



Institut für Informatik

Lehrstuhl für Organic Computing

Prof. Dr. rer. nat. Jörg Hähner

Masterarbeit

**Learning drinking patterns with
Q-Learning and Feature Selection on an
Ethereum Blockchain**

Fabian Pieringer

Erstprüfer: Prof. Dr. rer. nat. Jörg Hähner

Zweitprüfer: Prof. Dr. Elisabeth André

Betreuer: Wenzel Pilar von Pilchau, M.Sc.

Matrikelnummer: 150886

Studiengang: Informatik (Master)

Eingereicht am: 2. Juni 2019

Abstract

In dieser Masterarbeit wird ein System vorgestellt, welches die Blockchain Technologie und Maschinelles Lernen in Einklang bringt. Das Ziel der Arbeit ist es zu erforschen, zu welchem Grad sich diese neuartigen Technologien bereits verbinden lassen.

Zur Abbildung dieser Forschungsfrage wurde eine Problemstellung formuliert, welche das Erlernen des Kaffeetrinkverhaltens eines Users anhand von Daten einer Blockchain beinhaltet.

Dazu wurde eine Systemarchitektur entworfen, in welcher Getränkedaten von einer Mobile-App (Tablet) auf eine private Blockchain geschrieben werden und anhand dessen eine Lerner-Instanz, unter Beobachtung der Blockchain, das Kaffeetrinkverhalten der einzelnen Nutzer erlernen kann.

Zur Beantwortung der Forschungsfrage wurde eine Studie am Lehrstuhl durchgeführt, bei welcher eruiert wurde inwiefern das Erlernen des Kaffeetrinkverhaltens durch Daten von der Blockchain möglich ist. Aufgrund veralteter Hardware und den daraus resultierenden Problemen und Einschränkungen, reduzierte sich die Laufzeit der Testphase von 2 Monaten auf 3,5 Wochen.

Als Grundlage für die Analyse und Auswertung der Ergebnisse, wurde eine Anwendung zur Simulation eines Users entwickelt. Mit dieser Anwendung erfolgte der Nachweis der Funktionalität des implementierten Reinforcement Learning

Abstract

Algorithmus. Zudem dienten die Ergebnisse der simulierten Durchläufe als Anhaltspunkt und Bewertungsgrundlage für die Resultate der Studie.

Schlussendlich konnten Similaritäten in den Ergebnissätzen der Studie und der Simulation aufgezeigt werden und die Forschungsfrage anhand von Tendenzen positiv beantwortet werden. Da die Dauer der Studie einer der limitierenden Umstände der Arbeit war, wurde zur allumfassenden Erörterung der Forschungsfrage, auf die Notwendigkeit einer längeren Studie, ohne die technisch beeinträchtigenden Faktoren, hingewiesen.

Inhaltsverzeichnis

Abstract	i
1 Einleitung	1
1.1 Aufbau dieser Arbeit	1
1.2 Problemstellung	2
1.3 Related Work	4
2 Stand der Technik	7
2.1 Blockchain	7
2.1.1 Funktionsweise & Konzepte	7
2.1.2 Ethereum	11
2.2 Machine Learning	14
2.2.1 Überblick	14
2.2.2 Reinforcement-Learning	16
3 System Architektur	21
3.1 Überblick	21
3.1.1 Architektur	21
3.1.2 Workflow	27
3.2 Blockchain	34
3.2.1 Genesis Block	34

Inhaltsverzeichnis

3.2.2	CoffeeCoin	37
3.2.3	Beverage-List	39
3.3	Learner	41
3.3.1	Modellierung	41
3.3.2	Q-Learning	43
3.3.3	Lernprozess & Ablauf	47
3.4	Tablet-App	50
3.4.1	Interface	50
3.4.2	Internal Workflow	55
4	Studie	59
4.1	Simulation	59
4.2	Testphase	61
4.3	Evaluation	63
5	Zusammenfassung	69
5.1	Fazit	69
5.2	Ausblick & Weiterführende Forschungsfragen	71
	Literaturverzeichnis	I
	Abbildungsverzeichnis	V
	Tabellenverzeichnis	VII
A	Anhang A	IX
B	Anhang B	XI

1 Einleitung

1.1 Aufbau dieser Arbeit

Diese Masterthesis ist in fünf Teile gegliedert. Im ersten Teil wird zuerst ein Überblick zur Motivation und Problemstellung gegeben. Anschließend werden Arbeiten präsentiert, welche in der Thematik sehr ähnlich sind.

Im 2. Kapitel werden die theoretischen Grundlagen geschildert und betrachtet. Der dritte Teil der Arbeit beschreibt die praktische Umsetzung des entwickelten Systems, sowie die Erläuterung der einzelnen Komponenten.

Im anschließenden Kapitel wird eingangs die Methodik dargelegt und abschließend die Forschungsergebnisse präsentiert.

Das letzte Kapitel gibt eine Bewertung zu den Studienergebnissen ab und ordnet diese in den Kontext der Forschungsfrage ein. Zum Schluss wird ein kurzer Ausblick gegeben und Überlegungen zu weiterführenden Forschungsfragen durchgeführt.

1.2 Problemstellung

Im heutigen Zeitalter des Smartphones in dem die Überstimulation [Ste31] der Sinne durch Informationen von den unterschiedlichsten Kanälen (z.B. Social Media, Push-Notifications etc.) zum Alltag vieler gehört, ist es mittlerweile unabdingbar geworden neben Werbeanzeigen, auch Apps, Systeme und Services zu personalisieren. Dabei soll dem Nutzer eine User-Experience geboten werden, welche ihn einerseits vor dieser Überstimulation durch irrelevante Informationen bewahrt und andererseits eine möglichst lange Interaktion mit der App, Website, Plattform gewährleistet.

Dahingehend ist der erste Schritt, wie auch bei den maßgeschneiderten Werbeanzeigen auf den Social-Media Plattformen, das Verhalten der User zu erlernen und anhand diesem Wissens App-Interfaces oder auch Hintergrundprozesse anzupassen, sodass für jeden Nutzer eine optimale User-Experience sichergestellt werden kann. Hierbei kann auch von virtuellen Assistenten gesprochen werden, welche durch die direkte Interaktion mit dem User oder durch das Beobachten des Users, dessen Verhalten und Gewohnheiten versuchen zu erlernen.

Die Problematik die diesen Anwendungen anhaftet, ist die Weitergabe der privaten Nutzerdaten an die Plattform- bzw. App-Server, auf welchen die Informationen abgespeichert und die Assistenten trainiert werden, da vor allem bei mobilen Endgeräten die Kapazitäten und Rechenleistung dafür nicht ausreichen.

Eine Möglichkeit diese Weitergabe zu vermeiden, besteht in der Blockchain-Technologie. Hierbei werden die Daten nicht mehr zentral bei einem Knoten im Netzwerk abgespeichert, sondern dezentral in einer verteilten Datenbank, in der die Sicherheit der Daten und Privatsphäre der Nutzer gewährleistet ist. Aufgrund dieser Dezentralität können virtuelle Assistenten auch lokal gehos-

tet und trainiert werden, ohne dabei die Daten nach außen geben zu müssen. Durch diesen Ansatz entstehen jedoch bestimmte Problemstellungen, für die es in der Form noch keine standardisierten Lösungsansätze gibt und erst zu eruieren gilt.

So lautet die Forschungsfrage: “Inwiefern lassen sich Machine Learning und die Blockchain Technologie verbinden, beziehungsweise wie gut lässt sich dies bereits umsetzen?”

Konkret auf einen Anwendungsfall bezogen formuliert: “Ist es möglich einen Machine Learning Algorithmus anhand von Daten einer Blockchain zu trainieren, um damit das Verhalten eines Users zu erlernen?” Gerade im Bezug auf das Erlernen des Nutzerverhaltens ist dies ein noch sehr unerforschtes Gebiet. Um dies zu erforschen und abzubilden, wird in dieser Arbeit ein System vorgestellt, welches sich den gerade eben geschilderten Fragen annimmt.

Hierbei wird eine Getränkeliste an einem Lehrstuhl durch eine Tablet-App ersetzt, in welcher die konsumierten Getränke (Kaffee, Club Mate, Wasser) eingegeben werden. Die kontextuellen Daten werden daraufhin auf eine private Blockchain gespeichert und das Getränk mit einem Blockchain basierten Token bezahlt. Ein lokal gehosteter Reinforcement Learning Algorithmus liest die Informationen von der Blockchain aus und lernt für jeden Nutzer ein Modell, welches dessen Kaffeetrinkverhalten abbildet. Mit dem Fokus auf das Kaffeetrinkverhalten, soll einerseits die Komplexität des Lernproblems reduziert und andererseits die Durchführung einer problemspezifischen “Feature Selection” ermöglicht werden.

Das System wird am Lehrstuhl installiert und im Zeitrahmen von 2 Monaten getestet. In diesem Zeitraum soll eruiert werden, inwiefern es der Learner-Instanz möglich ist, das jeweilige Kaffeetrinkverhalten eines Lehrstuhlmitarbeiters zu erlernen.

1 Einleitung

Außerdem wird eine Anwendung zur Simulation eines realen Users präsentiert, deren Zweckmäßigkeit darin besteht, die Funktionstüchtigkeit des implementierten Algorithmus bezüglich dessen Lernfähigkeit zu bezeugen. Dabei werden die Daten dem Lernalgorithmus direkt übergeben, ohne diese zuvor auf die Blockchain zu schreiben.

Abschließend werden die Ergebnisse der Simulation analysiert und bei der Auswertung der Resultate aus der Testphase bzw. Studie in Bezug genommen.

1.3 Related Work

In diesem Abschnitt werden andere Arbeiten aufgeführt, welche in der Thematik Ähnlichkeit in einen oder mehreren Punkten der Problemstellung bzw. Lösungsansätzen, aufweisen.

Es sei darauf hingewiesen, dass aufgrund der speziellen Problemstellung und der Neuartigkeit der Technologien, lediglich zwei Arbeiten gefunden wurden, welche jeweils in den Punkten Blockchain und Machine Learning an Similaritäten verfügen.

Zaidenberg et. al. [ZRM10] beschreiben einen virtuellen (Desktop-)Assistenten, welcher in der Algorithmik und Vorgehensweise zu der in dieser Arbeit sehr ähnlich ist. Der Anspruch des virtuellen Assistenten ist die Unterstützung eines Users bei seinen täglichen Aufgaben. So soll dieser z.B. sicherstellen, dass der Nutzer keine Termine verpasst oder durch Benachrichtigungen nicht abgelenkt wird (Email Programm wird vom Assistenten geschlossen), sollte er sich gerade in einem Meeting befinden.

Das Ziel ist die Reduktion der kognitiven Belastung des Users durch alltägliche

Aufgaben.

Die Herausforderung des Systems besteht in der automatischen Akquirierung der User Präferenzen bzw. dem Erlernen des Userverhaltens und ist somit der Problemstellung dieser Arbeit äußerst ähnlich.

Als Lernalgorithmus wird auch hier eine Form des Q-Learning verwendet (DYNA-Q) [SB98], welches im Gegensatz zum modellfreien Ansatz des klassischen Q-Learning, zusätzlich ein Modell der Umgebung konstruiert, um die Dauer der Lernphase zu verkürzen.

Die Unterschiede zur Systemarchitektur dieser Arbeit, liegen zum einen in der Vakanz einer Blockchain Komponente und zum anderen in der stärkeren Präsenz des Assistenten, welcher in einer weitaus ausgeprägteren Form mit dem User interagiert und auf ein direkteres Feedback angewiesen ist.

Singla et. al [SBK18] schildern ein Blockchain (Ethereum) basiertes System, um Userverhalten bzw. Useraktivität vorherzusehen.

Konkret wird eine Personalisierung von Smart Home Geräten durchgeführt, indem individuelle User-Geräte-Profile erstellt und dezentral (IPFS) abgespeichert werden. Betritt ein User einen Raum, wird dieser vom System erkannt und die entsprechenden Profile für jedes Gerät im Raum geladen. Diese Profile werden jedoch nicht direkt auf der Blockchain abgespeichert, sondern nur deren Hashwerte. Wobei jeder Hashwert respektive jedes Profil einem Block auf der Blockchain entspricht. Der eigentliche Speicherort der Profile befindet sich auf dem IPFS (Interplanetary File System). IPFS ist eine verteilte Datenbank (verteiltes Dateisystem) mit einer Peer-to-peer Netzwerktopologie, die es ermöglicht Daten dezentral zu speichern [IPFb, IPFa].

Somit dienen die auf der Blockchain gespeicherten Hashwerte als Key, um das richtige User-Geräte-Profil von IPFS zu laden. Der Austausch und Zugriff der

1 Einleitung

Profile wird dabei von Smart Contracts übernommen.

Zum Erlernen der Useraktivität wird eine Assoziationsanalyse (association rule mining) verwendet. Dabei werden die Gerätelogs periodisch durchsucht und anhand der vordefinierten Geräte Profil Daten (Waschmaschine Wochentag, Zeit, Wassertemperatur, etc.) die Userprofile erstellt. Eine Assoziationsanalyse funktioniert dahingehend, dass es anhand von Auftretswahrscheinlichkeiten Korrelationen zwischen Eingabeparametern ermittelt und dadurch Muster in Datensätzen aufdeckt[CSPK07, onl].

Im Kontrast zur Problemstellung dieser Arbeit, liegt der Unterschied in der geringeren Komplexität des Lernproblems, weswegen die Ansätze bzw. Algorithmen, auf welche Art und Weise das Userverhalten gelernt wird, gänzlich differieren.

2 Stand der Technik

2.1 Blockchain

2.1.1 Funktionsweise & Konzepte

Damit das Verständnis für die Blockchain Technologie geschaffen werden kann, sollte zuerst die darunter liegende Technologie betrachtet werden. So wird der Begriff der Blockchain oft auch mit dem des *distributed ledger* gleichgesetzt. Ein *distributed ledger* ist eine verteilte Datenbank, welche im Kontext einer Blockchain auch als verteiltes Konto bzw. dezentral geführtes Kontobuch [Lui15] verstanden wird, in welchem jegliche Transaktionen abgespeichert werden.

Allerdings ist diese Parität nicht vollkommen korrekt, denn eine Blockchain ist nur ein spezieller Typus eines *distributed ledgers*. Im Allgemeinen kann dieser nämlich neben Transaktionen, auch aus weiteren Daten bestehen, wohingegen bei einer Blockchain die Blöcke stets Transaktionen beinhalten.

Somit stellt ein *distributed ledger* die technologische Grundlage aller virtuellen Währungen dar und ist dadurch auch einer der Hauptgründe weshalb Kryptowährungen auf einer Blockchain basieren.

2 Stand der Technik

Grundsätzlich steht Blockchain für eine Verkettung von geordneten Blöcken, welche in sich eine oder mehrere Transaktionen beinhalten. Zu den Transaktionsdaten wird zudem ein Hashwert des Vorgängerblocks und ein Zeitstempel in einem Block abgespeichert. Erst aufgrund der Berücksichtigung des Hashwerts des vorherigen Blocks werden die Blöcke miteinander verknüpft und bilden somit eine chronologisch geordnete Kette [Bas18].

Eine weitere Beschaffenheit einer Blockchain ist die Dezentralität durch das aufgespannte *Peer-to-Peer* Netzwerk. Der Vorteil dieser Topologie besteht in der Vakanz eines zentralen Knotens (*Node*), der für das Abspeichern und Bereitstellen aller bestehenden Daten eines Netzwerks zuständig ist. Jeder Knoten in einem solchem Netzwerk ist sowohl Sender als auch Empfänger, sodass alle Teilnehmer eine Kopie des Datenbestandes besitzen, was einen *Single Point of Failure* völlig ausschließt. Aus diesem Grund sind Daten einer Blockchain nie nur bei einem Knoten abgespeichert, sondern sämtliche Nodes besitzen eine Kopie der aktuellen Blockchain.

Welche Daten schließlich auf die Blockchain geschrieben werden dürfen oder genauer gesagt ob eine Transaktion durchgeführt werden darf, erfolgt stets in Anbetracht des Konsens aller Parteien im Netzwerk. Diese Eigenschaft wird als *distributed consensus* bezeichnet und ist das Fundament einer jeden Blockchain. Durch die Partizipation eines jeden Teilnehmers in die Entscheidungsfindung, wird die Notwendigkeit einer zentralen Entscheidungsinstanz obsolet. Damit löst die Blockchain folgendes Gedankenexperiment von Lamport aus dem Jahr 1982. Eine Gruppe an byzantinischen Generälen, die jeweils verschiedene Teile der byzantinischen Armee befehligen, planen einen Angriff auf eine Stadt. Dieser resultiert allerdings nur in einem Sieg, wenn alle gemeinsam angreifen. Der einzige Weg der Interkommunikation und Koordination ist via

Boten. Somit entsteht ein Dilemma, sollten sich ein oder mehrere Verräter unter den Generälen befinden und falsche Informationen streuen.

Aus diesem Grund bedarf es einem Mechanismus der es erlaubt, trotz Falschinformationen durch potentielle Verräter, einen Konsens unter den Generälen bezüglich eines gemeinsamen Angriffs zu schaffen.

Analog zur Blockchain entsprechen die Generäle den Knoten im Netzwerk, die Verräter sind “böartige” (malicious) Knoten und die Boten sind die Kommunikationskanäle zwischen den Knoten.

Gelöst wurde diese Problematik durch die Publikation des *Practical Byzantine Fault Tolerance (PBFT)* Algorithmus (1999, Castro und Liskov), welcher nach einer bestimmten Anzahl an Nachrichten mit gleichem signierten Inhalt einen Konsens unter allen Knoten herstellt [Bas18].

