

CAS BIDA 2018 Semesterarbeit

# Lotterie als Smart Contract

Orlando Tomás

Version 1.0, 2019-04-04

# Management Summary

In dieser Arbeit wurde eruiert, was das geeignetste Verfahren ist um eine Zufallszahl in einem Smart Contract Umfeld zu generieren. Anhand eines Lotterielos Verfahren wurde diese Problemstellung untersucht. Ziel ist einen Zufallszahlengenerator zu entwerfen, dem es keinem Akteur ermöglicht, seine Chancen im Voraus zu berechnen.

Während der Einarbeitung in diesem Thema wurden Schwierigkeiten identifiziert, dass das erstellen von Zufallszahlen erschweren. Zu den Hauptproblemen gehören: Smart Contracts Zustände sind für jedermann einsehbar und einbeziehen von externen Oracle sind begrenzt. Anhand einer Problemanalyse wurden drei Lösungsvarianten erarbeitet.

## *Variante A*

Smart Contract bezieht die Zufälligkeit von einem externem Oracle Service Anbieter. Welches nicht unter der Kontrolle des Smart Contract Besitzer befindet.

## *Variante B*

Die Input Quellen für den Zufallsgenerator sind Meta Daten von den Teilnehmern und einem zukünftigen generierten Block.

## *Variante C*

In dieser Variante kommt die Zufälligkeit von den Teilnehmern und mit Hilfe vom Commitment-Verfahren werden diese Input Daten bis zur Ziehung geheim gehalten.

## *Empfehlung*

Die Variante A bedingt ein grosses Vertrauen am externen Oracle Service Anbieter. Dieser kann den Spiel Verlauf kontrollieren. Aus diesem Grund scheidet diese Variante aus. Die Variante B und C scheinen valable Lösungen zu sein. Wobei in der Variante C geklärt werden muss, wie sich der Smart Contract verhalten soll, wenn ein Teilnehmer nicht bei der Aufdecken Phase teilnimmt. Schlimmstenfalls würde das Spiel sich in Tiefschlaf versetzen. Aus diesen Gründen empfiehlt der Autor die Variante B.

# Inhaltsverzeichnis

Aufgabenstellung .....	1
Ziele .....	1
Abgrenzungen.....	1
Methodik .....	2
Technische Grundlagen .....	3
Blockchain .....	3
Lotterie .....	7
Proof of Concept .....	8
Zufallszahlen.....	8
Zeitbasierender Trigger.....	10
Schlussfolgerung.....	10
Lösungsraum.....	11
Entscheid.....	13
Prototyp.....	14
Zufallszahl Generator .....	14
Production-Ready? .....	15
Zielerreichung .....	16
Erkenntnisse .....	17
Quellenverzeichnis .....	18
Appendix A: Smart Contract .....	20
Appendix B: React JS Web Entry Point.....	24
Appendix C: web3.js Provider .....	28

# Aufgabenstellung

## Ziele

Folgende Tabelle enthält die Definitionen der Ziele welche in dieser Arbeit erreicht werden sollen. Dabei wird zwischen “Must” Zielen (ZM) und optionalen Zielen (ZO) unterschieden. Die optionalen Ziele werden verfolgt, falls gegen Ende der Arbeit noch genügend Zeit zur Verfügung steht.

ZM1	Lotterielos Use Case als Smart Contract abbilden.
ZM2	Einschränkungen von Naiven Zufallszahlengenerator im Smart Contract Umfeld durchleuchtet.
ZO1	Einbindung von externen Oracle im Smart Contract

Auf die Erreichung dieser Ziele und deren Bewertung wird am Ende dieser Arbeit nochmals eingegangen. Abgesehen von diesen messbaren Zielen wird sich der Autor fundiertes Wissen in Blockchain und Smart Contract aneignen und vertiefen.

## Abgrenzungen

Um den Rahmen dieser Arbeit nicht zu sprengen, wird vor allem auf die Smart Contract Aspekte eingegangen. Dabei werden folgende, in der Tabelle aufgeführten, Aspekte explizit nicht durchleuchtet. Diese Punkte müssen gegebenenfalls in einer weiteren Arbeit analysiert werden.

A1	Entwickeln oder Aufbau eines externen Zufallszahl Oracle.
A2	Authentisierung
A3	Entwickeln eines sicheren und optimalen Zufallszahl Generator.

# Methodik

Um einen sicheren Zufallszahlgenerator entwerfen zu können, wird ich im Verlauf meiner Semesterarbeit folgende Phasen durchlaufen.

## *Phase 1 - Technische Grundlagen*

Um ein Grundsätzliches Verständnis für Blockchain und Smart Contract herzustellen, werden die technische Grundlagen durchleuchtet und beschrieben. Dies dient als Basis für das weitere Vorgehen.

## *Phase 2 - Lotterielos Spielprinzip verstehen*

Sich mit dem Spielprinzip und rechtlichen Grundlagen vom Lotterielos auseinandersetzen.

## *Phase 3 - Entwicklungsumgebung aufbauen*

Aufbau einer lokalen Umgebung um den Smart Contract entwickeln und testen zu können.

## *Phase 4 - Programmiersprache erlernen*

Erlernen der Programmiersprache um den Smart Contract abzubilden zu können.

## *Phase 5 - Smart Contract*

Den Smart Contract entwickeln und sich mit den Zufallszahlen auseinandersetzen

# Technische Grundlagen

## Blockchain

Blockchain hat seinen Ursprung in der darauf aufbauende dezentrale Kryptowährung Bitcoin. Das Konzept samt der Technologie wurde im Whitepaper "Bitcoin: A Peer-to-Peer Electronic Cash System" im November 2008 unter dem Pseudonym Satoshi Nakamoto über eine verschlüsselte E-Mail-Adresse veröffentlicht [bw].

Im Whitepaper wird der problematische Umgang mit monetären Werten thematisiert. Den zentralen Institutionen, wie unter anderem Banken, Versicherungen oder Regierungen, wird im Umgang mit monetären Werten ein Vertrauensmissbrauch vorgeworfen. Um einem Vertrauensmissbrauch auf einer systematischen Ebene vorzubeugen, wurde die Bitcoin Blockchain Technologie entworfen. Das im Sinne von Satoshi Nakamoto ein vollkommen und transparentes Ökosystem im internationalem Handeln ermöglichen soll.

Im grossen und ganzem kann die Blockchain als öffentliches digitales unveränderbares Transaktion Logbuch betrachtet werden, dass von einem dezentralen System verwaltet und genutzt wird. Transaktionen werden in der Blockchain als Block abgelegt. Jeder neue Block referenziert den vorherigen Block, wodurch sich eine lineare und chronologische Abfolge von Blöcken ergibt.

Für die Verknüpfung innerhalb eines Blockes und der Block Kette wird das Merkle Tree Konzept angewendet. Merkle Tree ist ein Hash Baum in dem jeder Datenblock und jeder Blattknoten mit dem Hash eines Datenblocks und jeder Nicht-Blattknoten mit dem kryptografischen Hash der Labels seiner Kind Knoten beschriftet ist. Dieses Verfahren wird genutzt um Konsistenzprüfung, Datenverifizierung und Datensynchronisation bei grossen Mengen an Daten zu gewährleisten. Im Fall der Blockchain wird für jede Transaktion innerhalb eines Blockes ein Hash erstellt und gemäss Merkle Tree verknüpft und mit dem vorherigem Block Hash gehasht.

