryptow%C3%A4hrung abgerufen

Moralis documentation, "https://docs.moralis.io/". (kein Datum). Von moralis: https://docs.moralis.io/ abgerufen

Remix IDE, "https://remix.ethereum.org/". (kein Datum). Von remix: https://remix.ethereum.org/ abgerufen

Solidity documentation, "https://docs.soliditylang.org/en/v0.8.17/". (kein Datum). Von Soliditylang.org: https://docs.soliditylang.org/en/v0.8.17/ abgerufen

Solidity, Blockchain, and Smart Contract Course, "https://youtu.be/M576WGiDBdQ". (kein Datum). Von YouTube: https://youtu.be/M576WGiDBdQ abgerufen

Visual Studio Code, "https://code.visualstudio.com/". (kein Datum). Von Visual Studio Code: https://code.visualstudio.com/ abgerufen

Was ist Mainnet und Testnet?, "https://portalcripto.com.br/de/Was-ist-Mainnet--und-Testnet-Kryptow%C3%A4hrungsentwicklung%3F/". (kein Datum). Von https://portalcripto.com.br/de/Was-ist-Mainnet--und-Testnet-Kryptow%C3%A4hrungsentwicklung%3F/ abgerufen

What are decentralized exchanges, and how do DEXs work?, "https://cointelegraph.com/defi-101/what-are-decentralized-exchanges-and-how-do-dexs-work". (kein Datum). Von cointelegraph: https://cointelegraph.com/defi-101/what-are-decentralized-exchanges-and-how-do-dexs-work abgerufen

What Are ERC-20 Tokens on the Ethereum Network?, "https://www.investopedia.com/news/what-erc20-and-what-does-it-mean-ethereum/". (kein Datum). Von Investopedia: https://www.investopedia.com/news/what-erc20-and-what-does-it-mean-ethereum/ abgerufen

What Are Smart Contracts on the Blockchain and How They Work?, "https://www.investopedia.com/terms/s/smart-contracts.asp". (kein Datum). Von https://www.investopedia.com/terms/s/smart-contracts.asp abgerufen