Der Mechanismus, dessen Zuständigkeitsbereich das Erzielen des Konsens betrifft, ist je nach Implementierung des Blockchainprotokolls unterschiedlich. Eine populäre Methodik, welche unter anderem von Bitcoin und Ethereum verwendet wird, ist der *Proof-of-Work (PoW)* Ansatz. Hierbei müssen von den sog. “Minern” Iterationen an aufwändigen und komplexen Berechnungen (“Puzzles”) durchgeführt werden, bevor von diesen ein neuer Block zur Kette hinzugefügt werden darf. Erst nach der erfolgreichen Verifikation dieser Puzzles, wird ein neuer Block generiert. Der enorme Aufwand, welcher im ersten Schritt aufgewendet werden muss, sollen das Netzwerk vor Angriffen schützen und somit dessen Sicherheit und Stabilität gewährleisten.

Als eine weitere Konsens-Methodik ist *Proof-of-Stake* zu nennen. Hierbei können sich Netzwerkteilnehmer als “Validators” registrieren lassen und daraufhin einen Block vorschlagen (“vote”), der als nächstes generiert werden soll. Die Auswahl des nächsten Blocks und somit die Bestätigung des Votes, erfolgt durch eine (pseudo) Zufallsauswahl, welche eine Gewichtung aufgrund des

2 Stand der Technik

Vermögens (“stake”) des Teilnehmers in der Blockchain spezifischen Währung erfährt. Dabei ist das Vermögen proportional zum Anteil der Transaktionen die ein Validator bestätigen kann. Dadurch ist es um so wahrscheinlicher, dass ein Teilnehmer einen Block generieren darf, desto höher sein Stake ist.

Diese Methodik hat mehrere Vorteile gegenüber *PoW*, der GröÙte jedoch besteht in der Recheneffizienz. So werden bei *PoW* durch die Berechnung der Puzzles große Mengen an Strom verbraucht, um das Netzwerk letztlich vor Angreifern zu schützen. Dies ist im Hinblick auf *PoS* äußerst ineffizient, da der Schutzmechanismus nicht auf der Erbringung einer gewissen Rechenleistung beruht, sondern ein Validator bei netzwerkwidrigem Verhalten seinen Stake verliert und somit sichergestellt wird, dass alle Teilnehmer im Sinne des Netzwerks agieren [Vit26, Vit20].

Eine weitere Besonderheit einer Blockchain ist die Unveränderbarkeit der Blöcke. So gibt es, im Gegensatz zu den bekannten CRUD-Operationen, welche zur Kommunikation zwischen Client und Server verwendet werden, Daten zu schreiben, zu downloaden, zu löschen und zu editieren, bei einer Blockchain lediglich eine Schreib- und eine Leseoperation. Weswegen im Zusammenspiel mit dem Konsensverfahren, Transaktionen auf einer Blockchain einzig hinzugefügt und gelesen, jedoch nie zu einem späteren Zeitpunkt gelöscht oder editiert werden können.

Dabei sind die gespeicherten Daten (unverschlüsselt oder auch verschlüsselt) für alle im Netzwerk einsehbar. Dies bedeutet somit volle Transparenz für jeden User.

Die gerade geschilderten Eigenschaften sind einer jeden Blockchain inhärent. Was jedoch erst in neueren Blockchains (Blockchain 2.0) vorzufinden ist, sind

die *Smart Contracts* [Dav15, Bas18].

Smart Contracts sind sozusagen Anwendungen die auf einer Blockchain installiert werden können.

Diese Programme können z.B. Business Logiken abbilden und ausführen oder auch Verpflichtungen und Vereinbarungen im rechtlichen Sinne durchsetzen, ohne der Notwendigkeit eines Mittelsmanns, welcher das Vertrauen aller beteiligten Parteien inne hat. Diese “Trust-Komponente” wird durch die Anerkennung des Smart Contracts von allen Parteien übernommen.

Das erste Mal in Erscheinung getreten sind Smart Contracts mit der Veröffentlichung der Ethereum Blockchain, welche nun im Anschluss genauer betrachtet wird.

2.1.2 Ethereum

Die Ethereum Blockchain wurde 2015 in Betrieb genommen, mit dem Ziel nicht nur eine Kryptowährung zu schaffen, sondern eine Plattform zu entwickeln, auf welcher sogenannte *Dapps* (Decentralized Apps) betrieben werden können. So wird Ethereum im Vergleich zu Bitcoin auch als Blockchain 2.0 bezeichnet. Dies ist der Tatsache geschuldet, dass es eben nicht nur eine Kryptowährung umfasst, sondern aufgrund der Smart Contracts möglich ist Software auf einer Blockchain zu installieren.

Als Ethereum wird genauer genommen das Protokoll tituiert, welches die Blockchain implementiert. Die Kryptowährung die auf der Blockchain basiert wird als *Ether* bezeichnet und fungiert zudem als Zahlungsmittel im Kontext der Smart Contracts. So ist das Bestreben der Gründer nicht eine weitere Kryptowährung zu schaffen, sondern einen Art “Supercomputer”, welcher immer online ist und aus Millionen von Computern im Netzwerk besteht, die

2 Stand der Technik

ihre Rechenleistung zur Verfügung stellen und als Gegenleistung in Form von Ether vergütet werden [Bus18].

Dabei können zwar, wie bei einer Kryptowährung, Transaktionen durchgeführt und Ether von einem Konto auf ein anderes transferiert werden. Jedoch liegt der eigentliche Fokus auf den Smart Contracts und den Dapps.

Dazu wurde die EVM (Ethereum Virtual Machine) entwickelt, in welcher letztlich die Smart Contracts bzw. Dapps gehostet und ausgeführt werden. Dabei stellt die Virtual Machine eine Abstraktionsebene zur physischen Schicht der Blockchain dar. Sie ermöglicht den Smart Contracts, Daten auf die Blockchain zu speichern und bietet diesen gleichzeitig eine Laufzeitumgebung, in der die Anwendungen ausgeführt werden können.

Die Implementierung der Smart Contracts erfolgt stets in der eigens entwickelten Programmiersprache *Solidity*. Diese folgt dem Prinzip der Objekt Orientierung, ist der Sprache Javascript angelehnt und zudem Turing-Vollständig, was im Bezug auf die Entwicklung von Apps eine hohe Bandbreite an Anwendungsfällen eröffnet.

Damit die Funktionen der Smart Contracts überhaupt genutzt werden können, muss einem jedem Methodenaufruf genauer genommen jeder Transaktion Ether - im Kontext der Smart Contracts als *Gas* bezeichnet - mitgegeben werden, um die Miner für ihre zur Verfügung gestellte Rechenleistung zu entlohnen. Das bedeutet, um Daten auf die Blockchain zu speichern, wird je nach Transaktion eine bestimmte Menge an Gas benötigt, welches im Endeffekt den Minern als Anreiz zur Bereitstellung von Rechenleistung übertragen wird.

Zu den Dapps gilt es anzumerken, dass diese zwar als vollkommen dezentrale Applikationen gedacht sind, bisherige Umsetzungen jedoch lediglich eine "semi-dezentrale" Lösung darstellen.

So ist das Backend einer Dapp, hinsichtlich der Smart Contracts, meist als

dezentral zu betrachten. Da es jedoch nicht möglich ist Frontend Templates auf der Ethereum Blockchain zu hosten, sind diese in der Regel immer auf einem Server abgespeichert, wodurch eine Dapp nie als vollkommen dezentral angesehen werden kann.

Zudem sind, trotz der Turing-Vollständigkeit von *Solidity*, aufwendige Berechnungen und Abbildungen komplexer Business Prozesse infolge der Transaktionskosten nur schwer oder gar nicht zu bewerkstelligen, weswegen diese Logiken gewöhnlich auch auf einen Server ausgelagert werden.

In naher Zukunft könnte hierbei ein Umstieg auf IPFS Abhilfe schaffen, indem es an Stelle der Server als Speicherort der Frontend-Logik verwendet wird.

Wie das Mining bereits impliziert, basiert der Konsensalgorithmus der Ethereum Blockchain auf dem Proof of Work Konzept. Da dieser Ansatz jedoch einige Nachteile mit sich bringt, was Energieeffizienz und Transaktionen pro Sekunde betrifft, wird bereits an der Implementierung eines Proof of Stake Konsensalgorithmus gearbeitet. Eine erste Version wurde am 07.05.2019 im *Prysm Eth2 Testnet* bereits veröffentlicht [Pre07].

Die große Community, der Opensource Ansatz, die Verfügbarkeit von Libraries in verschiedenen Programmiersprachen und Kommandozeilenanwendungen, sowie die Möglichkeit Smart Contracts bzw. Dapps auf der Blockchain zu betreiben, sind einer der Hauptgründe, weshalb die Wahl bei dieser Arbeit auf Ethereum gefallen ist. [Bus18, Lui15, Ant17]

2.2 Machine Learning

2.2.1 Überblick

Der Fortschritt in der Entwicklung von Machine Learning Algorithmen in den letzten Jahren hat dazu geführt, dass Algorithmen in vielerlei Bereiche des Alltags Einzug gefunden haben. Vor allem im Kontext von Aufgabenstellungen bei denen eine große Menge an Daten vorhanden ist, werden diese Algorithmen zur Extrahierung relevanter Informationen verwendet.

Mit dem Anstieg der Digitalisierung entstehen immer mehr Datensätze von solch enormer Größe. Sei es durch die Vernetzung von Geräten und Sensoren im Bereich des IoT oder auch das Sammeln von Nutzerdaten durch Smartphone-Apps, die Intention besteht stets in der Erlangung eines besseren Verständnisses für eine konkrete Problemstellung in der realen Welt.

Aufgrund dessen ergeben sich eine Vielzahl an Anwendungsfeldern für Machine Learning Algorithmen.

Eines davon fällt unter die Begrifflichkeit des Data Mining's. Hierbei erfolgt eine Entscheidungsfindung durch den Algorithmus anhand von Datenaufzeichnung in einem bestimmten Gebiet. Als Beispiel wäre hier die Analyse von Krankenakten zu nennen, welche zu einem besseren Verständnis von Krankheitsbildern, sowie einer genaueren Diagnose führen soll.

Ein weiteres Anwendungsgebiet sind Problemstellungen, deren Lösung nicht durch eine (klassisch) imperativ implementierte Software gefunden werden kann. Dies ist dann der Fall, wenn die Anwendung aufgrund der enormen Vielzahl an Varianten nicht alle möglichen Status abbilden kann. Zu nennen sind hier die Spracherkennung oder auch das autonome Fahren, welche schlichtweg nur mit Machine Learning realisierbar sind.

In die Kategorie des nächsten Gebietes, fällt auch die Problemstellung dieser Arbeit. Es handelt sich um die Personalisierung (“customization”) von Software-Anwendungen und Services. Dabei wird Machine Learning dazu verwendet, um das Verhalten der User besser zu verstehen. Die gesammelten Nutzerdaten werden analysiert und Präferenzen der User aufgrund ihres (Online-)Verhaltens abgeleitet. So werden zum Beispiel im Falle von Facebook oder Google, dadurch gezielt Werbeanzeigen platziert und auf den User abgestimmt. [Mit04]

Somit eignen sich je nach Anwendungsgebiet und Problemstellung unterschiedliche Typen von Machine Learning Algorithmen. Eine inhärente Eigenschaft eines Machine Learning Algorithmus ist die Notwendigkeit eines Input Y in Form von *Features*. Ein *Feature* ist eine numerische Repräsentation eines Teilaspekts der Inputdaten. Dabei ist die Konkatenation jener auch als *Feature Vektor* bekannt. Machine Learning beschreibt somit die Identifikation von Strukturen und Mustern auf Basis einer Kollektion von Feature Vektoren. [Zhe18]

Generell lässt sich eine Unterteilung der Machine Learning Algorithmen in drei Typen machen, dabei erfolgt eine Unterscheidung aufgrund der Art des Lernprozesses.

Die wohl bekannteste Kategorie ist das Supervised Learning. Algorithmen die unter diesen Teilbereich fallen, haben das Ziel für einen Input X einen Output Y zu finden. Hierfür wird diesen in der Lernphase ein Datensatz vorgelegt, bei welchem diese Abbildung ($X \rightarrow Y$) bereits vorhanden ist und die Algorithmen sich somit Wissen aneignen können. Nach dem erfolgreichen Durchlauf der Lernphase sind die Algorithmen in der Lage bei Eingabe von unbekannten Input-Daten, basierend auf dem angeeigneten Wissen, eine Generalisierung

2 Stand der Technik

durchzuführen und einen Output Y (*Labels*) für X zu liefern.

Der Nachteil dieser Algorithmen liegt in der Aufbereitung und Bereitstellung dieser Lerndaten, welche in der Regel manuell erstellt werden müssen. [ES18]

Im Kontrast dazu ist das Unsupervised Learning zu verstehen. Diese Algorithmen benötigen zum Lernen nur einen Input X im Datensatz. Trotz fehlender *Labels* können diese im Datensatz statistische Strukturen aufdecken bzw. ein Modell für diesen Datensatz erarbeiten. Als Beispiele hierfür sind das Clustering (k-Means) der Daten oder die Reduktion (PCA) zu nennen. [ES18, Mit04, SB98]

Ein weiterer Lernansatz welcher sich zwischen den beiden eben genannten Kategorien einreicht, trägt die Bezeichnung Reinforcement Learning und wird im nächsten Unterkapitel genauer betrachtet.

2.2.2 Reinforcement-Learning

Das folgende Unterkapitel basiert auf der Literatur folgender Quellen: [Wei01, Mit04, SB98]

Das Grundprinzip des Reinforcement-Learning besteht darin, bestimmte Situationen auf Aktionen zu projizieren, um dabei den numerischen Reward des Agenten zu maximieren. Diese Aktionen werden dem Agenten jedoch nicht durch eine “Supervisor-Instanz” mitgeteilt, hingegen versucht dieser durch die “Trial and Error” Methodik herauszufinden, welche Aktion in welchem Zustand den größten Reward zur Folge hat. Dabei können diese Aktionen nicht nur die Belohnung des Agenten beeinflussen, sondern zudem die Umgebung in

der er sich bewegt und dadurch auch den Folgezustand.

Dieses Konzept der Reward-Maximierung resultiert in dem Tradeoff zwischen *Exploration* und *Exploitation*. *Exploration* beschreibt den Versuch mehr Information über die Umgebung zu erlangen, indem die Reward-Maximierung außer Acht gelassen und eine Aktion zufällig ausgewählt wird, mit dem Ziel den bisherigen Reward zu übertreffen. Im Gegenteil dazu spezifiziert *Exploitation* die Maximierung des Rewards. Hierbei greift der Agent stets auf die, in einem bestimmten Zustand, bestbewertete Aktion zurück.

Je nach Algorithmus und Lernproblem variiert dieses Verhältnis, welches durch eine iterative Justierung der Parameter anzupassen gilt.

Durch die Wechselwirkung ist es die Aufgabe des Agenten eine Strategie zu finden, die letztlich den maximalen Reward garantiert, indem es sein Verhalten in den jeweiligen Zuständen bereits vorgibt.

Dabei gilt es vor allem zu beachten, ob es sich bei dem Lernproblem um ein Deterministisches oder Nichtdeterministisches handelt. Deterministisch bedeutet, es wird stets die gleiche Aktion in einem bestimmten Zustand gewählt, wohingegen nichtdeterministisch lediglich eine Wahrscheinlichkeitsverteilung beschreibt, anhand jener der Agent in einem Zustand entscheidet, welche Aktion auszuwählen ist.

Im Allgemeinen lassen sich folgende inhärente Eigenschaften an Reinforcement Learning Algorithmen feststellen:

- Agent: lernendes System, dessen Ziel es ist seine Belohnung zu maximieren, indem es seine Umgebung wahrnimmt und mittels Aktionen beeinflusst

2 *Stand der Technik*

- Umgebung: diese beschreibt Bestandteile außerhalb des lernenden Systems (andere Agenten, Programme, Menschen etc.) und ist entweder von deterministischer oder nichtdeterministischer Natur
- Reward: numerischer Wert der die Nützlichkeit einer Aktion im aktuellen Zustand beschreibt und dadurch als direktes Feedback für den Agenten zu sehen ist
- Wertfunktion: ist die Akkumulation aller erhaltenen Belohnungen, startend bei einem bestimmten Zustand bis hin zum Ende eines Durchlaufs und stellt somit eine Schätzung bezüglich der Wertigkeit dieses Zustandes dar
- Strategie: definiert das Verhalten des Agenten zu einem bestimmten Zeitpunkt bzw. welche Aktionen in welchen Zuständen ausgeführt wird
- Umgebungsmodell: ist die Abbildung des Verhaltens der Umgebung, um kommende Zustände und Belohnung vorherzusagen und auf dieser Basis eine geeignete Strategie zu finden

Jedoch sind nicht alle Reinforcement Algorithmen gleich, sondern in ihrer Definition und Umsetzung der eben genannten Eigenschaften oftmals sehr unterschiedlich. Eine Kategorisierung wird in der Regel anhand der Berechnung der Wertfunktion durchgeführt, woraus sich drei verschiedene Klassen an Reinforcement Algorithmen ergeben. Diese differieren in der Berechnung der Wertfunktion durch verschiedene Lösungsansätze.