Fürs Verständnis dieser Semesterarbeit ist es essentiell folgende Blockchain Charakteristiken zu kennen:

Unveränderbarkeit	Die Blockchain Traktionskette ist unveränderbar. Wird ein Block in die Kette hinzugefügt, kann dies nicht mehr modifiziert oder entfernt werden.
Dezentral	Die Blockchain ist ein verteiltes Netzwerk, welches eine direkte Interaktion der Teilnehmer ermöglicht und ein offenes System darstellt.
Integrität	Mit kryptografischen Algorithmen wird die Konsistenz der Kette gewährleistet.

Validität	Jeder Block und seine abgebildete Transaktionen werden vom Netzwerk validiert. Erst dann synchronisieren die Nodes die aktualisierte Block Kette.
-----------	---

## Hashfunktion

Eine Hashfunktion ist eine Funktion, die eine Zeichenfolge beliebiger Länge auf eine Zeichenfolge mit fester Länge, dem sogenannten Hashwert, abbildet. Zu einem identischen Text ist der Hashwert nämlich immer gleich.

```
SHA-256("CAS BIDA 2018") ==>
fc2b453db660aeb5112797142a67247a899b2e8d3d2f56c1db46dc71ce51e6ca
```

Wie in vorherigen Kapiteln erwähnt wurde, sind Hashwerte essentielle Bestandteil um Integrität und Validierung eines Blockes und der Kette zu gewährleisten. Es gibt kein De-facto Standard bei Kryptografische Hashfunktion, aber zurzeit wird im Blockchain Umfeld die SHA-256 Hashfunktion verwendet.

## Konsens Mechanismen

In einem dezentralen System wie der Blockchain gibt es nicht eine einzige Autorität, die für Recht und Ordnung sorgt, stattdessen vertraut man auf Konsens Mechanismen. Unter einem Konsens Mechanismus versteht man die Vorgehensweise, durch die eine Gruppe eine Entscheidung herbeiführt. Somit können sich fremde Teilnehmer in einem autoritären Netzwerk einigen.

Auf der Blockchain will man mit einem Konsens Mechanismus den double spending Effekt verhindern. Ein Benutzer soll nicht in der Lage sein, denselben Token mehrfach ausgeben zu können. Sonst wäre der Token respektive die Blockchain Währung einer Inflation unterlegen.

## Proof of Work

Der Proof of Work ist der bekannteste Konsens Mechanismus, welches vom Satoshi Nakamoto im Bitcoin Whitepaper erwähnt wurde. Hier eine kurze und knackige Beschreibung:

"" A piece of data which is difficult (costly, time-consuming) to produce but easy for others to verify and which satisfies certain requirements. Producing a proof of work can be a random process with a low probability so that a lot of trial and error is required on average before a valid proof of work is generated. ""

Wenn ein Benutzer eine Transaktion durchführt, muss das Netzwerk sicherstellen, dass der Benutzer der die Tokens transferieren will auch besitzt und das die Transaktion mit seiner digitalen Signatur unterzeichnet wurde. Der Prozess ist als Schlürfen (engl. Mining) bekannt.

Um einen neuen Block an Transaktionen erstellen zu können, müssen die Miner ein Hash

Puzzle lösen. Das Puzzle beinhaltet unter anderem den Merkle Tree, den Hash vom vorherigen Blockes und einen zufälligen Nonce. Aus diesen Daten wird ein Hash erzeugt welche mit der Lösungsvorgabe, dem Difficulty Level, entsprechen muss. Der berechnete Hash dient als Identifikationsnummer vom erzeugten Block, welche wiederum für die Berechnung des Hash-Puzzles des nächsten Blockes verwendet wird. [crr]

Um das Netzwerk aufrechtzuerhalten, bedient sich der Proof of Work Mechanismus zweier finanzieller Vergütungen: Der Erlös aus dem Mining und Transaktionskosten. Dem ersten Miner, der das Hash Puzzle löst, kann neue Tokens generieren und die Transaktionsgebühren beanspruchen.

## **Proof of Stake**

Im Gegensatz zum Proof of Work kommt der Proof of Stake ohne dem Zeit- und Energieintensiven Mining aus. Wie die folgende Proof of Stack Beschreibung besagt, ist die Wahrscheinlichkeit einer Entlohnung direkt proportional zu dem Anteil an Coins, die ein User besitzt, geteilt durch die Gesamtmenge an Coins welche im Umlauf sind [pos]:

"" In PoS-based public blockchains (e.g. Ethereum's upcoming Casper implementation), a set of validators take turns proposing and voting on the next block, and the weight of each validator's vote depends on the size of its deposit (i.e. stake). ""

Es existieren mehrere Variationen des Proof of Stake Prinzips, wie zum Beispiel Leased Proof of Stake und Delegated Proof of Stake. Beide Mechanismen führen zu ähnlichen Resultaten, doch nur „Proof-of-Work“ führt zu einer negativen Externalität. Die Blockchain Währungen mit der grössten Marktkapitalisierung beruhen auf den Proof of Work Konsens Mechanismus, wollen aber im Laufe dies und kommenden Jahres von Proof of Work zu Delegated Proof of Stake wechseln.

## **Smart Contract**

Den Begriff Smart Contract wurde vom Nick Szabo 1996 definiert, er beschreibt es als computerbasierendes Transaktionsprotokoll, dass die Bedingungen eines Vertrages beinhaltet [szabo]. In diesem Kontext sind Smart Contract ein Konzept das auf der Blockchain Technologie aufbaut [ew].

Im Wesentlichen lässt sich der Smart Contract in drei integrale Bestandteile unterteilen. Der erste Bestandteil besteht aus den Signaturen von N nutzende Smart Contract Parteien. Die ihr Einverständnis oder Ablehnung der Vertragsbedingungen mit ihren digitalen Signaturen erklären. Der nächste Bestandteil ist der Gegenstand der Vereinbarung. Dies etwas was nur im Umfeld des Smart Contracts existiert. Der letzte und dritte Bestandteil sind die Bedingungen. Dazu gehören die Anforderungen, Regeln, Belohnungen und Strafen, die mit diesen Vertrag verbunden sind.

Befürworter von Smart Contracts erhoffen sich von der Technologie eine Erleichterung von Geschäftsvorgängen und Vertragsabwicklungen sowie eine höhere Vertragssicherheit.



Grundsätzlich lässt sich jede Form von Kauf- oder Mietvertrag über ein Smart Contract auf der Blockchain abwickeln. In der Theorie ist es schneller, billiger und effizienter, weil die bürokratische Verwaltungsstruktur eingespart werden kann und Dritte, die bisher für die Sicherheit der Vertragspartner bürgten überflüssig werden würden.

## **Ethereum**

Ethereum gehört zur zweiten Generation der Blockchain Technologie und wurde vom Vitalik Buterin 2013 im Whitepaper "Ethereum: A Next Generation Smart Contract & Decentralized Application Platform" [ew] vorgestellt und 2014 im Yellow Paper von Gavin Wood ausgearbeitet. Im Juli 2015 startete der Betrieb von Ethereum.

Ethereum basiert auch auf der Blockchain Technologie, wobei es sich im Gegensatz zum Bitcoin nicht primär um eine Kryptowährung handelt. Es ist primär eine dezentralisierte Plattform für Distributed Apps (DApps), die auf der Blockchain Technologie basiert. Verteilte Nodes führen dabei Smart Contracts aus, die ihrerseits die Integrität der Güter und die Bezahlung in der Kryptowährung Ether sicherstellen. Der Token Ether ist ein Mittel zum Zweck.

Beim Abarbeiten der Blockchain Transaktionen, werden die Zustände der Ethereum Accounts verändert. Diese Zustände werden von allen Nodes berechnet und abgeglichen. Es gibt zwei Arten von Ethereum Accounts, die normalen Accounts und die Contract Accounts. Beide verwalten Ether Beträge, beim Contract Account kommt noch der Smart Contract Code hinzu.

Für Überweisungen von Beträgen wird eine entsprechende Message erstellt und signiert und als Block Transaction verschickt. Im Falle der Normalen Accounts bedeutet der Empfang und Versand von solchen Messages Überweisungen von Ether Beträgen. Wobei es auch komplexere Messages existieren. Wird von einem Normalen Account aus, eine Create Transaction Message erstellt und an einem weiteren normalen Account gesendet. Wird aus der Ziel Adresse ein Contract Account. Mit nachfolgenden Messages kann der Smart Contract Code aktiviert und getriggert werden.

Smart Contract können auf verschiedensten Weise entwickelt werden. Von Online Diensten bis zu lokalen Entwicklungsumgebungen ist alles vorhanden. Die Ethereum Smart Contracts werden in der Programmiersprache Solidity geschrieben und später mit Hilfe eines Compilers zu Byte Code kompiliert. Nach der Kompilierung werden die Smart Contract auf einer Ethereum Virtual Machine deployed indem es an eine Ethereum Contract Adresse versendet wird.

Dort werden sie dann auf einer Ethereum Virtual Machine ausgeführt, welches quasi eine Turing vollständige Maschine ist. Die Ausführung des Smart Contract ist durch das Gas begrenzt. Jede Funktion hat einen zugewiesenen Gas-Wert, welches den Ressourcenverbrauch widerspiegelt. Im Smart Contract kann auch eine Gas Limite festgelegt werden, die einem vor einem unerwarteten Hohen Verbrauch schützen soll.

# Lotterie

Die Lotterie ist ein Glücksspiel wo der Gewinner per Zufall entscheiden wird. Das bedeutet, dass Gewinn oder Verlust von einem Losverfahren abhängen. Es gibt verschiedene Arten von Lotterien, zu den bekanntesten gehören Tombola und Lotto. Mit dem Kauf von Losen besitzt man Teilnahmescheine an einem Losverfahren.

Veranstaltungen von Lotterie sind in der Schweiz der Comlot unterstellt. Sie unterscheiden zwischen Grosslotterien und Kleinlotterien, wobei Kleinlotterien weder automatisiert noch interkantonal noch online durchgeführt werden dürfen. Gemäss interkantonalem Recht dürfen Grosslotterien nur durch die Swisslos durchgeführt werden.

Seit Anfangs 2019 ist es ausländischen Anbietern nicht möglich Online-Glücksspiele in der Schweiz anzubieten. Die Regierung blockiert Webseiten aller internationalen Betreiber, die nicht zu den landbasierten Casinos gehören. Diese erhalten wiederum ein exklusives Anrecht auf eine Online-Lizenz.

"" In der Schweiz dürfen nur die Swisslos und die Loterie Romande im Internet Lotterien und Sportwetten durchführen. Alle anderen Angebote sind illegal. Die Comlot lässt deren Zugang zum Schweizer Markt durch die Internet Service Provider sperren. ""

Gemäss Art. 106 Abs. 6 BV müssen Reinerträge aus Lotterien müssen vollumfänglich gemeinnützigen Zwecken zukommen.

"" Die Kantone stellen sicher, dass die Reinerträge aus den Spielen gemäss Absatz 3 Buchstaben a und b vollumfänglich für gemeinnützige Zwecke, namentlich in den Bereichen Kultur, Soziales und Sport, verwendet werden. ""

# Proof of Concept

In den vorherigen Kapiteln wurden die Grundlagen vermittelt, um ein Grundverständnis für die Kern Themen dieser Semesterarbeit zu haben. Ziel ist es anhand eines Proof of Concept ein Lotterielos auf der Ethereum Platform zu entwickeln. Doch bevor mit der Umsetzung gestartet werden kann, müssen vorgängig zwei essentielle Aspekte vom Spiel durchleuchtet werden. Dies wäre zu einem der Zufallszahlengenerator und der Zeitbasierte Trigger. In den nachfolgenden Abschnitten werden die Schwierigkeiten und Problematik dieser Aspekte erläutert.

Danach wird der mögliche Lösungsraum festgelegt und anschliessend eine praktische Umsetzung vorgestellt.

## Zufallszahlen

Zufallszahlen haben in der Informatik einen wichtigen Stellenwert und sind in verschiedenen Anwendungen wieder zu finden. Zum Beispiel in Simulationen, Kryptographie oder probabilistische Algorithmen. Doch was sind Zufallszahlen, der österreichische Wissenschaftler Philipp Frank hat im Jahre 1932 folgende Erklärung dafür [zufallszahlen].

"" Ein Zufall schlechthin, also gewissermaßen ein absoluter Zufall wäre dann ein Ereignis, das in Bezug auf alle Kausalgesetze ein Zufall ist, das also nirgends als Glied einer Kette auftritt. ""

Zur Erzeugung von Zufallszahlen gibt es verschiedene Verfahren, welche als Zufallszahlengeneratoren bezeichnet werden. Ein entscheidendes Kriterium für Zufallszahlen ist, ob das Ergebnis der Generierung als unabhängig von früheren Ergebnissen betrachtet werden kann oder nicht. Grundsätzlich unterscheidet man zwischen nicht-deterministischen und deterministischen Zufallszahlengeneratoren. Die nicht-deterministischen erzeugen sogenannte Echte Zufallszahlen und weisen eine optimale statistische Eigenschaft und haben keine Reproduzierbarkeit. Eingangs Quellen von solchen Zufallszahlen kommen in der Regel in der Natur vor. Einer der prominenteste ist die Rauschgrössenmessung. Ein deterministischer Zufallszahlengenerator liefert bei gleichen Ausgangsbedingungen dagegen immer die gleiche Folge von Zufallszahlen sogenannte Pseudozufallszahlen.

Software orientierte Lösungen sind grundsätzlich deterministisch. Damit ein nicht-deterministischer Zufallszahlengenerator realisiert werden kann, muss zwingend eine externe Quelle einbezogen werden. Hierzu können zum Beispiel Impulsschwankungen elektrischer Schaltungen genutzt werden.

In der Praxis werden häufig arithmetische Zufallszahlengeneratoren verwendet, welche eine Mischform darstellen. Sie liefern zwar Pseudozufallszahlen, nutzen aber als Startwert eine echte zufällige Zahl. Dies kann zum Beispiel die aktuelle Systemzeit oder Hardware-Ereignisse sein.