Als erste Kategorie ist die Dynamische Programmierung (DP) zu nennen. Jene Algorithmen sind mathematisch präzise, benötigen dafür aber ein perfektes Umgebungsmodell, welches bereits die Rewards und Übergangswahrscheinlichkeiten aller Zustände beinhaltet. So beruht die Berechnung der Wertfunktion auf dem *Markov-Entscheidungsproblem* (*MEP* oder auch *MDP Markov decision process*). Dieses Modell erlaubt die Erarbeitung einer Strategie zur Reward-Maximierung, unter der Berücksichtigung der bekannten Übergangswahrscheinlichkeiten und den erwarteten Rewards. Das ist auch der große Nachteil der Methodik, da diese Berechnungen sehr rechenaufwändig sind und ein perfektes Umgebungsmodell in den meisten Fällen nicht vorhanden ist.

Unter die nächste Kategorie fallen die Monte Carlo (MC) Methoden. Im Gegensatz zur Dynamischen Programmierung berechnen jene lediglich eine Schätzung der Wert-Funktion und finden aufgrund dessen die optimale Strategie. Außerdem benötigen diese auch kein Vorwissen über die Umgebung, sondern lernen durch die Interaktion mit der Umgebung, die Abfolge von Zuständen und Aktionen, sowie deren Rewards. Bei der Berechnung der Wertfunktion wird stets bis zum Ende einer Episode gewartet und eine Mittelung über alle erhaltenen Rewards durchgeführt. Dies funktioniert jedoch nur bei Problemstellungen, in denen Episoden auch terminieren. Sogleich ist das auch ein Nachteil der MC Methoden, da eine Bewertung eines Zustandes immer erst am Ende einer Episode realisiert werden kann.

Der letzte Ansatz wird als Temporal Difference (TD) Learning bezeichnet. Hierbei wird im Kontrast zu den MC Methoden nicht erst auf die Beendigung der Episode gewartet, sondern die Wertfunktion eines Zustandes gleich im nächsten Zeitschritt geschätzt. Diese Schätzungen basieren wiederum auf

2 Stand der Technik

den bereits vorangegangene Schätzungen und nähern sich dadurch inkrementell dem “wahren” Wert der Wertfunktion an, ohne dabei wiederholt auf die Terminierung einer Episode warten zu müssen.

So ist auch bei dieser Art von Algorithmen kein Umgebungsmodell von Nöten, stattdessen werden die Bewertungen der Aktionen über die Zeit hinweg erörtert und gesammelt.

3 System Architektur

3.1 Überblick

Im folgenden wird nun die System Architektur und der Workflow erläutert, welche als Grundlage für die Studie dienen, um die in 1.2 geschilderte Problemstellung abzubilden und letztendlich zu lösen.

3.1.1 Architektur

Um ein besseres Bild davon zu bekommen, wie die einzelnen Komponenten zusammenhängen bzw. welche Aufgaben diese in Wechselwirkung zu anderen Instanzen übernehmen und ausführen, wird zunächst die Architektur des Systems erläutert. Als Basis soll dabei die Abbildung 3.1 dienen, anhand jener vorrangig die jeweiligen Komponenten bezüglich ihrer Funktionsweise beschrieben werden. Das Zusammenspiel der Anwendungen wird im Anschluss unter 3.1.2 detailliert beleuchtet.

3 System Architektur

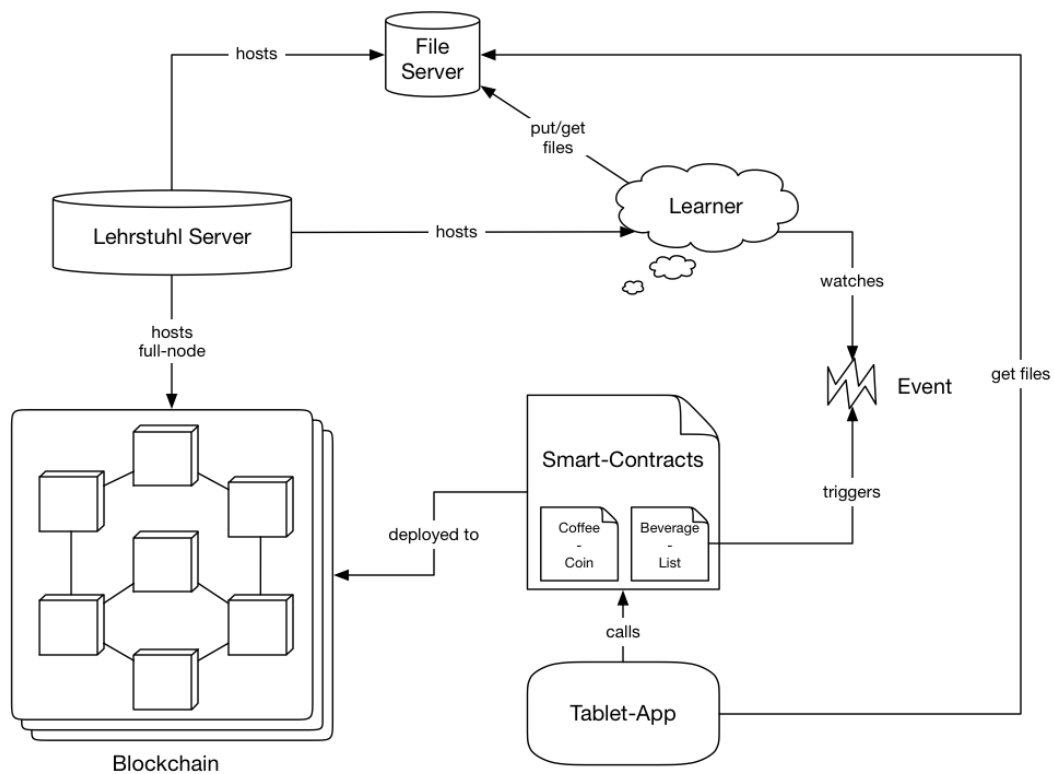


ABBILDUNG 3.1: Systemarchitektur

Lehrstuhl Server

Der Lehrstuhlserver ist dafür zuständig einen Großteil der Anwendungen zu hosten bzw. zu starten. Dies gilt sowohl für den HTTP-Fileserver als auch für die Learner-Anwendung, welche dessen Betriebssystem als Plattform nutzen. Auch die Blockchain wird auf dem Server gestartet und verwendet diesen zudem als *full-node*, um Transaktionen zu minen.

Der Server ist im Grunde die Basis der Anwendungen, dessen primäre Funktionsweise darin besteht, jenen eine Plattform zu bieten und mit Rechenleistung zu versorgen.

Learner

Der sogenannte *Learner* ist eine in Golang implementierte Softwareanwendung, deren Hauptaufgabe darin besteht, das Kaffeetrinkverhalten der Nutzer zu erlernen - wovon auch die Namensgebung der Anwendung stammt. Um dies zu erreichen, wurde die Anwendung in Submodule unterteilt, welche einen dedizierten Aufgabenbereich abdecken und eigenständig bearbeiten. Trotz ihrer autarken Funktionsweise, kann das Trinkverhalten letztendlich erst in der gegenseitigen Wechselwirkung jener erlernt werden.

Die Submodule lauten wie folgt:

- Q-Learning
- Worker
- Watcher
- (Smart Contract Deployment Skript)

Das **Q-Learning** ist, wie der Name bereits impliziert, für das eigentliche Erlernen des Trinkverhaltens zuständig. Es ist im Grunde die Implementierung des Q-Learning Algorithmus, sowie die damit einhergehende Zustandsraummodiellerung, welche aber unter 3.3.1 genauer erläutert wird.

Der **Worker** ist einerseits für die Userverwaltung und andererseits für die, in einem festgelegten Intervall stattfindende, Ausführung des Q-Learning Algorithmus, zuständig. (vgl. Kap. 3.3.3)

Der **Watcher** beobachtet Events, die vom Smart-Contract *Beveragelist* ausgelöst wurden. Die Daten, welches das Event beinhaltet, werden daraufhin verwendet den Q-Learning Algorithmus damit zu "befüllen" und aufgrund dessen

3 System Architektur

das Trinkverhalten zu erlernen.

Das **Smart Contract Deployment Skript**, ist in der Form zwar nicht in der Systemarchitektur vorhanden. Da es aber auch ein Submodul des *Learners* und für das gesamte Konstrukt dahingehend essentiell ist, weil es die Smart Contracts auf der Blockchain installiert und im Zuge dessen erst die Verbindung zwischen Blockchain und Learner ermöglicht, wird es in dieser Auflistung trotzdem aufgeführt.

Fileserver

Der Fileserver ist eine Golang-Anwendung, welche eine rudimentäre REST (REpresentational State Transfer) API (Application Programming Interface) zur Verfügung stellt. Von den bereits erwähnten *CRUD* Operationen, welche als grundlegend für alle persistenten Datenspeicher angesehen werden können, implementiert dieser nur das *GET* und das *PUT*. Sowohl die *PATCH* als auch die *DELETE* Operation bieten keinen Mehrwert für die Gesamtarchitektur bzw. den Workflow und sind in Anbetracht dessen nicht implementiert.

Das bedeutet, die Hauptaufgabe des Fileservers besteht darin, Dateien zu empfangen und zu speichern (*PUT*) und diese auf Anfrage (*GET*) an einen Antragsteller wieder zu versenden.

Außerdem bietet die Anwendung zusätzlich zur API einen UDP-Broadcast, welcher vor allem beim "Testing" und beim Setup eine große Erleichterung darstellt. Dieser Broadcast versendet in seinen Nachrichten die IP-Adresse des Lehrstuhl-Servers und somit seine eigene, sowie die der Blockchain.

Da die IP-Adresse und der Port des Broadcasts stets gleich bleiben, sich aber die Host-IP der Blockchain und des Fileservers je nach Deployment theoretisch ändern können - was in der Entwicklungsphase sehr oft der Fall war - müssen

sich sowohl der Learner als auch die Tablet-App lediglich auf den Broadcast “subscribe” und können dadurch die IP der Blockchain und des Fileservers erfahren. Durch diese dynamische Zuweisung der IP-Adresse, müssen keine Updates beim Learner und der App durchgeführt werden, sollte die Blockchain und der Fileserver auf einem anderen Host deployed werden.

Aufgrund der Tatsache, dass sich die IP-Adresse des Lehrstuhl-Servers während der Studie nicht ändert, ist der UDP-Broadcast auch nicht in der Abbildung (3.1) der Systemarchitektur berücksichtigt worden. Der Anwendungsbereich ist trotz alledem im Bereich der Testphase und auch für die künftige Projekte, bei denen das System Verwendung findet, definitiv vorhanden.

Blockchain

Die Blockchain ist eine private, eigens für die Studie erstellte Ethereum-Blockchain, dessen *Genesis-Block* aus dem JSON-File (vgl. Abbildung 3.3) generiert wird. Die Erläuterungen zu den jeweiligen Key-Value-Pairs sind unter Kap. 3.2.1 zu finden. Das Generieren und Starten der Blockchain erfolgt auf dem Lehrstuhlserver. Dabei hostet der Server zudem eine sogenannte *full-node* (hier auch *miner* genannt) der Blockchain, welche dafür zuständig ist Transaktionen zu berechnen und zu bestätigen. Aus Ressourcengründen ist der *miner* der einzige im Gesamtsystem, was aus theoretischer Sicht einen Single Point of Failure als Nachteil mit sich zieht. Das bedeutet, sollte die *full-node* ausfallen, könnten keine Transaktionen mehr bestätigt werden. Da es weder während der Entwicklungsphase, noch während der Studie zu einem einzigen Ausfall kam, ist dieser Nachteil als sehr klein einzuschätzen, weswegen auch keine weitere *full-node* zum System hinzugefügt wurde. Allerdings besteht der große Vorteil darin, dass Transaktionen sehr schnell bestätigt werden, da es keine weiteren *nodes* gibt, die um die Berechnung eines Blocks konkurrieren.

3 System Architektur

Vor allem aus Sicht der User-Experience stellt das einen großen Mehrwert dar, da ein User in wenigen Sekunden erfährt ob seine Transaktion erfolgreich durchgeführt wurde. Dies kann bei anderen Blockchains wie z.B. Bitcoin bis zu 10 Minuten dauern [Rap09], was im Kontext der Systemarchitektur nicht tragbar wäre.

Um letztendlich mit der Blockchain zu kommunizieren und ihr Potential in voller Gänze ausschöpfen zu können, werden auf diese sogenannte *Smart-Contracts* deployed. Im Rahmen der Systemarchitektur sind es zwei dedizierte Smart-Contracts (*Coffe-Coin* 3.2.2, *Beverage-List* 3.2.3), welche komplett unabhängig voneinander agieren.

Smart Contracts

Die Smart Contracts, welche auf die Blockchain deployed werden, tragen die Titel *Coffe-Coin* und *Beverage-List*. Diese decken zwei völlig unterschiedliche Aufgabenbereiche ab, weswegen sie unabhängig voneinander operieren und dadurch auch keinen Einfluss aufeinander ausüben.

Lediglich der *Beverage-List Contract* löst ein Event aus, sobald eine bestimmte Funktion dessen aufgerufen wird.

Die Ausführung (*call*) beider erfolgt jedoch stets von Seiten der *Tablet-App*. Diese ist auch die einzige Instanz, welche in Form von Transaktionen mit der Blockchain interagiert.

Tablet-App

Die *Tablet-App* ist eine mit React-Native [RN:] erstellte plattformübergreifende App, welche auf einem Android Tablet installiert ist. Die Hauptaufgabe der App ist es Funktionen der beiden Smart Contracts aufzurufen, in dem es die

benötigten Daten an den Smart Contract übergibt, um schlussendlich Transaktionen auszulösen.

Damit eine Kommunikation mit einem Smart Contract überhaupt zustande kommt, schickt die App einen Request an den Fileserver, welcher wiederum mit den angefragten Smart Contract Daten in Form einer Datei antwortet.

Event

Das Event beschreibt im Grunde die indirekte Kommunikation zwischen dem *Learner* und der *Tablet-App* mit dem Smart Contract *Beverage-List* als Mittelsmann. So wird jenes im Zuge eines Funktionsaufrufs des Smart Contracts, von Seiten der App ausgelöst, anschließend vom *Learner* detektiert und dessen Inhalt zum Erlernen des Kaffeetrinkverhaltens verwendet.

3.1.2 Workflow

Die unter Kap. 3.1.1 beschriebene Architektur wird im folgenden unter dem Gesichtspunkt des Workflows, also dem Zusammenspiel der einzelnen Komponenten und dem Gesamtablauf, näher betrachtet. Dabei beschreibt der Gesamtablauf die einzelnen Schritte, startend beim Setup der Komponenten, hin zum eigentlichen Durchlauf der einzelnen Softwareanwendungen, was letztlich im Erlernen des Kaffeetrinkverhaltens resultiert. Im Zuge dessen werden auch einzelne Algorithmen der Instanzen und Kommandos kurz erläutert, um ein besseres Verständnis für die Funktionsweise der Anwendungen zu bekommen.

Der Workflow lässt sich in zwei Phasen unterteilen. In der ersten werden die einzelnen Komponenten konfiguriert und gestartet. Die Zweite beschreibt den eigentlichen Ablauf und das Zusammenwirken der Instanzen.

3 System Architektur

Setup

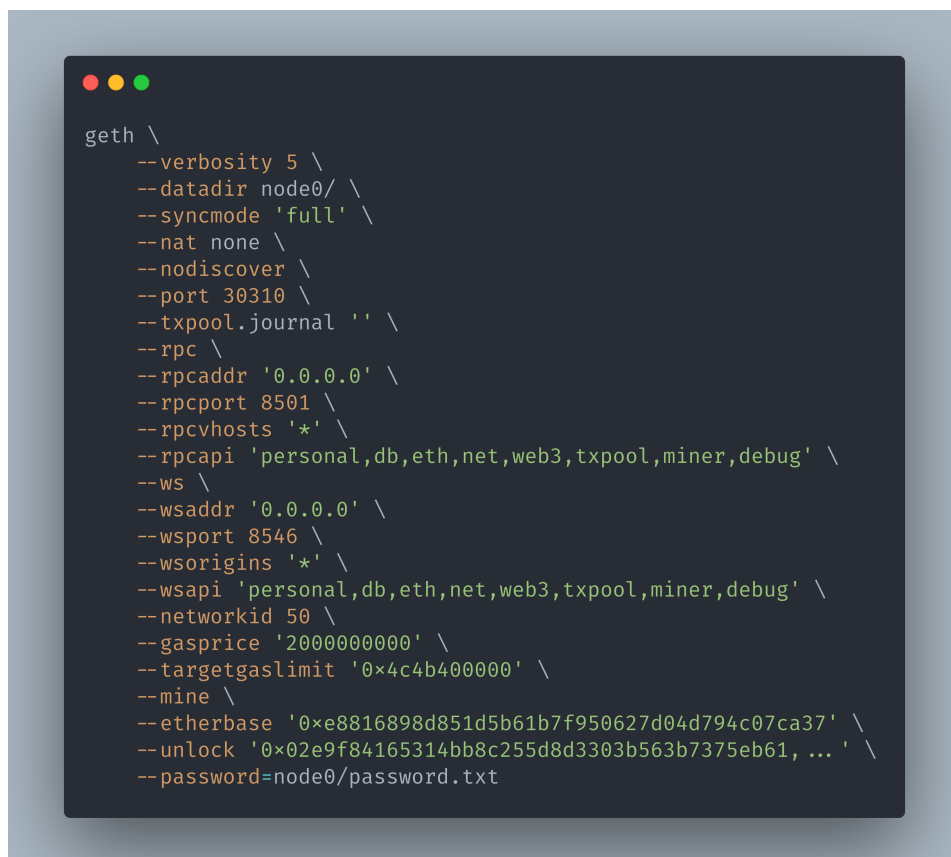
1. Blockchain
 - 1.1. erstellen & konfigurieren
 - 1.2. starten
2. Fileserver
 - 2.1. REST API starten
 - 2.2. UDP Broadcast starten
3. Smart Contracts
 - 3.1. deploy Beveragelist Smart Contract und sende JSON-File mit *ABI* und Adresse an Fileserver
 - 3.2. deploy CoffeeCoin Smart Contract und sende JSON-File mit *ABI* und Adresse an Fileserver
4. Learner
 - 4.1. Worker starten
 - 4.2. Watcher starten
5. Tablet App
 - 5.1. installieren
 - 5.2. starten

3.1 Überblick

Die Punkte 1. und 2. sowie 4. und 5. können auch parallel ausgeführt bzw. deren Reihenfolge vertauscht werden.

Im ersten Schritt muss die private Blockchain erstellt werden. Dabei müssen zuerst die benötigten Accounts generiert und daraufhin die Blockchain erzeugt werden. Sollte 1a) zu einem früheren Zeitpunkt bereits durchgeführt worden sein, kann dieser Punkt übersprungen und gleich mit 1b) begonnen werden. Sobald 1a) einmal durchgeführt wurde, kann die Blockchain gestartet werden.

Dies geschieht mit folgendem Befehl (vgl. 3.2):

A terminal window with a dark background and light-colored text. The window has three colored window control buttons (red, yellow, green) in the top-left corner. The text inside the terminal is a single line of a command to start a Geth node, with various options and their values listed on separate lines, separated by backslashes. The command starts with 'geth \' and ends with ' --password=node0/password.txt'.

```
geth \  
  --verbosity 5 \  
  --datadir node0/ \  
  --syncmode 'full' \  
  --nat none \  
  --nodiscover \  
  --port 30310 \  
  --txpool.journal '' \  
  --rpc \  
  --rpcaddr '0.0.0.0' \  
  --rpcport 8501 \  
  --rpcvhosts '*' \  
  --rpcapi 'personal,db,eth,net,web3,txpool,miner,debug' \  
  --ws \  
  --wsaddr '0.0.0.0' \  
  --wsport 8546 \  
  --wsorigins '*' \  
  --wsapi 'personal,db,eth,net,web3,txpool,miner,debug' \  
  --networkid 50 \  
  --gasprice '2000000000' \  
  --targetgaslimit '0x4c4b400000' \  
  --mine \  
  --etherbase '0xe8816898d851d5b61b7f950627d04d794c07ca37' \  
  --unlock '0x02e9f84165314bb8c255d8d3303b563b7375eb61, ...' \  
  --password=node0/password.txt
```

ABBILDUNG 3.2: Geth Kommandozeilenbefehl zum starten der Blockchain

3 System Architektur

Dieser Befehl setzt zum einen weitere Konfigurationsparameter der Blockchain. So wird z.B. festgelegt unter welcher IP-Adresse und Port (`--rpcaddr`, `--rpcport`, `--wsaddr`, `--wsport`) die Blockchain erreichbar ist und welche API-Befehle unter dieser Schnittstelle ausgeführt werden dürfen (`--rpcapi`, `--wsapi`).

Zum anderen wird zugleich eine *full node* gestartet (`--syncmode`), die durch das “Flag” `--mine` sofort zum “minen” beginnt.

Des Weiteren werden alle Accounts entsperrt, die unter (`--unlock`) gelistet sind und deren Passwörter in der angegebenen Textdatei bei (`--password`) hinterlegt sind.

Damit eine *Node* Daten speichern kann, muss ein Verzeichnis angegeben werden (`--datadir`), in welchem Dateien abgelegt werden können. Hier werden z.B. die Daten der Accounts oder auch die Textdatei mit den Passwörtern (`--password`) gespeichert.

Wurde die Blockchain in Betrieb genommen, wird im nächsten Schritt die REST API und der UDP Broadcast des Fileservers gestartet.

Daraufhin ist es möglich die beiden Smart Contracts zu deployen. Dabei wird jeweils für 3a) und 3b) das identische Skript mit unterschiedlichen Eingabeparametern ausgeführt. Dieses Skript liest zuerst die Datei des angegebenen Smart Contracts ein und erzeugt daraufhin die Binaries und die *ABI*, welche schlussendlich verwendet werden ein Go Bindingsfile zu generieren. Im Anschluss wird ein Go-Skript ausgeführt, welches auf Grundlage des Bindingsfiles den Smart Contract auf der Blockchain installiert und die zurückgelieferte Adresse, sowie die bereits bekannte *ABI* in ein JSON-File packt und an den Fileserver schickt.

Wurden die Smart-Contracts erfolgreich deployed, kann der Learner gestartet werden. Dieser “subscribed” sich im ersten Schritt auf den UDP Broadcast und extrahiert aus den Nachrichten die IP-Adresse der Blockchain und des Fileservers. Anschließend werden sowohl der *Worker* als auch der *Watcher* in Form von “Goroutines” (nebenläufige Kindprozesse) aktiviert. Der *Worker* iteriert über die Liste aller User und kontaktiert jeweils den Fileserver, ob bereits gelernte Daten für diesen User vorhanden sind. Sollte das Fall sein, initialisiert er damit die Parameter des Q-Learning Algorithmus des jeweiligen Users.

Der *Watcher* schickt ebenfalls eine Anfrage an den Fileserver und bekommt als Antwort die Daten der Smart Contracts. Daraufhin kann er sich mit der Blockchain verbinden und sich auf die Events des *Beveragelist* Smart Contracts subscriben.

Abschließend wird die App auf dem Tablet installiert, sollte sich diese noch nicht auf dem Tablet befinden und daraufhin gestartet. Die Initialisierung erfolgt hierbei nach dem selben Prinzip wie beim Learner. Zuerst wird der UDP Broadcast nach der Server Adresse abgefragt, mit welcher anschließend der Request an den Fileserver geschickt wird, um die benötigten Smart Contract Daten zu bekommen. Im Anschluss werden jene verwendet eine Verbindung zur Blockchain bzw. zu den Smart Contracts herzustellen.

Erfolgte eine fehlerlose Abarbeitung dieser Schritte, kann zum eigentlichen Workflow übergegangen werden.

3 System Architektur

Workflow

1. App
 - 1.1. User wählt Getränk aus
 - 1.2. *call* CoffeeCoin
 - 1.3. *call* Beveragelist
2. Smart Contracts
 - 2.1. Beveragelist *triggers* Event
3. Learner
 - 3.1. Watcher:
 - 3.1.1. detektiert Event
 - 3.1.2. extrahiert Daten aus Event
 - 3.1.3. “befüllt” Q-Learning Algorithmus mit den Event-Daten (*evaluate & predict*)
 - 3.2. Worker (periodisch alle 3h)
 - 3.2.1. “triggers” Q-Learning Algorithmus (*evaluate & predict*)
 - 3.2.2. sendet gelernte Daten an Fileserver

3.1 Überblick

Diese Auflistung beschreibt einen synchronen, erfolgreichen Durchlauf der Systemarchitektur - die Asynchronität des Workers 3b) außer Acht gelassen. Alternative Abläufe sowie Zustände die aus Fehlern resultieren, werden bei den einzelnen Komponenten nochmals genauer betrachtet.

Der Workflow wird durch den User angestoßen, indem dieser auf dem Tablet ein Getränk auswählt und eine Transaktion auslöst. Zuerst wird dabei der *Coffee-Coin* Contract aufgerufen und das ausgewählte Getränk bezahlt. Nachdem diese Transaktion erfolgreich bestätigt wurde, wird als nächstes der *Beveragelist* Contract ausgeführt. Die dabei aufgerufene Funktion des Smart Contracts verwendet die übergebenen Daten (Zeit, Getränk, Wochentag, Ethereum-Adresse) und löst damit ein Event aus.

Dieses Event wird vom *Watcher* detektiert und die Daten (Zeit, Getränk, Wochentag, Ethereum-Adresse) daraus extrahiert. Daraufhin wird die *Learn-Methode* des Q-Learning Algorithmus aufgerufen, bei der zuerst die vorherige *Prediction* evaluiert und basierend auf dem aktuellen Zustand eine neue *Prediction* gemacht wird.

Zu diesem synchronen Durchlauf führt der Worker am Ende jedes Timeslots (alle 3h) die *Learn-Methode* für jeden bekannten User aus. Das heißt es werden wie auch beim Watcher die *Predictions* des vorangegangenen Timeslots evaluiert, neue *Predictions* für den kommenden Timeslot erstellt und die gelernten Daten als Datei an den Fileserver gesendet.

3.2 Blockchain

Das folgende Kapitel erläutert im Detail den Setup der privaten Blockchain, sowie die beiden Smart Contracts, welche ebenso einen Teil der Systemarchitektur darstellen.

3.2.1 Genesis Block

Der große Vorteil einer privaten (Ethereum) Blockchain gegenüber einer öffentlichen, ist die Möglichkeit die Blockchain nach den eigenen Vorstellungen und Anwendungszwecken zu konfigurieren. So können, wie auch bei einer öffentlichen Blockchain, Smart Contracts erstellt und Transaktionen durchgeführt werden, allerdings ohne dabei wirkliches Ether zu besitzen. Denn eine Besonderheit eines sogenannten “Testnet’s” ist das Erzeugen von “privatem” Ether, welches Accounts zugeordnet und Transaktionen damit durchgeführt werden können [Dem12].

Die Konfiguration dessen erfolgt durch ein “genesis.json file” (3.3). Diese Datei ist die Grundlage für den *Genesis Block* der zu erstellenden Blockchain, welcher der erste Block in der Kette ist und somit auch keinen Vorgänger besitzt [Bas18].

Das in 3.3 abgebildete JSON Objekt zeigt die in der Systemarchitektur verwendete Datei einen solchen *Genesis Block* zu erzeugen. Nicht alle “properties” bedingen einer Erklärung, die essentiellen werden in Folge kurz erläutert [Bas18, 05]:

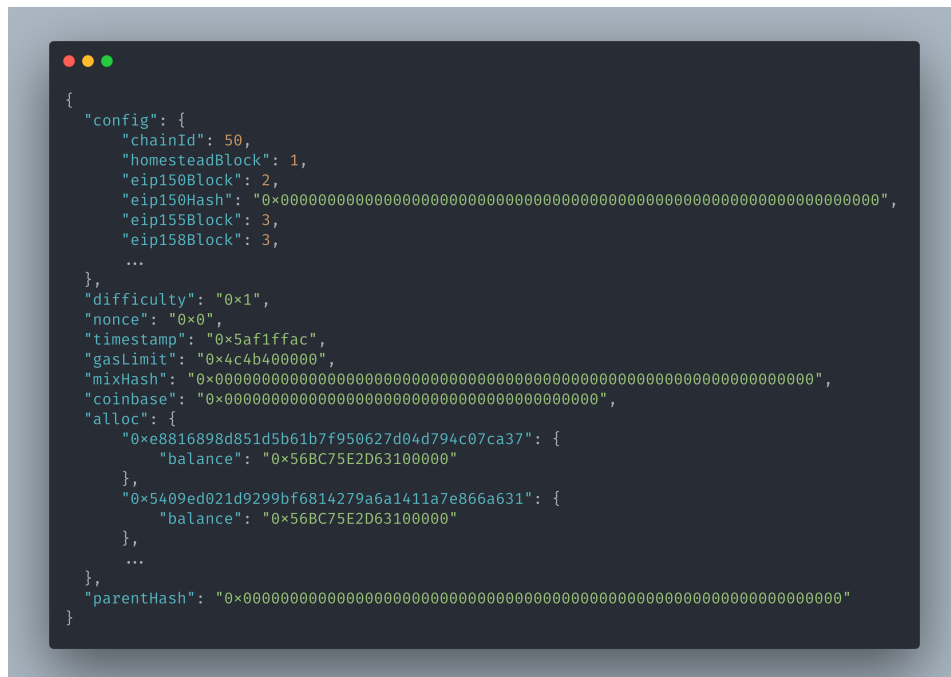


ABBILDUNG 3.3: JSON-File zur Generierung des Genesis Blocks

- chainId: ist eine einzigartige Id für die private Blockchain
- eip150Block/eip155Block/eip158Block: eip steht für “Ethereum Improvement Proposal”. Diese drei Community getriebenen “Proposals” beschreiben sog. “hard forks”, welche dafür sorgen sollen, Fehler im Protokoll zu beheben.
- homesteadBlock: Homestead ist die zweite “major version” von Ethereum, welche einige Änderungen an dem Protokoll vornahm
- difficulty: beschreibt die Schwierigkeitsstufe für einen Miner einen validen Block zu finden. Das heißt, je höher der Wert desto mehr Berechnungen müssen statisch durchgeführt werden und desto mehr Zeit wird benötigt, um eine Transaktion zu bestätigen. Im Falle einer privaten Blockchain ist es deshalb ratsam einen sehr niedrigen Wert zu wählen

3 System Architektur

- gasLimit: beschreibt das Limit für eine Transaktion wie viel an Gas verbraucht werden darf.
- alloc: hier können Accounts schon im Voraus mit “fake ether” befüllt werden.
- nonce: ist ein Zähler für die Anzahl der durchgeführten Transaktionen einer Adresse

Sobald die genesis.json Datei fertiggestellt ist, kann die Blockchain mit folgendem Befehl erstellt werden (vgl. 3.4):

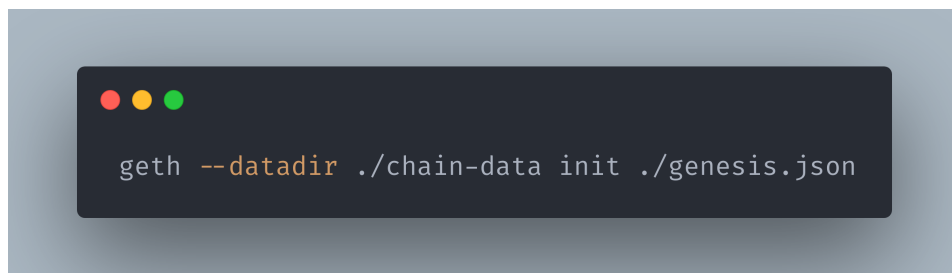


ABBILDUNG 3.4: Kommandozeilenbefehl zur Generierung des Genesis Blocks

Das Flag `--datadir` ist mit dem aus Abbildung 3.2 identisch. Das bedeutet das Verzeichnis welches hier im Befehl angegeben ist, muss beim Kommandozeilenbefehl in Abbildung 3.2 exakt gleich sein. Andernfalls ist es nicht möglich die Blockchain zu starten.

Nachdem der Setup der privaten Blockchain abgeschlossen ist, kann mit der Entwicklung eines Smart Contracts begonnen werden.[Abh09]

Die nächsten beiden Unterkapitel befassen sich mit den Smart Contracts, die im Zuge der Arbeit entwickelt wurden.

3.2.2 CoffeeCoin

Der Smart Contract *CoffeeCoin* stellt einen sogenannten *ERC-20 Token* mit gewissen Abwandlungen dar. Dabei ist ein Token im Grunde ein zusätzliches Zahlungsmittel zur eigentlichen Währung von Ethereum dem Ether. Das bedeutet Smart Contracts können somit eine eigenständige Währung abbilden. Für solche Smart Contracts gibt es mittlerweile einige Standardisierungen, die Vorschreiben welche Methoden und Datenstruktur ein Smart Contract zu implementieren hat. Der am weitesten verbreitete Standard ist der *ERC-20* [ERC25, Fra20, Ame20], bei welchem es folgende Funktionen und Events zu implementieren gilt (vgl. Abbildung 3.5):



```

contract ERC20Interface {
    function totalSupply() public view returns (uint);
    function balanceOf(address tokenOwner) public view returns (uint balance);
    function allowance(address tokenOwner, address spender) public view returns (uint
remaining);
    function transfer(address to, uint tokens) public returns (bool success);
    function approve(address spender, uint tokens) public returns (bool success);
    function transferFrom(address from, address to, uint tokens) public returns (bool success);

    event Transfer(address indexed from, address indexed to, uint tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}

```

ABBILDUNG 3.5: ERC-20 Interface

- `totalSupply`: Gesamtanzahl der existierenden Tokens
- `balanceOf`: Anzahl der Tokens eines bestimmten Users
- `allowance`: besagt wie viele Tokens eines bestimmten Users durch einen bestimmten “spender” abgehoben werden dürfen
- `transfer`: transferiert die Inputanzahl an Tokens des “senders” (Adresse welche diese Funktion aufgerufen hat) an die angegebene Adresse

3 System Architektur

- approve: Erlaubniserteilung an den “spender” die in *allowance* festgelegte Anzahl an Tokens abzubuchen
- transferFrom: transferiert die angegebene Anzahl von Tokens von der “from Adresse” zur “to Adresse”
- event Transfer: wird von den beiden *transfer* Funktionen ausgelöst
- event Approval: wird von der *approve* Funktion ausgelöst

Der eben erläuterte Token Standard, wurde im Falle des CoffeeCoin Smart Contracts um zusätzliche Funktionen und Datenstrukturen erweitert. Diese Erweiterungen stellen eine gesonderte Abstraktionsebene nach außen hin da, um die Kommunikation seitens der Tablet-App zu erleichtern und einfacher zu gestalten.