Weil arithmetische Zufallszahlengeneratoren keine statistischen Auffälligkeiten vorweisen, werden Sie in Software Prozeduren oft verwendet.

## **Softwaretechnische Realisierung**

Grundsätzlich existieren verschiedene Implementierungsmöglichkeiten um ein arithmetischer Zufallszahlengenerator abzubilden. Wobei diese sich in zwei Gruppen unterteilen lassen [hzz]:

### *Arithmetische Zufallszahlengenerator*

"" Dieser basiert auf der Arithmetik. Irrationale Zahlen wie Wurzel oder e können als Zufallszahlengenerator verwendet werden, indem man den gebrochenen Teil beliebiger Vielfache als Zufallszahlen nutzt. Nachteil des Verfahrens ist, dass sich irrationale Zahlen nur als Näherungswerte innerhalb der Rechengenauigkeit darstellen lassen. ""

### *Rekursiver arithmetischer Zufallszahlengenerator*

"" Beruht auf der Berechnung einer neuen Zufallszahl aus einer oder mehreren vorhergehenden Zahlen. Die neu erzeugte Zahl wird gespeichert und geht bei erneutem Aufruf des Zufallszahlengenerators in die Berechnung ein. Beim allerersten Aufruf des Zufallszahlengenerators muss ein willkürlich gewählter Startwert, respektive Seed verwendet werden. ""

## **Einschränkungen**

Smart Contracts auf der Ethereum Plattform sind deterministisch und laufen unabhängig voneinander auf verschiedenen Blockchain Nodes respektive Ethereum Virtual Machine (EVM). Wegen dem deterministischen Charakter eines Smart Contracts und Schwierigkeit eine nicht-deterministische Zufallszahl auf verteilten unabhängigen Nodes zu erzeugen. Sucht man in der Programmiersprache Solidity vergeblich nach einer Zufallszahl Funktion. Ausserdem ist es per se nicht möglich eine externe Quelle ausserhalb der Blockchain anzusprechen. Das bedeutet es ist nicht möglich ein HTTP Request auf eine Internet Ressource abzusetzen.

## Zeitbasierender Trigger

In Protokollen oder Log Dateien sind Zeitstempel wiederzufinden, sie dienen in der Regel als Meta Information. Doch in einem verteilten Netzwerk, dienen diese Daten unter anderem um chronologische Abläufe zu erkennen. Damit dies möglich ist, müssen alle Teilnehmer eines Computer Netzwerk die gleiche System Zeit verwenden. Weil aber die System Zeit bei einem Computer im Verlauf der Zeit eine Abweichung aufweist. Sollten Computer in einem Kommunikationsnetzwerk ihre Systemzeit kontinuierlich justieren. Aus diesem Grund gleichen Computer ihre Systemzeit mit einem Network Time Protokoll Service aus dem Internet ab.

Erstellen von Ethereum Blocks sind nicht abhängig von einem zeitlichen Intervall und das Ethereum Protokoll beschreibt nicht, wie sich Nodes respektive Miners ihre Zeit abgleichen. Miner müssen beim erstellen eines Blockes einen Zeitstempel setzen. Im Ethereum Whitepaper findet man für den Zeitstempel nur folgendes Kriterium [ew]

*"" Check that the timestamp of the block is greater than that of the previous block and less than 2 hours into the future ""*

— Vitalik Buterin, Ethereum Whitepaper

Aktionen die einen Smart Contract Zustand verändern, benötigen eine Blockchain Transaktion und werden in einem Blockchain Block abgebildet. Weil die Blockchain per se Single Thread ist und Transaktionen nacheinander abgearbeitet werden, bedeutet, dass die Blockchain Block Generierung die Intervalle der möglichen Smart Contract Aktivitäten bestimmt.

Smart Contracts basierend auf der Ethereum Plattform werden von externen Event getriggert. Was wiederum bedeutet, dass Smart Contract nicht ein Computer Prozess ist, welches von selbst aus aufwacht und eine Zustandsveränderung respektive Transaktion auslöst. Es agiert nur wenn ein Akteur ein Smart Contract Funktion aufruft. Aus diesem Grund findet man auf der Ethereum Plattform keine Funktionalität die einen Zeitintervall abbildet.

## Schlussfolgerung

Grundsätzlich ist ein Smart Contract deterministisch und der dahinterliegende Code auf der Blockchain für jedermann einsehbar. Zusätzlich zu dem, sind auch die Smart Contract Zustände ablesbar. Damit ein Lotterie Teilnehmer nicht seine Chancen im Voraus berechnen kann, muss der Zufallszahlgenerator einen Input Wert verwenden welches in der Zukunft liegt und nicht erraten werden kann oder Werte müssen verschlüsselt auf der Blockchain liegen.

Smart Contract sind keine ewig laufende Prozesse, sie werden von extern getriggert. Aus diesem Grund muss ein Akteur die effektive Ziehung explizit oder implizit triggern. Wichtig zu verstehen ist, dass der Zeitintervall vom Block Generierung respektive von den Miners definiert werden. Ausserdem sind die Zeit Intervalle in der Regel nie gleich lang.

Zeitorientierte Meta Daten aus der Blockchain sind keine garantierte zuverlässige Input Quellen für den Zufallsgenerator. Denn gemäss Ethereum gibt es keine klare Definition wie die Nodes ihre Systemzeit synchronisieren sollten.

Jetzt stellt sich die Frage was für Input Quellen und Verfahren könnte man verwenden um einen Zufallszahlengenerator zu entwickeln. In dieser Semesterarbeit wurden einige Ideen gesammelt, aber weil es den Rahmen dieser Arbeit sprengen würde, kann nicht jede Lösung in Detail eingegangen werden.

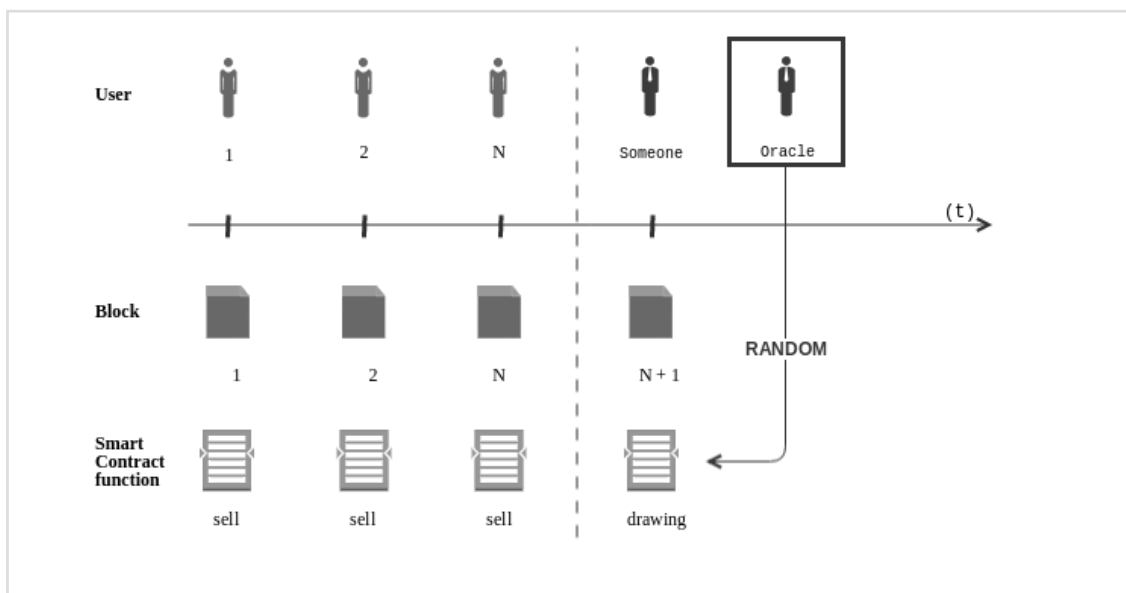
## Lösungsraum

Es wurden drei mögliche Lösungsvarianten eruiert, wobei dies nicht unter einer detaillierten Risiko Analyse unterzogen wurden.