So werden schon beim Deployment die einzelnen Preise der Getränke und die Adresse, zu jener die Tokens beim Bezahlen eines Getränks überwiesen werden, gesetzt.

Dies ermöglicht die Implementierung folgender Funktionen (vgl. Abbildung 3.6):

A screenshot of a code editor window with a dark background and light-colored text. The code defines a Solidity contract interface named 'CoffeeCoinInterface'. It includes three functions: 'payCoffee()', 'payWater()', and 'payMate()', each returning a boolean 'success'. The code is color-coded: 'contract' is purple, 'function' is blue, and the rest is white. There are three colored circles (red, yellow, green) in the top left corner of the editor window.

```
contract CoffeeCoinInterface {  
    function payCoffee() public returns (bool success);  
    function payWater() public returns (bool success);  
    function payMate() public returns (bool success);  
    ...  
}
```

ABBILDUNG 3.6: CoffeeCoin Interface Auszug

Diese Funktionen bieten eine Abstraktionsebene zur ERC-20 Funktion *transferFrom*. Da bereits beim Deployment die Parameter für die Getränke und die Zieladresse gesetzt werden, benötigen diese Funktionen keine weiteren Daten von der Seiten der User (Tablet-App). Somit kann nach Auswahl des Getränks, die entsprechende Funktion aufgerufen werden, ohne sich dabei mit den Details der Transaktion beschäftigen zu müssen.

Des weiteren wird einem User, beim ersten Aufruf einer jener Funktionen (erste ausgelöste Transaktion des Users), eine festgelegte Anzahl an Tokens als “Startguthaben” überwiesen, sodass dieser stets über genügend Token verfügt und jederzeit Transaktionen durchführen kann.

Im Kontext der Problemstellung liegt die Zweckmäßigkeit des Smart Contracts grundsätzlich in der Bezahlung der Getränke in Form des Tokens. Dabei wird vor allem die Möglichkeit einer solchen Bezahlmethode aufgezeigt, weswegen Transaktionen lediglich auf exemplarischer Ebene durchgeführt werden. Das bedeutet, den Usern wird, wie gerade beschrieben, eine nahezu unendliche Menge an Tokens zugewiesen ohne eine Gegenleistung zu fordern.

3.2.3 Beverage-List

Im Gegensatz zur *CoffeeCoin* basiert der *Beveragelist* Contract auf keinem festgelegten Standard, sondern ist in voller Gänze an die Problemstellung angepasst.

Dessen Zweck besteht im Grunde darin eine Getränkeliste abzubilden, in welcher jede Getränketransaktion eines Users vorzufinden ist. Dabei wird pro Transaktion nicht nur das Getränk, sondern auch das aktuelle Datum inklusive Uhrzeit und der aktuelle Wochentag gespeichert. Diese Daten sollen es

3 System Architektur

dem Learner schlussendlich ermöglichen das Kaffeetrinkverhalten des Users zu erlernen.

Damit der *Learner* ohne großen Aufwand auf die Informationen zugreifen kann, löst der Smart Contract, mit den eben genannten Daten beinhaltend, ein Event aus, sobald seine Methode “setDrinkData” aufgerufen wird. Die Funktion hinterlegt die übergebenen Daten (Ethereum-Adresse, Zeit, Wochentag, Getränk) in der festgelegten Datenstruktur und löst zugleich das Event für den *Learner* aus.

Dieser Smart Contract umfasst noch weitere Funktionen, welche aber vor allem zu Testzwecken implementiert wurden und in der finalen Systemarchitektur keine Verwendung finden.

3.3 Learner

Als nächstes wird der *Learner* detailliert betrachtet. Dabei wird zuerst die Problemstellung hinsichtlich des Reinforcement-Learnings modelliert und im Anschluss der verwendete Algorithmus (Q-Learning) allgemein und im Kontext der Problemstellung erläutert.

3.3.1 Modellierung

Die Modellierung eines Lernproblems, im Hinblick auf einen Reinforcement-Algorithmus, erfolgt in der Regel stets nach der selben Systematik.

Zuerst werden der Zustandsraum (alle möglichen Zustände), alle Aktionen des *Agenten* und der *Reward* für ausgeführte Aktionen eruiert. Hier ist in diesem Fall ist der Agent der Learner, welcher das Kaffeetrinkverhalten der User zu erlernen versucht.

Die Modellierung sieht wie folgt aus:

Zustandsraum

- Wochentag: Montag, Dienstag, Mittwoch, Donnerstag, Freitag
- Timeslot:
 - 7-10 Uhr (T0)
 - 10-13 Uhr (T1)
 - 13-16 Uhr (T2)

3 System Architektur

- 16-19 Uhr (T3)
- 19-7 Uhr (T4)
- Kafee-Anzahl: $n * \text{Kaffee}$ (n =Anzahl pro Tag)

Aktionen

- Kaffee
- Nothing

Reward

- +1 bei richtiger Prediction
- -1 bei falscher Prediction

Daraus lässt sich folgender exemplarischer Zustand konstruieren:

`< Wochentag: Montag; Timeslot: 2; Kaffee-Anzahl: 3 >`

Das bedeutet konkret: an einem Montag wurden einschließlich des 2. Timeslots (13-16 Uhr) 3 Kaffee getrunken. Dieser Status gibt jedoch keinen Aufschluss darüber, welcher sein Vorgänger war und welcher sein Nachfolger sein wird. So können einerseits alle Kaffees nur in Timeslot 2 getrunken worden sein oder in jedem Timeslot (0,1,2) jeweils einer. Andererseits besteht auch die Möglichkeit, dass in Timeslot 2 nochmals ein Kaffee konsumiert wird oder eben erst in einem nachfolgendem Timeslot.

Dieses Lernproblem ist auch als “Multi-Armed Bandit Problem” bekannt und

wird in Kap. 3.3.2 erläutert.

Der erste Modellierungsansatz sah anstatt der Kaffee-Anzahl eine Getränkeanzahl vor, bei der zum Kaffee auch die Quantität der getrunkenen *Club Mate* und *Wasser* berücksichtigt werden sollten. Der Grund dafür liegt in dem erheblichen Einfluss des Konsums weiterer Getränke auf das Kaffeetrinkverhalten, wodurch mit solch einer granularen Modellierung genauere Predictions zu erwarten sind. Jedoch steigt somit auch die Anzahl der zu durchlaufenden Zustände und einhergehend die zu erlernenden optimalen Aktionen für die Zustandsübergänge. Letztlich resultiert dies in der längeren Trainingsphase des Algorithmus, was aber aufgrund des zeitlich begrenzten Rahmen dieser Arbeit nicht durchführbar war und deshalb der Ansatz mit lediglich der Kaffee-Anzahl gewählt wurde.

3.3.2 Q-Learning

Der Q-Learning Algorithmus fällt unter die Methodik des *TD-Learning* (vgl. 2.2.2). Der Algorithmus wartet aus diesem Grund nicht bis zur Terminierung einer Episode, um die Wertfunktion zu schätzen, sondern berechnet diese im nächsten Zeitschritt. Im Gegensatz zu anderen TD-Learning Algorithmen bemisst die Wertfunktion des Q-Learning nicht den aktuellen Zustand, sondern beurteilt die Aktion in diesem Zustand. Hierbei wird von der Aktion-Wert Funktion Q gesprochen [SB98].

Die Berechnung von Q erfolgt nach folgender Formel (3.1):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (3.1)$$

3 System Architektur

- $Q(s_t, a_t)$: beschreibt den aktuellen Q Wert bzw. Q-Value der Aktion a welche zum Zeitpunkt t im Zustand s ausgeführt wurde
- α : wird als Lernrate bezeichnet und gibt an in welchem Maße der alte Q-Value durch den Neuen überschrieben wird
- r_{t+1} : ist der Reward zum Zeitpunkt t+1, nachdem im Zustand s die Aktion a ausgeführt worden ist
- γ : ist der Diskontierungsfaktor und wird als Gewichtung für zukünftige Rewards verstanden
- $\max_t Q(s_{t+1}, a_t)$: ist das Maximum an Reward, der durch die Ausführung der Aktion a' im neuen Zustand s' erfolgt

Abgespeichert werden diese Q-Values in der sogenannten Q-Tabelle (vgl. Tabelle 3.1), hierbei gibt es ein 1:1 Mapping von jeweils einem Zustand auf jeweils eine Aktion.

TABELLE 3.1: Exemplarische Q-Tabelle

	a1	a2	a3
s1	0.1	-0.2	1.2
s2	0.4	-0.5	0.8
s3	1.1	-0.3	0.7
...

Die Anwendung und der Ablauf wird anhand des Pseudocodes in Algorithmus 1 abgebildet.

Zum Start wird der Q-Table initialisiert, indem jedes Feld eines Zustand-Aktions-Paars auf '0' gesetzt wird. Im Anschluss wird ein Durchlauf mit einer

Algorithmus 1 Q-learning algorithm

```

1: Initialize  $Q(s,a)$ , for all  $s \in S, a \in A(s)$  and  $Q(\text{terminal-state}, \cdot) = 0$ 
2: repeat(for each episode):
3:   Initialize  $s$ 
4:   repeat(for each episode):
5:     Choose  $A$  from  $s$  using policy derived from  $Q$  (e.g., greedy)
6:     Take action  $A$ , observe  $R, s'$ 
7:      $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$ 
8:      $s \leftarrow s'$ 
9:   until  $s$  is terminal

```

bestimmten Anzahl an Episoden gestartet. Hierbei wird zunächst der aktuelle Zustand initialisiert bzw. beobachtet und daraufhin eine weitere Schleife betreten, welche solange durchlaufen wird bis der Zustand s terminiert. Als nächstes wird eine Aktion a anhand der aktuellen Strategie ausgewählt. Diese Aktion wird ausgeführt und der resultierende Reward, sowie der neue Zustand werden detektiert. Anhand dieser Beobachtung wird mit der Formel (vgl. 3.1) der neue Q-Value berechnet und die Q-Tabelle upgedatet. Zum Schluss wird der neue Zustand s' als der aktuelle Zustand s gesetzt und eine neue Iteration beginnt [SB98, Mit04, Wei01].

Der gerade eben beschriebene Algorithmus findet sich in einer abgewandelten Form (vgl. 3.2) in der Implementierung des “Learners” wieder. So wird eine Aktion nicht immer nach dem greedy-Prinzip der Reward-Maximierung ausgewählt, sondern nach der Methodik des ε -greedy. Dabei beschreibt die Variable ε einen Wert zwischen ’0’ und ’1’ und gibt dadurch das Verhältnis zwischen Exploration und Exploitation an - in der Regel ist der Wert von ε eher klein.

So wird im Falle von MaxQ stets die Aktion a ausgeführt, welche in Zustand s den größten Reward verspricht - auch als greedy Strategie bekannt. Bei ε -

3 System Architektur

greedy jedoch, wird zu einem gewissen Anteil - definiert durch das ε - nicht immer die Aktion ausgewählt, die den größten Reward verspricht. Sondern ist wird zufällige Selektion einer Aktion durchgeführt, obgleich des erwarteten Rewards. Somit wird eine Erforschung des Zustandsraums durch den Lerna-
genten sichergestellt und eine Beschränkung auf lediglich bekannte Aktionen und Zustände vermieden.

Eine Besonderheit der Implementierung liegt in der Vakanz der Berücksichtigung von Folgezuständen bei der Berechnung des Q-Values.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} - Q(s_t, a_t)] \quad (3.2)$$

Aufgrund der Interdependenz der Zustände, ist das Verfolgen einer Strategie und das Einbeziehen der Folgezustände in die Berechnung der Q-Werte nicht notwendig. Der Fokus wird dadurch nur auf den Zustand als solches gelegt und die Annäherung des “wahren” Q-Wertes indessen einzig anhand des Rewards r_{t+1} durchgeführt.

Eine solche Form der Problemstellung wird auch als *n-armiges Bandit Problem* bezeichnet. Dieses ist ein klassisches Problem des Reinforcement Learning, welches das Dilemma zwischen *Exploration* und *Exploitation* aufdeckt. Es beschreibt das Spielen an mehreren (n) Glücksspielautomaten, deren einzige Aktion die Betätigung des Hebels ist. Das Ziel des Spielers ist die Maximierung seiner Belohnung (*Exploitation*) und wird deswegen nach jeder Aktion vor die Entscheidung gestellt, ob der Wechsel zu einem anderen Automat einen größeren Gewinn für ihn bedeutet (*Exploration*) oder ob er den aktuellen beibehält.

Die Analogie zum Lernproblem besteht in der Abbildung des unmittelbaren Rewards auf die gewählte Aktion. Eine Aktion wird nie angesichts von nach-

folgenden Aktionen bewertet, sondern einzig anhand der Belohnung nach ihrer Ausführung. Außerdem handelt es sich in beiden Fällen um ein nichtdeterministisches Lernproblem, bei welchem die Ausführung einer Aktion in einem bestimmten Zustand stets einen Reward bezüglich einer Wahrscheinlichkeitsverteilung nach sich zieht und somit eine hundertprozentige Vorhersage der zu erhaltenden Belohnung nicht möglich ist [SB98].

3.3.3 Lernprozess & Ablauf

Aufgrund der Modellierung eines Zustandes ergeben sich zwei Arten bei denen eine Änderung dessen hervorgerufen wird:

- durch eine zeitliche Komponente, bei der sich entweder der Timeslot oder der Tag ändert
- durch den User indem er eine Transaktion auslöst

Das ist in dieser Hinsicht von großer Bedeutung, da für diese Übergänge ein Feedback für den Agenten notwendig ist, um die optimale Aktion dafür zu erlernen.

So wird das Feedback einerseits von Seiten des Users in Form einer bestätigten Transaktion generiert, was bedeutet Aktion “Kaffee” wurde ausgeführt. Oder andererseits als Folge einer fehlenden Rückmeldung des Nutzers, welche “Nothing” als optimale Aktion impliziert.

Das heißt, erfolgt eine Zustandsänderung durch den User, so ist stets “Kaffee” die zu erlernende, beste Aktion. Wird eine Zustandsänderung durch einen Timeslotwechsel bewirkt, so ist das Feedback der Umgebung immerfort die Aktion “Nothing”.

3 System Architektur

Um dem Agenten bzw. dem Q-Learning Algorithmus stets das nötige Feedback zu geben und Zustandsübergänge herbeizuführen, werden im Learner die Komponenten *Watcher* und *Worker* verwendet.

Der *Watcher* ist dafür zuständig das Feedback des Users durch das Beveragelist-Event zu detektieren und die darin enthaltenen Daten dem Algorithmus zur Verfügung zu stellen.

Hierbei gilt es zu erwähnen, dass der Watcher, aufgrund der definierten Felder im Event, lediglich eine Feature Extraction durchführen muss und aus diesem Grund auf eine Feature Selection verzichtet werden kann [Zhe18].

Der *Worker* wird für die zeitliche Komponente verwendet, indem er am Ende jedes Timeslots aktiv wird, die Zustandsänderung durchführt und dem Agenten die Aktion “Nothing” als Feedback gibt.

Auf Basis dieser Feedbacks wird es dem Q-Learning Algorithmus ermöglicht die optimalen Aktionen zu erlernen. Hierbei wird die Prediction, also die aus Sicht des Agenten beste Aktion für den Zustandsübergang, anhand der Rückmeldung, sei es durch den Watcher oder Worker, evaluiert und für den nächsten Übergang eine neue Aktion eruiert.

Dieses Prinzip der Evaluierung und Vorhersage der Aktion kann im Kontext des Lernproblems jedoch nicht kontinuierlich angewendet werden. So bedingen bestimmte Zustände nur eine Evaluierung oder nur eine Vorhersage, hingegen nie beides. Der Grund hierfür liegt in der Abgeschlossenheit der Tage als Lernabschnitt, welche komplett unabhängig voneinander agieren.

Zudem werden die Uhrzeiten zwischen 19 und 7 Uhr (T4) nicht in der Zustandsraummodellierung berücksichtigt, da in diesem Zeitraum keine Aktion durch den User zu erwarten ist.

Dies hat für den *Worker* zur Folge, dass zum einen am Anfang jeden Tages

bzw. beim Wechsel von T4 auf T0, lediglich eine Vorhersage für den nächsten Zustandsübergang gemacht und zum anderen am Ende von T3 einzig die letzte Vorhersage evaluiert werden muss.

Dies lässt sich veranschaulicht folgendermaßen darstellen:

- Evaluierung & Vorhersage:
 - User führt Transaktion durch
 - Timeslot wechselt
- Evaluierung:
 - am Ende von T3
- Vorhersage:
 - am Anfang von T0

3.4 Tablet-App

Das folgende Unterkapitel befasst sich mit dem Aufbau und der Funktionsweise der App und schildert zudem den verwendeten Algorithmus, welcher die Getränke- und Userdaten auf die Blockchain schreibt.

Die entwickelte App basiert auf dem Framework “React Native” [RN:], dieses erlaubt es mit einer einzigen Codebasis Apps für unterschiedliche Plattformen (z.B. iOS, Android) zu entwickeln. Der wesentliche Vorteil allerdings liegt in der Verfügbarkeit einer offiziellen Library, mit der es erst möglich ist, eine Verbindung zur Blockchain bzw. den Smart Contracts herzustellen. Jene Library (Web3.js) wurde von Ethereum entwickelt, um sog. DApp’s (“decentralized apps”) [DAP] auf Basis von Javascript erstellen zu können.

So ist die Entwicklung einer DApp in einer nativen Programmiersprache (Java/Kotlin/Swift) bisher nur mit “third-party” Libraries möglich, weswegen die Umsetzung letztlich mit ReactNative erfolgte.