### L1 - Externer Oracle und externem Zeit Trigger

Weil es offensichtlich nicht trivial ist eine sichere Zufallszahl auf der Ethereum Plattform zu erzeugen und eine Applikation bei einem bestimmten Zeitpunkt zu triggern. Haben einige Entwickler auf die Idee einen kostenpflichtigen Service für diese Problemstellung zu entwickeln. Der prominenteste unter den Zufallszahl Service Provider ist Provable. Dieser ermöglicht einem Quellen von Zufallszahlen ausserhalb der Blockchain zu beziehen. Dies bedingt aber dass ein Smart Contract auf Basis der Provable erstellt wird, welches vom Lotterielos Smart Contract getriggert wird.

Damit man Aktionen triggern kann, welche in der Zukunft liegen, wie zum Beispiel die Lotterielos Ziehung, könnte man die Alarm Clock Service verwenden. Dieser Triggert den Smart Contract anhand der Blockchain Block Zähler.



Aus Usability Sicht hat die Lösung den Vorteil dass Lotterielos Teilnehmer ein Ticket kaufen können und dass die Ziehung an einen vordefinierten Zeitpunkt stattfindet. Dies entspricht dem

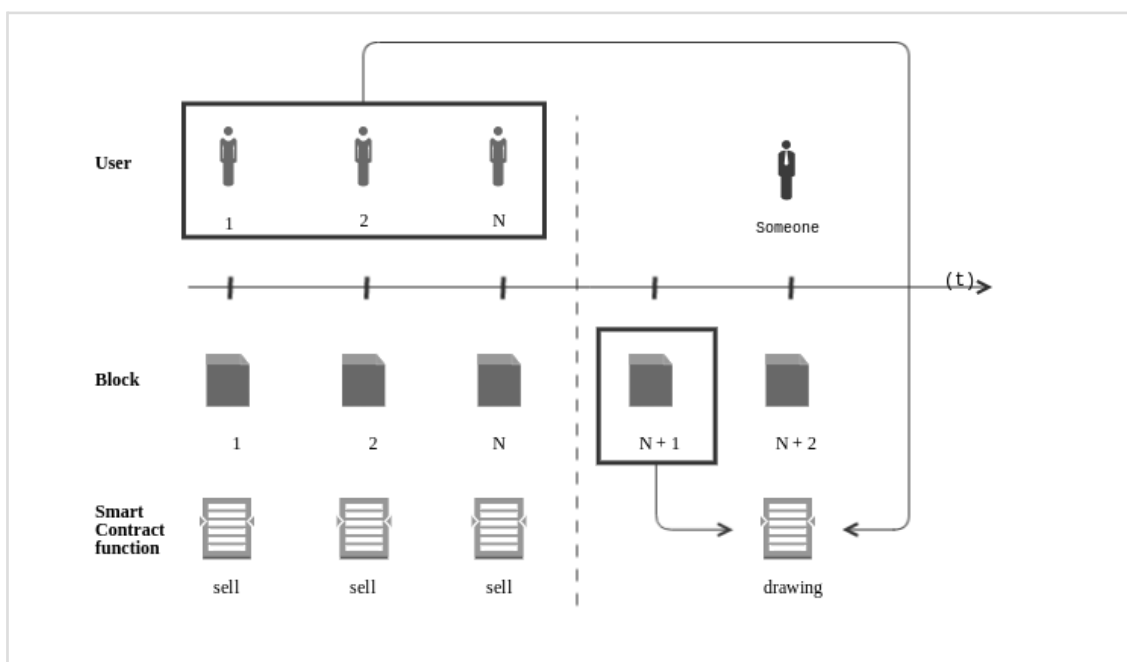
wie aktuellen Lotterielos Verfahren funktionieren.

Der grosse Nachteil liegt auf der Hand. Was mit dem Smart Contract unter anderem bezwecken will, ist die Anzahl der Vertragspartner auf ein Minimum zu reduzieren und Dritte ausklammert. Ausserdem übergibt man die Kontrolle vom Lotterielos an den externen Service.

Aus Sicht eines Angreifers ist dieses Konzept sehr interessant, er muss nur noch den externen Oracle Service angreifen um N Smart Contracts zu blockieren. Wenn die Smart Contracts keine Redundanz eingebaut haben. Ist ein System Ausfall auf seiten des Oracle Service Anbieter mit einem Spiel Unterbruch oder sogar Abbruch gleichzusetzen.

## L2 - User Adressen und Block Daten

Ein andere Variante wäre auf externe Service zu verzichten und die Zufallszahl anhand von Teilnehmern Adressen und zukünftigen Blockchain Block Meta Daten zu erzeugen. Das bedeutet das bis zu Ende der Lotterielos Teilnahme, Teilnehmer ihr Lotterielos kaufen und danach auf den nächsten Blockchain Block Generierung warten und dessen Block Hash und Difficulty mit dem Teilnehmer Adressen als Input Quellen für den Zufallszahlengenerator verwendet.

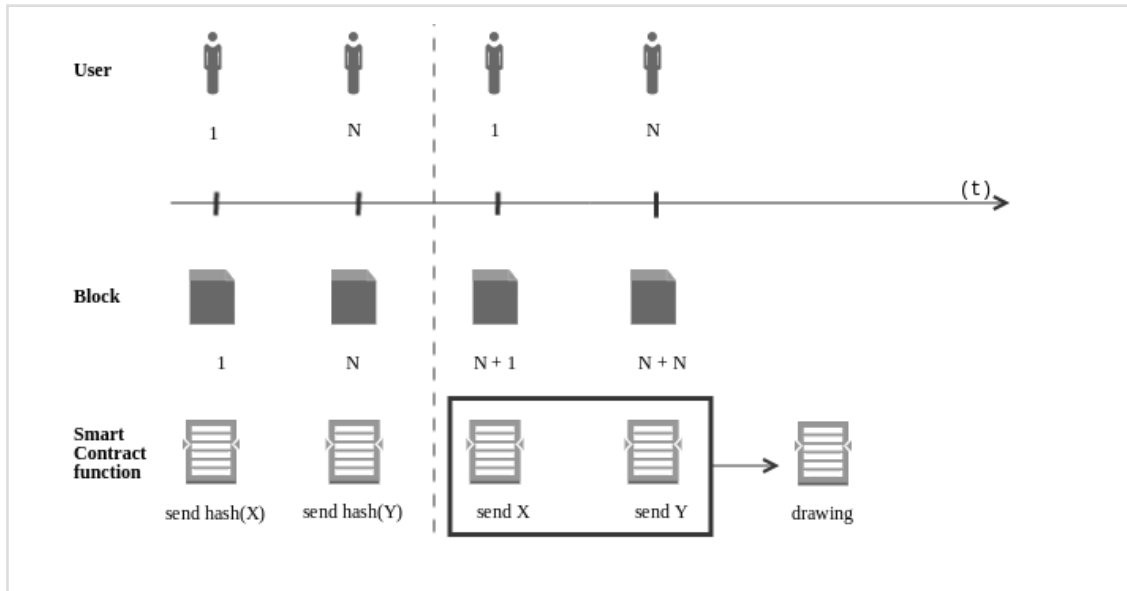


Interessant an dieser Lösung ist das dies ohne fremde Quellen auskommt und die Nachvollziehbarkeit gewährleistet wird. Jedoch ist es theoretisch möglich das der Miner welche die Blocks erstellt diese Verlosung manipulieren kann. Denn er kann den Block Hash beeinflussen.