3.4.1 Interface

Eine essentielle Eigenschaft von ReactNative ist der komponentenbasierte Ansatz. Dabei setzt sich eine App aus vielen einzelnen Komponenten zusammen, welche jeweils einen dedizierten Aufgabenbereich abdecken.

Im Falle der entwickelten App existieren jeweils zwei “page components”, die wiederum mehrere kleine Komponenten in sich vereinen. Da es aber den Rahmen dieser Arbeit sprengen würde auf jede einzelne Komponente und deren Funktionsweise einzugehen, werden nur die Hauptkomponenten anhand ihrer Funktion und Bedienung geschildert.

Wird die App gestartet und es besteht eine Verbindung zum Internet, so findet der User folgende Startseite (vgl. 3.7) vor.

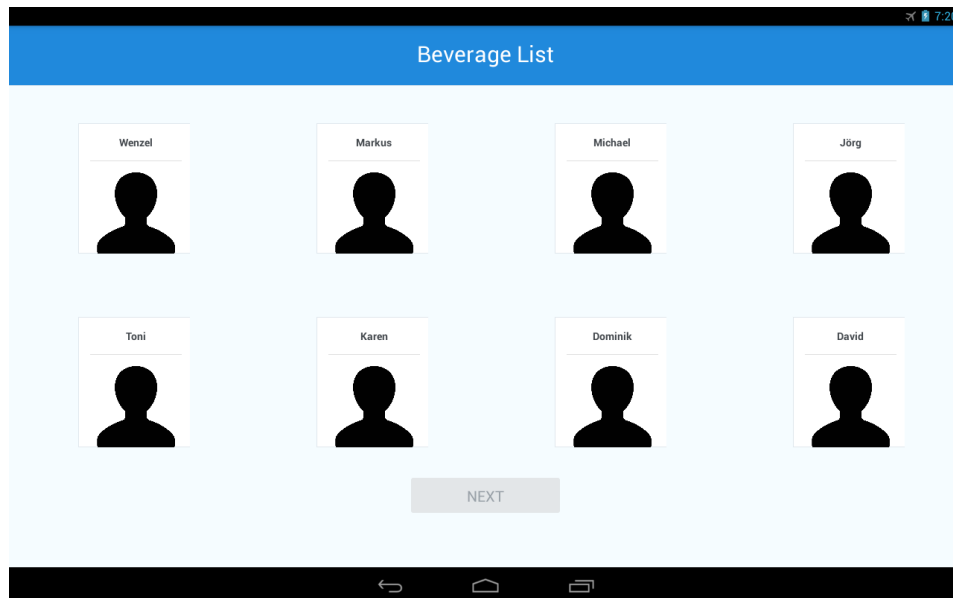


ABBILDUNG 3.7: Mitarbeiter Page: kein Mitarbeiter ausgewählt

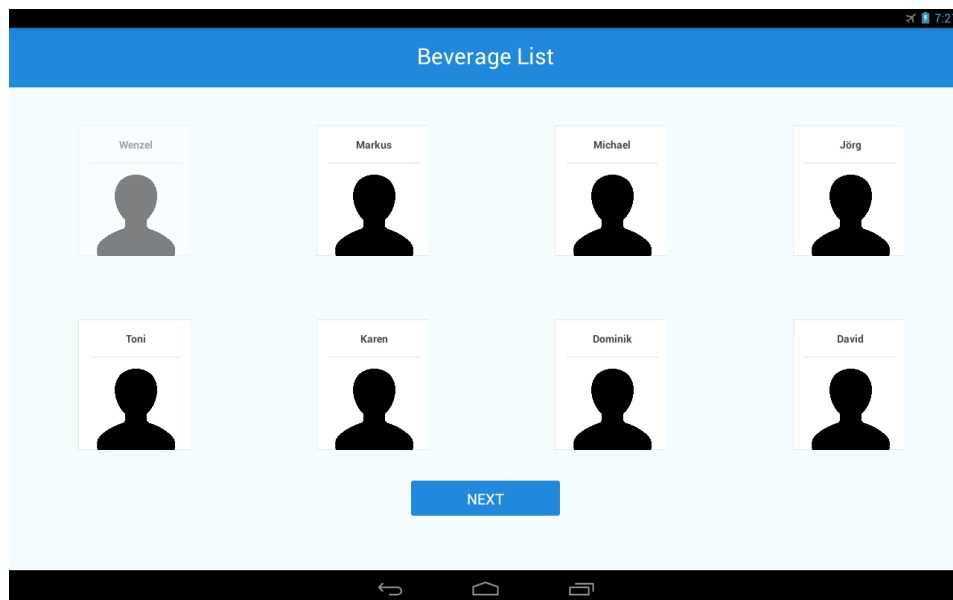


ABBILDUNG 3.8: Mitarbeiter Page: Mitarbeiter ausgewählt

3 System Architektur

Hier kann der User seinen Avatar selektieren und deselektieren. Ist ein Avatar ausgewählt wird der “NEXT” Button aktiviert (vgl. Abbildung 3.8) und durch dessen Betätigung gelangt der User zur nächsten Seite (vgl. Abbildung 3.9):

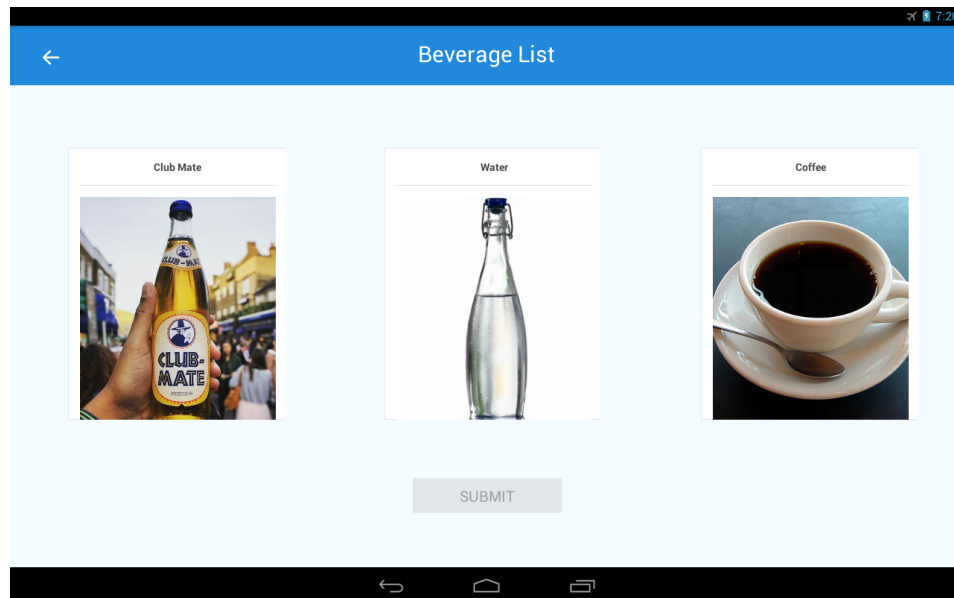


ABBILDUNG 3.9: Drinks Page: kein Getränk ausgewählt

Hier besteht eine Auswahl aus folgenden Getränken: Club Mate, Wasser und Kaffee. Ausgewählt kann jedoch immer nur eines werden (vgl. Abbildung 3.12). Das Prinzip der Selektion und Deselektion ist identisch mit dem der vorherigen Seite. So wird der “SUBMIT” Button aktiv, sobald ein Getränk ausgewählt ist und inaktiv wenn das Getränk wieder deselektiert wird (vgl. Abbildung 3.9).

Wird schließlich der “SUBMIT” Button gedrückt und die Transaktion als erfolgreich bestätigt erscheint folgendes Overlay (vgl. Abbildung 3.11):

Nach 4 Sekunden wird dieses Overlay wieder ausgeblendet und automatisch zur Startseite (vgl. Abbildung 3.7) navigiert, wodurch der Workflow von neuem startet.

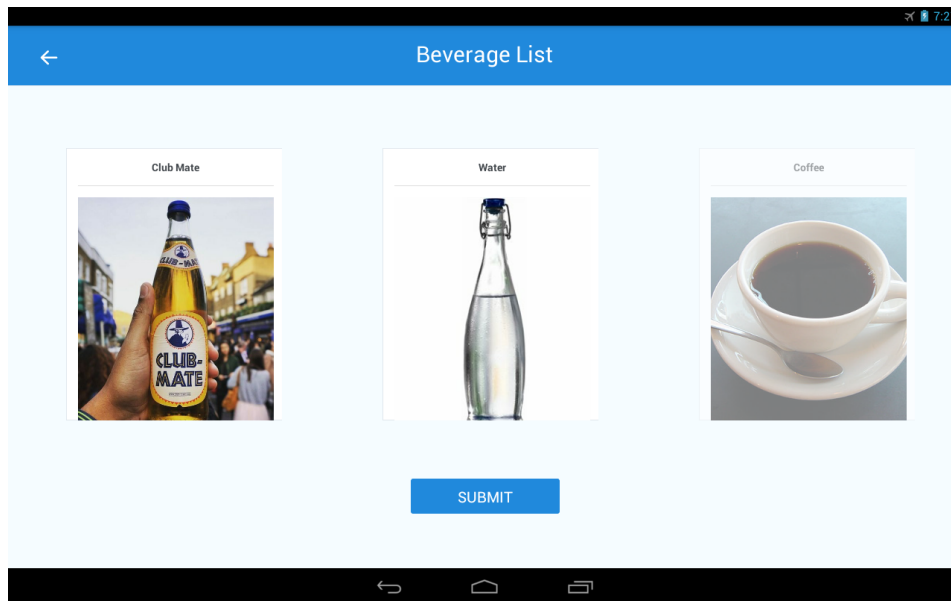


ABBILDUNG 3.10: Drinks Page: Getränk ausgewählt

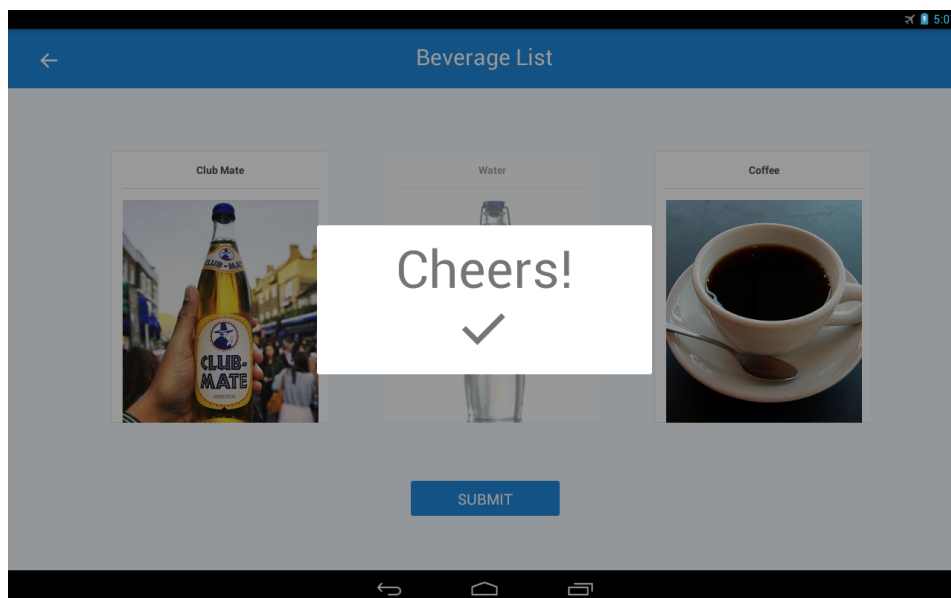


ABBILDUNG 3.11: Overlay: Bestätigung der Transaktion

Aufgrund der Tatsache, dass die Verbindung zum Uni-Netzwerk (eduroam) nur über einen Workaround hergestellt werden konnte, bei dem die Verbin-

3 System Architektur

dung zum Netzwerk trotz dessen nach einer unbestimmten Zeit immer wieder abgebrochen ist, wurde eine weitere Komponente entwickelt.

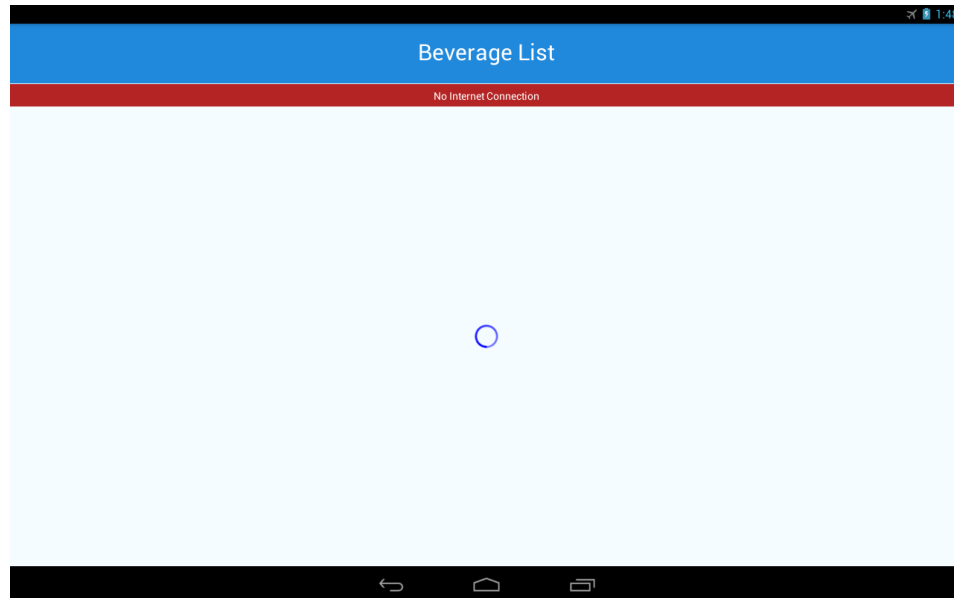


ABBILDUNG 3.12: No Internet: keine Internetverbindung verfügbar

Diese wird sofort eingeblendet, sobald die Verbindung zum Internet unterbrochen ist, allerdings nur wenn sich der User auf der Startseite befindet.

Wird die Seite mit den Getränken angezeigt, so wird dem User die Möglichkeit gegeben seine Transaktion abzuschließen. Da in diesem Moment jedoch keine Verbindung zur Blockchain hergestellt werden kann, werden die Transaktionsdaten zwischengespeichert und diese durchgeführt, sobald wieder eine Verbindung zum Netzwerk besteht (vgl. Kap. 3.4.2).

Die Loading Animation sowie das “No Internet Connection” Label werden wieder ausgeblendet, sobald die App eine erneute Verbindung zum Uni-Netzwerk detektiert.

3.4.2 Internal Workflow

Basierend auf der Problematik eines unerwarteten Verbindungsabbruch und dem einhergehenden Verlust von essentiellen Lerndaten, wird im folgenden der implementierte Algorithmus geschildert, der dieser Komplikation entgegenwirkt.

1. Initialisierungsphase:
 - 1.1. “GET” Smart Contract Daten von Fileserver
 - 1.2. Initialisierung Web3.js
 - 1.3. Wiederhole für jedes Transaktionsfile:
 - 1.3.1. Transaktion durchführen
 - 1.3.2. bei “Success” Transaktionsdatei löschen
2. Warten auf Usereingabe:
 - 2.1. User ausgewählt → Ethereum-Adresse
 - 2.2. Getränk ausgewählt → Getränk
 - 2.3. Submit → (Getränk & Ethereum-Adresse)
3. Wiederhole für jedes User-Transaktionsfile:
 - 3.1. Transaktion durchführen
 - 3.2. bei “Success” Transaktionsdatei löschen

3 System Architektur

4. Generierung Transaktionsdaten:

- Gas Estimate, Datum (inkl. Zeit), Wochentag
- Ethereum-Adresse & Getränk aus 2.3

5. Transaktion starten:

5.1. Erstellung der Transaktionsdatei

5.2. Transaktion durchführen

5.3. bei “Success” Transaktionsdatei löschen

5.4. gehe zu 1.

Der interne Workflow beginnt mit der Initialisierungsphase, hierbei werden zuerst die Smart Contract Files vom Fileserver runtergeladen und mit den beinhaltenden Daten das Web3.js Modul initialisiert. Somit besteht eine Verbindung zu den Smart Contracts und es kann im Anschluss über alle vorhandenen Transaktion Files iteriert werden.

Indessen wird zuerst der CoffeeCoin Contract und dann der Beveragelist Contract aufgerufen. Wenn beide ihre Transaktionen bestätigt haben, wird die Datei gelöscht, andernfalls bleibt sie bestehen. Dies geschieht noch bevor der User das Interface zu Gesicht bekommt.

Nachdem 1.) abgeschlossen ist, wird auf die Eingabe des Users gewartet. Wählt dieser einen Avatar aus und bestätigt mit “NEXT”, wird seine hinterlegte *Ethereum-Adresse* temporär gespeichert. Wird dann im Anschluss ein Getränk selektiert und “SUBMIT” gedrückt, werden die Ethereum-Adresse und das Getränk an das interne Blockchain-Modul weitergereicht und die Transaktion

gestartet.

Dabei wird erneut über die Transaktionsdaten iteriert, allerdings nur über die des Users. Damit soll stets die richtige Reihenfolge der konsumierten Getränke sichergestellt werden, da dies ansonsten auf Seiten des Q-Learning Algorithmus zu falschen Lerneffekten führen würde.