## L3 - Commitment-Verfahren

In dieser Variante gibt es aus Sicht vom Teilnehmer zwei Phasen. In der ersten Phase erstellt

der Teilnehmer eine Zufallszahl und berechnet den Hash Wert davon. Dieser Wert wird dem Smart Contract übermittelt. In der zweiten Phase, wenn die Teilnahme am Lotterielos Verfahren zu Ende ist. Senden die Teilnehmer ihre erzeugte Zufallszahl am Smart Contract, dieser erzeugt den Hashwert davon und überprüft diese mit übermittelten Hashwert aus der ersten Phase. Wenn diese Stimmen werden anhand die Zufallszahl XOR verknüpft und dient als Zufallszahl fürs festlegen des Gewinners.



Interessante an dieser Variante ist die Selbstbestimmung und Nachvollziehbarkeit der Zufallszahl. Die Smart Contract Akteure beschränkt sich auf die Lotterielos Teilnehmer. Es werden auch keine externe Blockchain Block Daten bezogen welche theoretisch von Blockchain Miners beeinflusst werden könnten. Jedoch liegt die Knacknuss in der zweiten Phase, ein Teilnehmer könnte die zweite Phase blockieren, indem es nicht teilnehmen würde. Somit wäre das Spiel einer Denial of Service unterstellt.

Man könnte einen Finanziellen Anreiz erschaffen, so das Benutzer motiviert wären and der Zweite Phase teilzunehmen und/oder bei einer bestimmten Anzahl Block Generierung die eingereichten Zufallszahlen verwenden. Hier stellt sich die Frage, ob das aus rechtlichen Sichten verhebt. Dies müsste man in einer nachfolgende Arbeit in Erfahrung bringen.

## Entscheid

Grundsätzlich müsste man anhand eines Bewertungskatalog und einem Scoring Model die Lösungsvarianten bewerten. Weil dies aber den Rahmen dieser Semesterarbeit sprengen würde. Beruht die folgende Argumentation auf die subjektive Sicht des Autors.

Weil die Lösungsvariante L1 nicht den Smart Contract Philosophien entsprechen und die Kontrolle vom Spiel dem externen Oracle übergeben wird. Stellt sich die Frage, für was man ein Blockchain basierter Smart Contract benötigt. Wäre es nicht einfacher und sinnvoller auf eine konventionelle Applikation Architektur zu setzen. Würde man diese Variante trotzdem

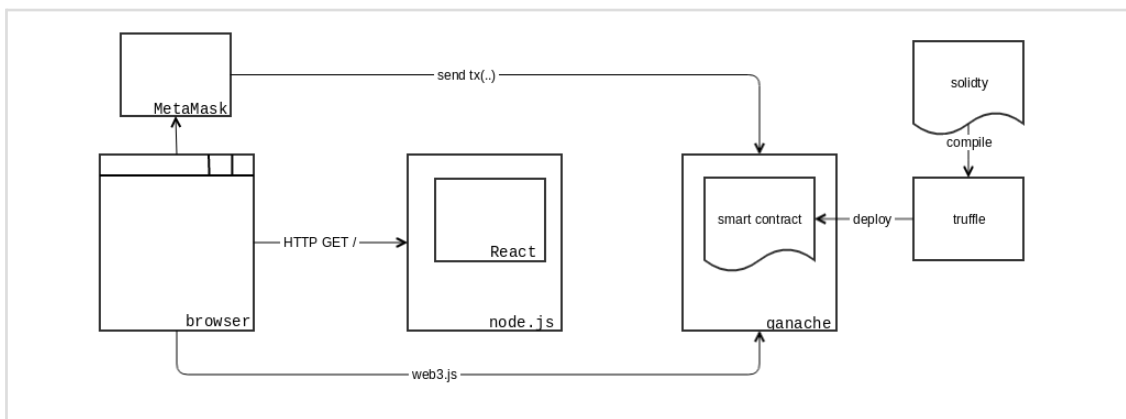


verfolgen, stellt sich die Frage wie mit der Single Point of Failure Problematik umgegangen wird. Die logische Schlussfolgerung wäre einbeziehen von N verschiedenen externen Oracles, anderweitig würde der Ausfall eines Oracle das Lotterielos Spiel blockieren.

Die Lösungsvarianten L2 und L3 haben ihre Vor- und Nachteile. Aus Sicht von Usability punktet die L3 weil es weniger User Interaktion benötigt. Aus Sicht der sicheren Zufälligkeit punktet die L3 Variante. Weil aber die L3 auf einfacher Weise einer DOS Attacken unterstellt ist, benötigt ihr ein Konzept. Aus Sicht vom Autor ist die sichere Zufälligkeit mit der Lösungsvariante L2 gegeben. Aus diesem Grund wurde die Variante L2 in einem Prototyp umgesetzt.

## Prototyp

Für die Umsetzung der Lösungsvariante L2 wurde eine Web Applikation basierend auf React einer Single Page Architektur aufgebaut, welches mit Hilfe der JavaScript web3.js Bibliothek an die Blockchain kommuniziert. Während der Entwicklung wurde mit Hilfe der Truffle Suite die Ethereum Blockchain simuliert. Das folgende Architektur Diagram zeigt die Abhängigkeiten der Komponenten auf. Der Source Code vom PoC wurde auf GitHub gestellt: <https://github.com/0r1t0m/cas-bida-term-paper>



## Zufallszahl Generator

Die Zufallszahl basiert auf den Hashwert aller Adressen und dem zukünftigen generierten Blockchain Block Hashwert.

```
function random() private view returns (uint8) {
    bytes memory addresses = abi.encodePacked(buyers);
    bytes32 blockHash = blockhash(block.number - 1);
    uint difficulty = block.difficulty;
    bytes memory value = encode(blockHash, difficulty, addresses);
    uint ticketCount = tickets.length;
    return uint8(uint256(keccak256(bytes(value))) % ticketCount);
}
```

## Production-Ready?

Grundsätzlich kann man diese Frage mit Nein beantworten. Denn der PoC dürfte nach schweizer Recht nicht Produktiv aufgeschaltet werden, siehe Kapitel "Lotterie". Aber es wäre denkbar dies unter dem Mantel von Swisslos durchzuführen. Dies bedingt das Swisslos an einer solchen Partnerschaft einwilligt.