Daraufhin kann mit der Generierung der fehlenden Transaktionsdaten angefangen werden. Zuerst wird ein *Gas Estimate* für beide Smart Contract Transaktionen durchgeführt. Dies ist, wie der Name bereits impliziert, eine grobe Schätzung darüber, wie viel Gas beim Funktionsaufruf eines Smart Contracts benötigt wird.

Abschließend wird das Datum (inkl. Zeit in Sekunden) und der Wochentag ermittelt und alle benötigten Daten an die jeweiligen Smart Contract Funktionsaufrufe übergeben. Im Falle des Smart Contracts Beveragelist sind es folgende Parameter: *Gas-Estimate*, *Datum*, *Wochentag*, *Ethereum-Adresse*, *Getränk*.

Bei CoffeeCoin wird anhand des Getränks die entsprechende Smart Contract Funktion ausgewählt und lediglich das *Gas-Estimate* und die *Ethereum-Adresse* des Users übergeben.

Bevor jedoch beide Transaktionen durchgeführt werden, wird ein Transaktionsfile mit dem Namen “«ethereum-adresse»-«datum».json” und den Daten *Datum*, *Wochentag*, *Ethereum-Adresse*, *Getränk* gespeichert. Sogleich werden beide Funktionsaufrufe getätigt und bei erfolgreicher Bestätigung beider Transaktionen, das gerade erstellte File wieder gelöscht. Somit soll sichergestellt werden, dass keine Transaktionen aufgrund von Verbindungsabbrüchen verloren gehen.

Anschließend erfolgt ein Reload der App und der Ablauf beginnt wieder bei 1.

4 Studie

4.1 Simulation

Die Simulation ist die Grundlage der Evaluation der durchgeführten Studie. Sie soll in erster Linie zeigen, dass die Implementierung des Q-Learning Algorithmus fehlerfrei funktioniert und es für einen Agenten damit möglich ist das Verhalten eines Nutzers zu erlernen.

Dazu bedarf es allerdings Testdaten welche das Kaffeetrinkverhalten eines Users repräsentieren. Im “Simulator” wird es wie folgt umgesetzt.

Abstrakt gesehen wird ein virtueller User anhand von Zufallszahlen einer Normalverteilung simuliert.

Genauer betrachtet werden die Zeiten, an denen dieser virtuelle User einen Kaffee trinkt, basierend auf der Ziehung von Zahlen aus einer Normalverteilung erzeugt.

Dabei wird pro Tag (Montag-Freitag) und in jedem Timeslot (T0-T4) unter Berücksichtigung einer vordefinierten Abweichung und Erwartungswert (siehe 4.1), die Anzahl der getrunkenen Kaffees ermittelt. Für jeden Kaffee wird zufällig eine Uhrzeit gezogen, welche sich in dem Zeitraum des Timeslots befindet. Somit wird ein Trainingsset für den Agenten erstellt, welches das Verhalten einer realen Person in einem Zeitraum von einer Woche abbildet.

TABELLE 4.1: Normalverteilung Tabelle

Timeslot	Erwartungswert	Abweichung
0	1,0	1,0
1	1,5	0,2
2	1,5	0,01
3	0,2	0,5
4	0,0	0,0

Der Pseudocode des Simulators sieht folgendermaßen aus (vgl. Algorithmus 2):

Algorithmus 2 Simulation algorithm

```

1: weeks  $\leftarrow$  x                                     //x = Anzahl der Episoden
2: for weeks do
3:   trainSet  $\leftarrow$  generateTrainingsSet()
4:   for  $1 \leq \textit{day} \leq 6$  do                         //Montag-Freitag
5:     for  $7 \leq \textit{ts} < 19$  do                         //Timeslot 0-4
6:       times  $\leftarrow$  trainSet[day][ts]
7:       for all time  $\in$  times do
8:         learn(time)
9:       end for
10:    end for
11:  end for
12: end for=0

```

Zuerst wird eine bestimmte Anzahl an Episoden bzw. Wochen festgelegt. In der Simulation waren es 250. Daraufhin wird pro Woche ein Trainingsset mit einem neuen Seed erzeugt, mit welchem der Agent trainiert wird. Durch das Setzen eines neuen Seeds wird vermieden, dass der Agent stets mit dem selben (Wochen) Trainingsset konfrontiert wird. Stattdessen wird mit jeder neuen Episode ein Trainingsset erzeugt, welches komplett neue Uhrzeiten beinhaltet, die jedoch nach der selben Normalverteilung gezogen worden sind.

Pro Episode wird zuerst über die Tage (Montag-Freitag), dann über den Zeitraum der Timeslots (7 Uhr - 18 Uhr) und schließlich über die Zeiten an denen der virtuelle User einen Kaffee trinkt iteriert und der Agent letztendlich trainiert.

4.2 Testphase

Die Dauer der Testphase betrug 3,5 Wochen und umfasste 8 Teilnehmer des Lehrstuhls. Der Versuchsaufbau entsprach dem der Systemarchitektur aus Kapitel 3.

Wurde ein Getränk durch einen Mitarbeiter in der App ausgewählt und bestätigt, erfolgte eine Transaktion auf der Blockchain. Jene wurde wiederum vom Learner detektiert, gefiltert und die Daten zur Modellierung des Kaffeetrinkverhaltens verwendet. Zudem wurde jede Prediction und dessen Evaluierung geloggt, sowie in einer Datei abgespeichert. Unter Hilfenahme dieser Log-Datei erfolgte schließlich auch die Auswertung der Ergebnisse.

Bevor jedoch auf die Evaluierung eingegangen wird, sollen zunächst die Probleme und Hindernisse geschildert werden, die während der Testphase auftraten und die Laufzeit auf lediglich 3,5 Wochen reduzierten.

Das Hauptproblem war die veraltete Hardware des Tablets, welche nur noch ein Update auf eine 7 Jahre alte Android Version (4.2.1) zuließ. Mit dieser Version war es nicht möglich, eine direkte Verbindung zum Universitätsnetzwerk per WLAN herzustellen, da das benötigte Zertifikat diese Version nicht unterstützt. Eine Verbindung zum Netzwerk ist aber notwendig, um mit der

4 Studie

Blockchain zu kommunizieren bzw. eine Transaktion auslösen zu können.

Durch einen Workaround (freies WLAN + VPN-Client) konnte das Tablet mit dem Universitätsnetzwerk verbunden werden. Trotz allem sorgte diese Konstellation für viele unerwartete Verbindungsabbrüche, was dazu führte, dass einige Transaktionsdaten verloren gingen.

Um den Verbindungsabbrüchen entgegenzuwirken wurde daraufhin ein Mechanismus implementiert (vgl. 3.4.2), welcher zwar zu einer Verbesserung und Erhaltung der Daten beitrug, das Kernproblem allerdings nicht lösen konnte. Denn ein Verbindungsabbruch führte oftmals dazu, dass das Blockchain-Modul (Ethereum Library) der App abstürzte und die App manuell geschlossen und neu gestartet werden musste, damit wiederholt eine Verbindung zur Blockchain hergestellt werden konnte.

Erst durch den schlussendlichen Setup (Verbindung über einen Hotspot eines Laptop) konnten die Abstürze auf ein Minimum reduziert werden.

Ein weiteres Hindernis welches während der Testphase auftrat, war das Fehlen (Urlaub) einiger Mitarbeiter über den Zeitraum von 1-2 Wochen hinweg. Dadurch wurden lediglich zwei Datensätze erzeugt, welche als Grundlage für die Evaluation verwendet werden konnten.

4.3 Evaluation

Simulation 1

Bei der ersten Simulation (vgl. Abbildung 4.1) ist eine Konvergenz hin zu einer 100 % “success rate” nach ca. 170 Episoden bzw. Wochen zu erkennen. Dies bedeutet, der Agent hat nach dieser Anzahl an Durchläufen das Kaffeetrinkverhalten des Users in voller Gänze erlernt.

Die anfänglichen großen Sprünge, v.a. zwischen den Episoden 5 und 15, sowie 50 und 60, sind charakteristisch für einen Reinforcement Algorithmus. So ist anfänglich die Rate für die Exploration in der Regel sehr hoch, was oftmals zu vielen falschen Predictions führt, da die Aktionen zufällig ausgewählt werden. Mit der Zeit wird diese Rate aber nach und nach verringert und die Sprünge werden weniger.

Ein weiteres Merkmal des Graphs ist die logarithmische Trendlinie. Diese ist zum einen auch auf die Reduktion der Exploration zurückzuführen. Zum anderen ist sie dem Lernerfolg des Agenten geschuldet, welcher allerdings erst durch die anfängliche Exploration des Zustandsraums ermöglicht wird.

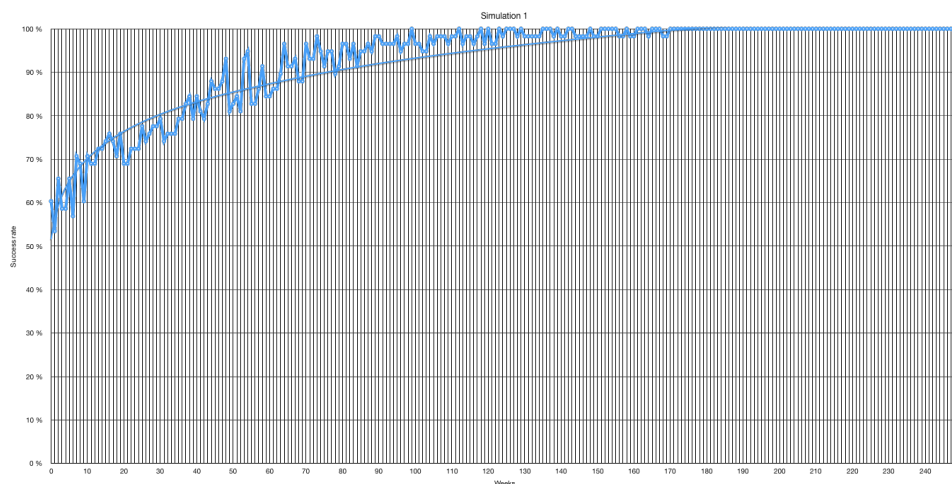


ABBILDUNG 4.1: Simulation 1: Lernkurve

4 Studie

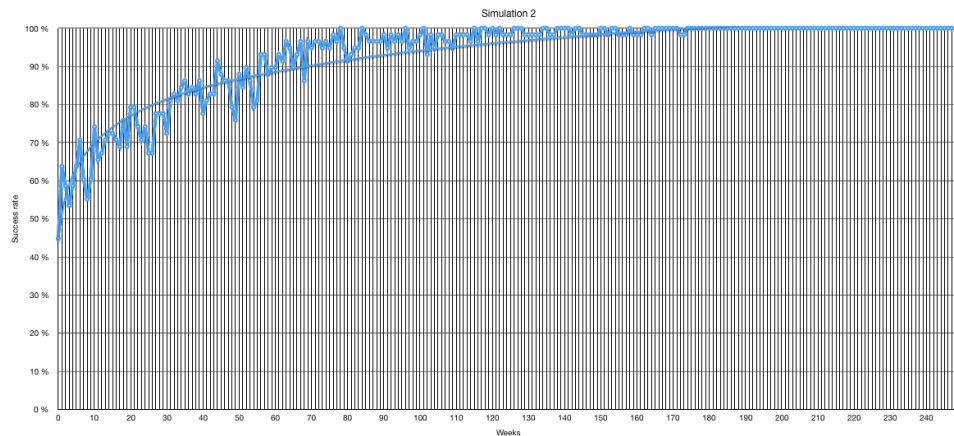


ABBILDUNG 4.2: Simulation 2: Lernkurve

Simulation 2

Der Graph der 2. Simulation (vgl. Abbildung 4.2) ist dem der Ersten äußerst ähnlich. So nähert sich dieser dem Optimum auch logarithmisch an und erreicht eine 100% “success rate” bei ca. 170 Episoden.

Aufgrund des sehr hohen Grads der Ähnlichkeit beider Graphen, wird hier auf eine weitere Interpretationen des Verlaufs verzichtet.

So sollen lediglich die Ergebnisse der ersten Simulation nochmals untermauert und die einwandfrei Funktionalität der Implementierung des Q-Learning Algorithmus gezeigt werden.

Testphase

Unter Berücksichtigung der in 4.2 geschilderten Probleme, gilt der Hinweis, dass während der Testphase nur sehr kleine Datensätze für die Evaluation erzeugt wurden. Dadurch erfolgt die Beantwortung der Forschungsfrage nur anhand von abgeleiteten Tendenzen aus den Datensätzen und lässt einen vollständigen Beweis außen vor.

Aus den fünf vorhandenen Datensätzen wurden jeweils zwei zur Auswertung herangezogen, bei denen am häufigsten Feedback durch den User gegeben wur-

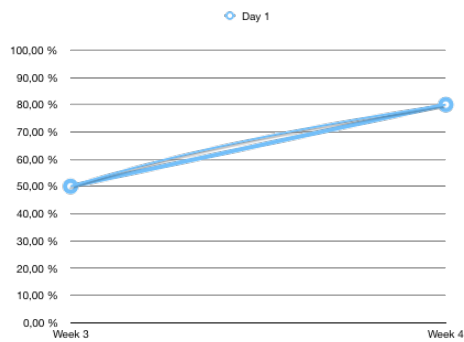


ABBILDUNG 4.3: User 1:
Montag

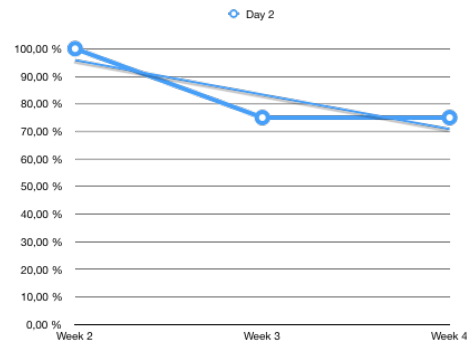


ABBILDUNG 4.4: User 1:
Dienstag

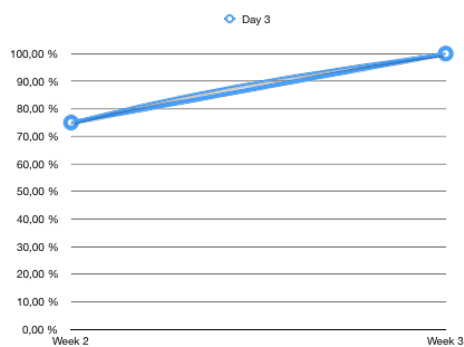


ABBILDUNG 4.5: User 1:
Mittwoch

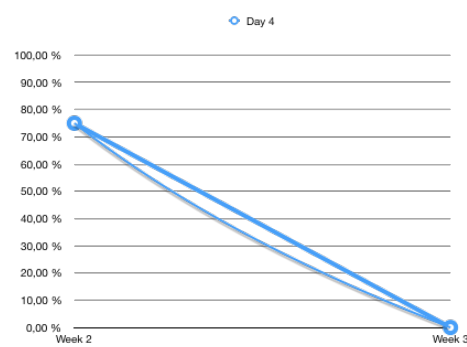


ABBILDUNG 4.6: User 1:
Donnerstag

de.

User 1

User 1 wurde erst in der zweiten Woche aktiv, weswegen die Teilnahmedauer nur 2,5 Wochen betrug. Betrachtet man den Graphen des Wochenüberblicks (vgl. Abbildung 4.8), startet dieser bei einer Erfolgsrate von 75%, fällt in der Woche 3 auf 60% ab und steigt in Woche 4 auf 77,8%. Was daraufhin deutet, dass der Agent aus der Explorationsphase in Woche 3 neues Wissen schöpfen und in Woche 4 anwenden konnte.

In Anbetracht der Tagesgraphen, ist im Falle von 3 Tagen (Montag, Mittwoch,

4 Studie

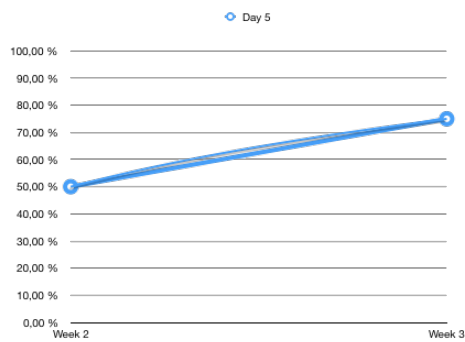


ABBILDUNG 4.7: User 1:
Freitag

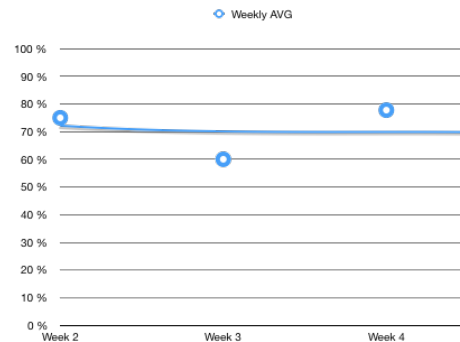


ABBILDUNG 4.8: User 1:
Wochenüberlick

Freitag; vgl. Abb. 4.3, 4.5 und 4.7) ein Anstieg der Erfolgsrate um ca. 30 % zu erkennen.

Der enorme Einbruch an Tag 4 (vgl. Abbildung 4.6) von 75 % auf 0 %, schließt auf ein zur Vorwoche stark verändertes Verhalten des Users. Auch eine Explorationsphase wäre theoretisch denkbar, ist statistisch aber eher unwahrscheinlich.