Abgesehen davon müsste man den Smart Contract Code einer statischen Code Analyse und Security Audit unterziehen. Es bräuchte auch ein Operativen Konzept im Fall eines Hot Fixes. Wahrscheinlich wäre es sinnvoll, nach jeder Durchführung den Smart Contract zu destroyen und wieder zu deployen, so wäre man aus operativ Sicht vorbereitet, eine revidierte Version vom Smart Contract zu deployen.

# Zielerreichung

Die im Kapitel “Aufgabenstellung” definierten Ziele wurden grundsätzlich umgesetzt. Es wurden mehrer Lösungsvarianten ausgearbeitet, wobei die am Anfang angedachte Variante, welche externen Oracle einbezieht, als schlechtester Abschnitt. Aus diesem Grund wurde der optionaler Punkt ZO1 bewusst nicht umgesetzt. In einer nachfolgende Arbeit, müssten die Lösungsvarianten in Detail ausgearbeitet werden und einer Risiko Bewertungs Analyse unterzogen werden.

# Erkenntnisse

An dieser Stelle wird nochmals über allgemeine Erkenntnisse reflektiert, welche dem Autor während dieser Arbeit aufgefallen sind.

- In Solidity können eigene Typen mit Hilfe von struct erstellt werden. Will man diesen spezifischen Typ nach aussen propagieren. Ist man gezwungen die interne Struktur wieder in Solidity Default Type herunterzubrechen.
- Verglichen zu anderen Entwicklungsumgebung befindet sich Ethereum Smart Contract Umgebung noch im Anfangs Zustand.
- Die Integration mit der JavaScript Bibliothek web3.js und einer Single Page Application Javascript Framework ist harzig und fehleranfällig.
- Mit der Solidity Funktion blockhash(..) kann man nur auf die letzten 256 Blockchain Blöcke zugreifen, was bedeutet, dass in der Smart Contract Logik man darauf vorbereitet sein muss, nicht auf den gewünschten Block zuzugreifen zu können.
- Zustände eines Smart Contract sind für jedermann ablesbar. Im Fall von sensiblen Daten wäre eine Public-Key-Verschlüsselungsverfahren wünschenswert. Weder die Programmiersprache Solidity noch die Ethereum Smart Contract Adresse bietet diese Möglichkeit an.

# Quellenverzeichnis

- [bw] Satoshi Nakamoto [Oktober 2008]. Bitcoin: A Peer-to-Peer Electronic Cash System [Online]  
Available: <https://bitcoin.org/bitcoin.pdf>
- [szabo] Szabo [1997]. Formalizing and Securing Relationships on Public Networks. Internet economics and Security.
- [proof-of-work] Demelza Hays [2018]. Consensus Mechanisms [Online]  
Available: <https://cryptoresearch.report/crypto-research/consensus-mechanisms/>
- [pos] James Ray [2018]. Proof of Stake FAQs [Online]  
Available: <https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs#what-is-proof-of-stake>
- [zufallszahlen] Philipp Frank [1932]. Zufallzahl
- [sc] Smart contracts [Online]  
Available: <https://en.bitcoin.it/wiki/Contracts>
- [ew] Vitalik Buterin [2013]. Ethereum White Paper [Online]  
Available: <https://github.com/ethereum/wiki/wiki/White-Paper>
- [hzz] Prof. Dr.-Ing. Norbert Reifschneider [2000]. Hardwareanalysator für Zufallszahlen [Online]  
Available: <https://www.hs-heilbronn.de/1837998/hardwareanalysator-fur-zufallszahlen>
- [crr] Demelza Kelso Hays, Mark J. Valek [2018]. Crypto Research Report [Online]  
Available: <https://cryptoresearch.report/wp-content/uploads/2018/06/Crypto-Research-Report-Juni-2018-3.pdf>
- Tobias Litte [2005], Zufallsgeneratoren [Online]  
Available: <https://perso.u-cergy.fr/~ahonecker/bs/semstat/ausarb/litte.pdf>
- Der Bundesrat [Dezember 2018] Rechtliche Grundlagen für Distributed Ledger-Technologie und Blockchain in der Schweiz [Online]  
Available: <https://www.newsd.admin.ch/newsd/message/attachments/55150.pdf>
- Andreas Rabe [2016]. Wie funktioniert ein Zufallsgenerator? [Online]  
Available: <https://addis-techblog.de/2016/01/wie-funktioniert-ein-zufallsgenerator>
- Prof. Dr.-Ing. Norbert Reifschneider [2000]. Softwareanalysator für Zufallszahlen [Online]  
Available: <https://www.hs-heilbronn.de/3175988/softwareanalysator-fuer-zufallszahlen>
- Austin Thomas Griffith [Dezember 2018]. Commit Reveal Scheme on Ethereum [Online]  
Available: <https://medium.com/gitcoin/commit-reveal-scheme-on-ethereum-25d1d1a25428>

- Krishnendu Chatterjee, Amir Kafshdar Goharshady, Arash Pourdamghani [Februar 2019]. Probabilistic Smart Contracts: Secure Randomness on the Blockchain Available: <https://arxiv.org/pdf/1902.07986.pdf>
- Anand Paul Mookan [Mai 2017]. Die Blockchain Technologie: Eine Methode zur Identifikation von Anwendungsfällen Available: [https://digitalcollection.zhaw.ch/bitstream/11475/7678/1/Mookan\\_Anand\\_W.MA.WIN.pdf](https://digitalcollection.zhaw.ch/bitstream/11475/7678/1/Mookan_Anand_W.MA.WIN.pdf)

# Appendix A: Smart Contract

```
pragma solidity >=0.5.0 <0.6.0;

contract Lottery {

    enum State {
        Initialize,
        Running,
        Completed,
        Finished
    }

    struct Ticket {
        address payable buyer;
    }

    event Sold();

    event Draws();

    event Reset();

    event Completed();

    address owner;

    State currentState = State.Initialize;

    Ticket[] tickets;

    uint8 drawingNumber;

    address payable [] buyers;

    uint price;

    uint minimumSoldTickets;

    uint latestBlockNumber;

    constructor () public {
        owner = msg.sender;
        currentState = State.Running;
        price = 1000000000000000;
        minimumSoldTickets = 5;
        latestBlockNumber = 0;
    }

    modifier isOwner() {
        require(owner == msg.sender, "Sender not authorized.");
    }
```

```

    -;
}

modifier isReadyToDrawing() {
    require(currentState == State.Completed, "Not enough ticket
sold.");
    require(block.number >= latestBlockNumber, "Wait block
generation.");
    -;
}

modifier isPayable() {
    require(price == msg.value, "Not exactly required costs.");
    require(currentState == State.Running, "All tickets sold");
    -;
}

modifier isRunning() {
    require(currentState == State.Running || currentState ==
State.Completed, "Game is not running.");
    -;
}

function getCurrentState() public view returns (State) {
    return currentState;
}

function buy() public isPayable isRunning payable {
    tickets.push(Ticket({buyer : msg.sender}));
    buyers.push(msg.sender);
    emit Sold();

    if(tickets.length >= minimumSoldTickets) {
        currentState = State.Completed;
        latestBlockNumber = block.number + 2;
        emit Completed();
    }
}

function showTickets() public view returns (address payable []
memory) {
    return buyers;
}

function ticketSoldCount() external view returns (uint) {
    return tickets.length;
}

function drawingTicket() external view returns (uint8) {
    return drawingNumber;
}

function drawing() external isRunning isReadyToDrawing isOwner

```



```

{
    currentState = State.Finished;
    drawingNumber = random();
    Ticket memory ticket = tickets[drawingNumber];
    withdraw(ticket.buyer);
    emit Draws();
}

function reset() external isOwner {
    currentState = State.Initialize;
    for(uint i = 0; i<buyers.length; i++) {
        delete buyers[i];
        buyers.length--;
    }

    buyers.length = 0;
    for(uint i = 0; i<tickets.length; i++) {
        delete tickets[i];
        tickets.length--;
    }
    tickets.length = 0;

    delete drawingNumber;
    currentState = State.Running;
    emit Reset();
}

function random() private view returns (uint8) {
    bytes memory buyerAddress = abi.encodePacked(buyers);
    bytes32 blockHash = blockhash(block.number - 1);
    uint difficulty = block.difficulty;
    bytes memory value = encode(blockHash, difficulty,
buyerAddress);
    uint ticketCount = tickets.length;
    return uint8(uint256(keccak256(bytes(value))) %
ticketCount);
}

function encode(bytes32 blockHash, uint difficulty, bytes
memory buyerAddress) private pure returns (bytes memory) {
    return abi.encodePacked(blockHash, difficulty,
buyerAddress);
}

function withdraw(address payable _winnerAddress) private {
    uint256 balanceToDistribute = calculateEarnings();
    _winnerAddress.transfer(balanceToDistribute);
}

function calculateEarnings() private view returns (uint256) {
    uint256 balance = address(this).balance;
    uint256 amount = (price * tickets.length) * 4 / 5;
    if (amount >= balance) {

```

```
        return balance / 2;
    }
    else {
        return amount;
    }
}
}
```

## Appendix B: React JS Web Entry Point

```
import 'bootstrap/dist/css/bootstrap.min.css';
import React, {Component} from 'react';
import './style.css';

import Shop from '../Shop';
import MetaData from '../MetaData';
import Tickets from '../Tickets';
import getWeb3 from "../../utils/getWeb3";
import Lottery from "../../Lottery.json";

class App extends Component {

  state = {
    smart_contract_address: null,
    balance: null,
    curr_block: null,
    game_state: null,
    accounts: [],
    tickets: [],
    contract: null,
    web3: null,
    winner: null
  }

  componentWillMount = async () => {

    const web3 = await getWeb3();
    const networkId = await web3.eth.net.getId();
    const network = Lottery.networks[networkId];
    const contract = new web3.eth.Contract(
      Lottery.abi,
      network && network.address,
    );

    let accounts = await web3.eth.getAccounts();
    let soldTickets = await
contract.methods.showTickets().call();
    let gameState = await
contract.methods.getCurrentState().call();
    let blockNumber = await web3.eth.getBlockNumber();
    let smartContractAddress = await contract.options.address;
    let balance = await
web3.eth.getBalance(smartContractAddress);
    //let winner = await
contract.methods.getDrawingTicket().call();

    this.setState({
      accounts: accounts,
      contract: contract,
```

```

        web3: web3,
        tickets: soldTickets,
        game_state: gameState,
        curr_block: blockNumber,
        smart_contract_address: smartContractAddress,
        balance: balance,
        //winner: winner
    });

    this.watchTicketEvent();
    this.watchWinnerEvent();
    this.watchResetEvent();
    this.watchSoldOutEvent();
}

watchTicketEvent() {
    const {contract} = this.state;
    contract.events.Sold({}, (error, data) => {
        if (error) {
            console.log("Error: " + error);
        }
        else {
            this.updateSoldTickets();
            this.updateMetaData();
        }
    });
}

watchSoldOutEvent() {
    const {contract} = this.state;
    contract.events.Completed({}, (error, data) => {
        if (error) {
            console.log("Error: " + error);
        }
        else {
            this.updateMetaData();
        }
    });
}

watchWinnerEvent() {
    const {contract} = this.state;
    contract.events.Draws({}, (error, data) => {
        if (error) {
            console.log("Error: " + error);
        }
        else {
            this.updateSoldTickets();
            this.updateMetaData();
            this.updateWinner();
        }
    });
}

```

```

watchResetEvent() {
  const {contract} = this.state;
  contract.events.Reset({}, (error, data) => {
    if (error) {
      console.log("Error: " + error);
    }
    else {
      console.log("catch reset event.");
      this.resetGame();
      this.updateMetaData();
    }
  });
}

updateSoldTickets = async () => {
  const {contract} = this.state;
  let soldTickets = await
contract.methods.showTickets().call();
  this.setState({
    tickets: soldTickets
  });
}

updateMetaData = async () => {
  const {contract, web3} = this.state;
  let balance = await
web3.eth.getBalance(contract.options.address);
  let blockNumber = await web3.eth.getBlockNumber();
  let gameState = await
contract.methods.getCurrentState().call();
  this.setState({
    game_state: gameState,
    blockNumber: blockNumber,
    balance: balance
  });
}

updateWinner = async () => {
  const {contract} = this.state;
  let winner = await contract.methods.drawingTicket().call();
  this.setState({
    winner: winner
  });
}

resetGame = async () => {
  window.location.reload();
}

render() {
  const {smart_contract_address, balance, curr_block,
game_state, accounts, tickets, contract, winner} = this.state;

```

```

    return (
      <div className="game container fixed-top">
        <div className="app-header row"/>
        <div className="app-shop row">
          <Shop accounts={accounts} contract={contract}/>
        </div>
        <div className="app-meta-data row">
          <MetaData gameState={game_state}
            blockNumber={curr_block}

smartContractAddress={smart_contract_address}
            balance={balance}/>
        </div>

        <div className="app-tickets row">
          <Tickets tickets={tickets} winner={winner}/>
        </div>
      </div>
    );
  }
}

export default App;

```

## Appendix C: web3.js Provider

```
import Web3 from "web3";

const getWeb3 = () =>
  new Promise((resolve, reject) => {
    // Wait for loading completion to avoid race conditions
    with web3 injection timing.
    window.addEventListener("load", async () => {
      // Modern dapp browsers...
      if (window.ethereum) {
        const web3 = new Web3(window.ethereum);
        try {
          // Request account access if needed
          await window.ethereum.enable();
          // Accounts now exposed
          resolve(web3);
        } catch (error) {
          reject(error);
        }
      }
      // Legacy dapp browsers...
      else if (window.web3) {
        // Use Mist/MetaMask's provider.
        const web3 = window.web3;
        console.log("Injected web3 detected.");
        resolve(web3);
      }
      // Fallback to localhost; use dev console port by
      default...
      else {
        const provider = new Web3.providers.HttpProvider(
          "http://127.0.0.1:7545"
        );
        const web3 = new Web3(provider);
        resolve(web3);
      }
    });
  });

export default getWeb3;
```

# Selbständigkeitserklärung

Mit der Abgabe dieser Arbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat (Bei Teamarbeiten gelten die Leistungen der übrigen Teammitglieder nicht als fremde Hilfe).

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die vorliegende Arbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Ort, Datum .....

Unterschrift Studierender .....