Trotz der Reduktion der Prediction accuracy an Tag 2 (vgl. Abbildung 4.4), liegt die Erfolgsrate immer noch bei 70 %, was definitiv ein Indiz für den Lernerfolg des Agenten ist.

User 2

User 2 war drei Wochen aktiv und startete am 4. Tag der ersten Woche. Beim Wochenüberblick (vgl. 4.14) ist in den Wochen 1-3 zuerst ein Anstieg von 55 % auf 68 % zu erkennen, welchem anschließend ein Abfall auf eine Erfolgsrate von 44 % folgt.

Bis auf den dritten Tag ist in allen anderen Fällen (vgl. Abbildung Abb. 4.9, 4.10, 4.12 und 4.13) mindestens einmal ein Anstieg der Erfolgsrate im Vergleich zur Vorwoche zu beobachten. Gerade der Graph des dritten Tages (vgl. Abbildung 4.11), bei welchem eine Stagnation über alle 3 Wochen hinweg zu erkennen ist, deutet einerseits auf einen gewissen Lernerfolg hin, zeugt aber auch andererseits von einer fehlenden Exploration des Agenten.

Es lässt sich sowohl an den Tagesgraphen, als auch am Wochengraph ein Trend hin zum Lernerfolg des Agenten beobachten. So ist die “prediction accuracy” in keinem Tag bei 0 % und es ist stets ein Anstieg oder zumindest eine Stagnation zu beobachten, was letztendlich auf einen Lernerfolg des Agenten hindeutet.

In Anbetracht beider Simulationsergebnisse ist der Lernerfolg bei dieser Problemstellung von logarithmischer Natur. Sowohl der Auswertungsgraph von User 1 (vgl. Abbildung 4.8) als auch von User 2 (vgl. Abbildung 4.14) teilen diese Charakteristik und zeigen eine Tendenz hin zu einer steigenden Erfolgsrate. Dies würde vorerst die These bestätigen, dass sich Machine Learning und Blockchain verbinden lässt, bzw. das Lernen anhand von Daten einer Blockchain möglich ist.

Jedoch muss dies gewissermaßen auch relativiert werden, da die Dauer der Studie sehr kurz war. Die Ergebnisse können eben nur eine Tendenz geben, inwiefern bei einer längeren Dauer der Testphase der Lernerfolg im selben Maße ausfallen würde, wie im Falle der Simulation.

4 Studie

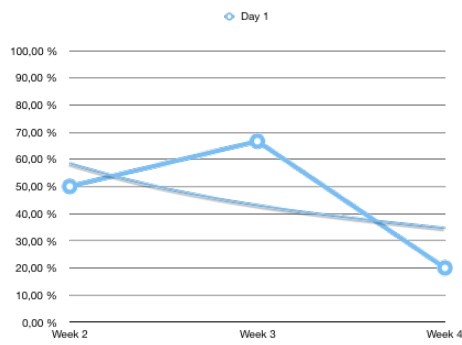


ABBILDUNG 4.9: User 2:
Montag

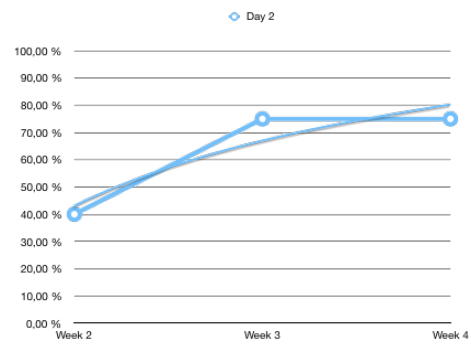


ABBILDUNG 4.10: User 2:
Dienstag

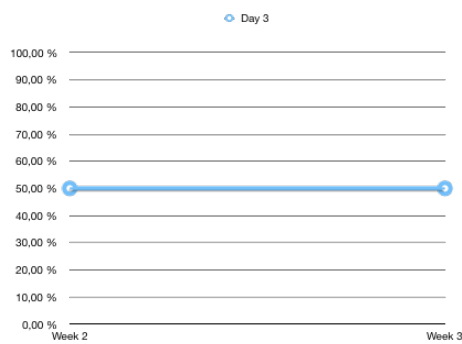


ABBILDUNG 4.11: User 2:
Mittwoch

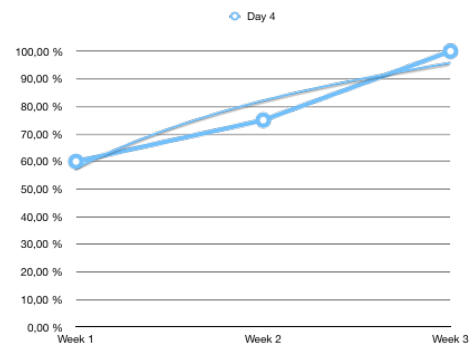


ABBILDUNG 4.12: User 2:
Donnerstag

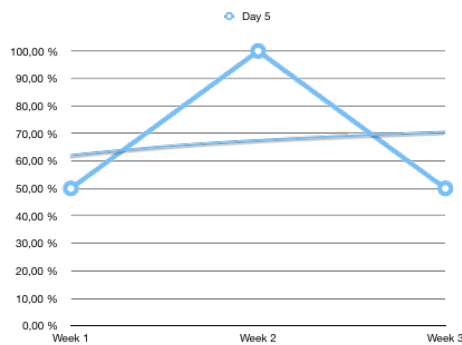


ABBILDUNG 4.13: User 2:
Freitag

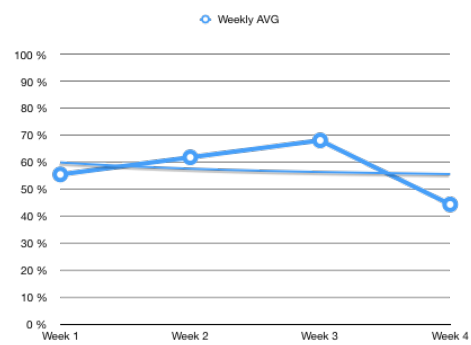


ABBILDUNG 4.14: User 2:
Wochenüberlick

Schlussendlich bedarf es einer weitaus längeren Studiendauer, um die Forschungsfrage in voller Gänze zu erörtern.

5 Zusammenfassung

Dieses Kapitel umfasst einerseits das Fazit, in welchem die Kernaspekte der Arbeit nochmalig geschildert werden und die Forschungsfrage beantwortet wird. Andererseits gibt es einen Ausblick bzw. thematisiert mögliche Weiterführungen der Forschungsfrage.

5.1 Fazit

Diese Arbeit befasst sich zum Einstieg mit der Kontradiktion zwischen dem Trend der Personalisierung, der in so gut wie allen Bereichen der digitalen Welt Einzug hält, und der einhergehenden Frage nach Datenschutz sowie Transparenz, was mit den gesammelten Userdaten passiert und auf welche Weise diese Personalisierung entsteht.

Abhilfe soll dabei die Kombination aus Machine learning und Blockchain schaffen. Diese Konstellation ist in der Form neu und weitestgehend unerforscht. Dadurch wurde folgende Forschungsfrage aufgestellt: “Inwiefern lassen sich Machine Learning und Blockchain verbinden bzw. wie gut lässt sich dies bereits umsetzen?”

Die daraus abgeleitete Problemstellung hatte die Entwicklung eines Systems

5 Zusammenfassung

zur Folge, dessen Ziel das Erlernen des Kaffeetrinkverhaltens der Lehrstuhlmitarbeiter durch eine Machine Learning Instanz (Q-Learning) war. Dieser *Learner* wiederum sollte mit Daten von einer privaten Ethereum Blockchain trainiert werden.

Zur Beantwortung der Forschungsfrage wurde eine Studie durchgeführt. Ziel der Studie war es zu erörtern, ob und wie gut der Q-Learning Algorithmus das Kaffeetrinkverhalten der User erlernt.

Im Kontrast dazu wird in der Arbeit eine Anwendung zur Simulation eines virtuellen Users vorgestellt. Die Durchführung mehrerer Simulationen zeigt, das vollständige Erlernen des Kaffeetrinkverhaltens, kann mit dem implementierten Q-Learning Algorithmus nach ca. 170 Episoden erreicht werden. Die Lernkurve spiegelt dabei im Verlauf ihres Graphen eine Ähnlichkeit zu dem des Logarithmus wieder.

Eine wichtige Rolle spielen hierbei die Modellierung (v.a. Größe des Zustandsraums) des Lernproblems, sowie die Konsistenz des Datensatzes, welcher das Verhalten des Users widerspiegelt.

Die Auswertung der Studienergebnisse zeigen grundsätzlich, dass sich die Blockchain Technologie mit Machine Learning kombinieren lässt. Probleme während der Testphase traten vor allem aufgrund der veralteten Hardware bzw. Software (Tablet) in Form von Verbindungsabbrüchen auf. Dies resultierte in der verkürzten Laufzeit der Studie und einer Reduktion der auswertbaren Datensätze.

Die analysierten Datensätze weisen in beiden Fällen charakteristische Eigenschaften des implementierten Q-Learning Algorithmus auf. Dies ist in der Hinsicht als positiv zu bewerten, da es darauf hindeutet, dass der Algorithmus mit Daten von der Blockchain lernen kann.

Wie gut dies jedoch im Vergleich zum herkömmlichen Weg ohne Blockchain

(vgl. Simulation) funktioniert, lässt sich, in Anbracht der gerade eben geschilderten Problematik, erst bei einer längeren Laufzeit der Studie vollständig beurteilen. Allerdings lassen die aufgezeigten Tendenzen in den Studienergebnissen vermuten, dass der Lernerfolg wohl im gleichen Maße möglich ist, sollten keine größeren technischen Ausfälle die Qualität der Datensätze beeinträchtigen.

5.2 Ausblick & Weiterführende Forschungsfragen

Auf Basis des entwickelten Systems können nun Überlegungen durchgeführt werden, inwiefern man z.B. die Forschungsfrage weiter verfolgt, an der Umsetzung der Komponenten Änderungen vornimmt oder eine Abwandlung der Problemstellung durchführt.

Denkbar wäre zum Beispiel ein Wechsel auf die öffentliche Ethereum Testchain. Dabei wäre zu eruieren inwieweit eine Diskrepanz zu einer privaten Blockchain entsteht. Einerseits hinsichtlich der Einschränkungen, welche durch eine solche public Blockchain vorherrschen, und andererseits wie sich das auf den Lernerfolg des Learners auswirkt. Der Wechsel zur Testchain wäre indessen relativ einfach vorzunehmen und würde nur wenige Änderungen im Quelltext bedürfen.

Zudem wäre es interessant zu sehen, ob ein Wechsel von Q-Learning auf DYNA-Q, bei einer identischen Laufzeit der Studie, einen signifikanten Anstieg in der Lernkurve bedeuten würde?

Eine mögliche Abwandlung der Forschungsfrage bzw. der Problemstellung wäre der Wechsel hin zu einem anderen Machine Learning Paradigma. So bestche

5 Zusammenfassung

die Möglichkeit das Verhalten durch Unsupervised Learning zu erlernen, in dem die Logs, wie auch bei Singla et. al [SBK18], in regelmäßigen Abständen durchsucht und die notwendigen Informationen extrahiert werden.

Hierbei wäre ein Vergleich zu ziehen, ob Unsupervised Learning anstatt Reinforcement Learning die bessere Wahl ist, um das Verhalten der User zu erlernen, respektive inwieweit eine Feature Selection/Extraction notwendig ist?

Als eine weiterführende Forschungsfrage ist es auch denkbar zu erforschen, ob es möglich ist einen Machine Learning Algorithmus in Form eines Smart Contracts zu implementieren und darauf zu trainieren. Diese “Verschmelzung” würde - in Abhängigkeit des gewählten Algorithmus - volle Transparenz für den User bedeuten, da sowohl die Rohdaten als auch die Algorithmusdaten zu jederzeit auf der Blockchain einsehbar sind.

Literaturverzeichnis

- [Abh09] Abhishek Chakravarty. Here's how i built a private blockchain network, and you can too, 2017-09-09.
- [Ame20] Ameer Rosico. What is an ethereum token: The ultimate beginner's guide, 2017-09-20.
- [Ant17] Andreas M. Antonopoulos. *Mastering Bitcoin: Programming the Open Blockchain*. O'Reilly Media, Inc., 2nd edition, 2017.
- [Bas18] Imran Bashir. Mastering blockchain second edition. 2018.
- [btc] btc-echo. Was ist proof-of-stake?
- [Bus18] Enée Bussac. Bitcoin, ethereum co. - praxiswissen kryptowährungen und blockchain. 2018.
- [CSPK07] Krzysztof J. Cios, Roman W. Swiniarski, Witold Pedrycz, and Lukasz A. Kurgan. *Unsupervised Learning: Association Rules*, pages 289–306. Springer US, Boston, MA, 2007.
- [DAp] Dapp.
- [Dav15] David Tuesta. Smart contracts: the ultimate automation of trust?, 2015-10-15.

- [Dem12] Demiro Massessi. Public vs private blockchain in a nutshell, 2018-12-12.
- [ERC20] Erc20 token standard, 2017-06-25.
- [ES18] Christoph Engemann and Andreas Sudmann, editors. *Machine Learning - Medien, Infrastrukturen und Technologien der Künstlichen Intelligenz*. Digitale Gesellschaft. transcript, Bielefeld, [2018].
- [Fra20] Francisco Giordano. openzeppelin-solidity, 2018-09-20.
- [IPFa] Ipfs - content addressed, versioned, p2p file system.
- [IPFb] Ipfs is the distributed web.
- [JJB18] Paul R. Allen Joseph J. Bambara. Blockchain - a practical guide to developing business, law, and technology solutions. 2018.
- [Lui15] Luisa Geiling. Distributed ledger: Die technologie hinter den virtuellen währungen am beispiel der blockchain, 2016-02-15.
- [Mit04] Tom M. Mitchell. *Machine learning*. McGraw-Hill series in computer science. McGraw-Hill, New York [u.a.], international ed., [nachdr.] edition, 2004.
- [onl] association rules (in data mining).
- [Pre07] Preston Van Loon. Ethereum 2.0 phase 0 testnet release, 2019-05-07.
- [Rap09] Raphael Honig. So lange dauert mining bei bitcoins | kryptopedia, 2018-04-09.
- [RN:] React native.
- [SB98] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

- [SBK18] Kushal Singla, Joy Bose, and Sharvil Katariya. Machine learning for secure device personalization using blockchain. pages 67–73, 09 2018.
- [Ste31] Greg Sterling. 90 percent use multiple screens during the same day, 2012-08-31.
- [Vit20] Vitalik Buterin. Proof of stake faq?, 2019-03-20.
- [Vit26] Vitalik Buterin. What proof of stake is and why it matters?, 2013-08-26.
- [Wei01] Gerhard Weiss, editor. *Multiagent systems*. MIT Press, Cambridge, Mass. [u.a.], 3. printing edition, 2001.
- [Zhe18] Alice Zheng. *Feature engineering for machine learning*. O’Reilly, First edition ; Beijing, April 2018.
- [ZRM10] Sofia Zaidenberg, Patrick Reignier, and Nadine Mandran. Learning User Preferences in Ubiquitous Systems: A User Study and a Reinforcement Learning Approach. In Harris Papadopoulos, Andreas S. Andreou, and Max Bramer, editors, *Artificial Intelligence Applications and Innovations*, volume 339 of *IFIP Advances in Information and Communication Technology*, pages 336–343, Larnaca, Cyprus, October 2010. Harris Papadopoulos and Andreas S. Andreou and Max Bramer, Springer.
- [05] Lee Ting Ting. Beginners guide to ethereum (3)—explain the genesis file and use it to customize your blockchain, 2017-07-05.

Abbildungsverzeichnis

3.1	Systemarchitektur	22
3.2	Geth	29
3.3	genesis.json	35
3.4	geth init	36
3.5	ERC-20 Interface	37
3.6	CoffeeCoin Interface Auszug	38
3.7	Mitarbeiter Page: kein Mitarbeiter ausgewählt	51
3.8	Mitarbeiter Page: Mitarbeiter ausgewählt	51
3.9	Drinks Page: kein Getränk ausgewählt	52
3.10	Drinks Page: Getränk ausgewählt	53
3.11	Overlay: Bestätigung der Transaktion	53
3.12	No Internet: keine Internetverbindung verfügbar	54
4.1	Simulation 1: Lernkurve	63
4.2	Simulation 2: Lernkurve	64
4.3	User 1: Montag	65
4.4	User 1: Dienstag	65
4.5	User 1: Mittwoch	65
4.6	User 1: Donnerstag	65
4.7	User 1: Freitag	66

Abbildungsverzeichnis

4.8	User 1: Wochenüberlick	66
4.9	User 2: Montag	68
4.10	User 2: Dienstag	68
4.11	User 2: Mittwoch	68
4.12	User 2: Donnerstag	68
4.13	User 2: Freitag	68
4.14	User 2: Wochenüberlick	68

Tabellenverzeichnis

3.1	Exmplarische Q-Tabelle	44
4.1	Normalverteilung Tabelle	60

A Anhang A

Der Quelltext der einzelnen Komponenten ist unter den Links folgender öffentlicher Github-Repos zu finden:

- Learner: "<https://github.com/pierback/bchain-qlearning>"
- Tablet-App: "<https://github.com/pierback/coffee-dash-rn>"
- Smart Contracts: "<https://github.com/pierback/smart-contracts.git>"
- File-Server: "<https://github.com/pierback/go-file-server>"
- Private Chain: "<https://github.com/pierback/private-net-docker>"
- Node.js Test Client: "<https://github.com/pierback/web3-node>"

B Anhang B