



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: 27. Mai 2019

# **Abstract**

In dieser Masterarbeit ...



# Inhaltsverzeichnis

<b>Abstract</b>	<b>i</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Aufbau dieser Arbeit . . . . .	1
1.2 Problemstellung . . . . .	1
1.3 Related Work . . . . .	4
<b>2 Stand der Technik</b>	<b>5</b>
2.1 Blockchain . . . . .	5
2.1.1 Funktionsweise & Konzepte . . . . .	5
2.1.2 Ethereum . . . . .	9
2.2 Machine Learning . . . . .	11
2.2.1 Überblick . . . . .	11
2.2.2 Reinforcement-Learning . . . . .	13
<b>3 System Architektur</b>	<b>17</b>
3.1 Überblick . . . . .	17
3.1.1 Architektur . . . . .	17
3.1.2 Workflow . . . . .	24
3.1.3 Entwicklungsprozess . . . . .	32

## *Inhaltsverzeichnis*

3.2	Blockchain . . . . .	37
3.2.1	Genesis Block . . . . .	37
3.2.2	CoffeeCoin . . . . .	40
3.2.3	Beverage-List . . . . .	42
3.3	Learner . . . . .	43
3.3.1	Modellierung . . . . .	44
3.3.2	Q-Learning . . . . .	46
3.3.3	Lernprozess & Ablauf . . . . .	50
3.4	Tablet-App . . . . .	52
3.4.1	Interface . . . . .	53
3.4.2	Internal Workflow . . . . .	57
<b>4</b>	<b>Studie</b>	<b>61</b>
4.1	Simulation . . . . .	61
4.2	Testphase . . . . .	63
4.3	Evaluation . . . . .	64
<b>5</b>	<b>Zusammenfassung</b>	<b>69</b>
5.1	Weiterführende Forschungsfragen . . . . .	69
5.2	Ausblick . . . . .	69
	<b>Literaturverzeichnis</b>	<b>I</b>
	<b>Abbildungsverzeichnis</b>	<b>III</b>
	<b>Tabellenverzeichnis</b>	<b>V</b>
<b>A</b>	<b>Anhang A</b>	<b>VII</b>
<b>B</b>	<b>Anhang B</b>	<b>IX</b>

# 1 Einleitung

## 1.1 Aufbau dieser Arbeit

## 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, Plattformen, Systeme 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 dieses Wissens App-Interfaces oder auch Hintergrundprozesse anzupassen, sodass für jeden Nutzer eine optimale User-Experience sichergestellt werden kann. Hierbei kann auch von sog. virtuellen Assistenten gesprochen

## *1 Einleitung*

werden, welche durch die direkte Interaktion mit dem User oder durch das Beobachten des Users, dessen Verhalten und Gewohnheiten versucht 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 die Rechenleistungen 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 gehostet 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 Grundsatzfrage: wie gut lässt sich Blockchain mit Machine Learning verbinden? Also ist es möglich virtuelle Assistenten anhand von Daten von 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.

Eine weitere Frage welche aus dieser Problemstellung resultiert ist: wie eine notwendige “Feature Selection” auf einer Blockchain aussieht?

Um dies zu erforschen und abzubilden, wird in dieser Arbeit ein System vorgestellt, welches sich den gerade eben geschilderten Problemstellungen annimmt. Hierbei wird eine Getränkeliste an einem Lehrstuhl durch eine Tablet-App er-



setzt, in welcher die konsumierten Getränke (Kaffee, Club Mate, Wasser) eingegeben werden. Die kontextuellen Daten werden daraufhin auf eine private Blockchain gespeichert und das Getränk zudem 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.

(Vor dem Hintergrund der Neuartigkeit der Blockchain Technologie und der daraus resultierenden Unausgereiftheit der dazugehörigen Tools (Libraries, Kommandozeilenbefehle etc.), ist im ersten Schritt die Umsetzung der gerade eben geschilderte Systematik das Ziel. Dabei soll eine indirekte Kommunikation zwischen der Tablet-App und der Learning-Instanz (Q-Learning Agent) über die Schnittstelle Blockchain hergestellt werden.

Als Methodik wird eine Analyse auf Basis der Quantität verwendet. Hierbei wird eine bestimmte Anzahl an Schreiboperationen auf der Blockchain durchgeführt und daraufhin analysiert wie viele von diesen von der Learner-Instanz detektiert wurden. Die Diskrepanz der beiden Werte gibt Aufschluss darüber, inwiefern es sinnvoll ist die gestellten Forschungsfragen auf die formulierte Art und Weise weiter zu verfolgen.)

## *1 Einleitung*

Bei erfolgreicher Durchführung von Schritt 1, wird das System am Lehrstuhl installiert und im Zeitrahmen von 2 Monaten getestet. In diesem Zeitraum soll eruiert werden inwiefern es der Learner-Instanz möglich ist, das Kaffeetrinkverhalten eines jeden Lehrstuhlmitarbeiters zu erlernen.

Während dieser Testphase wird eine Anwendung zur Simulation eines realen Users entwickelt, welche die Learner-Instanz direkt mit den Userdaten versorgt, ohne diese zuvor auf die Blockchain zu schreiben. In diesem Fall soll der Lernerfolg aus der Simulation als Maßstab und Bewertungsgrundlage dienen, an welchem der Lernerfolg aus der Testphase gemessen und bewertet wird.

semi dezentral

## **1.3 Related Work**

## 2 Stand der Technik

### 2.1 Blockchain

Der folgende Abschnitt basiert auf den Inhalten dieser Literatur: ??, ??, ?? und ??.

#### 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 “verteiltetes Konto” bzw. “dezentral geführtes Kontobuch” [Lui15] verstanden wird, in welchem jegliche Transaktionen abgespeichert werden.

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

## 2 Stand der Technik

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.

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 mit im Block abgespeichert. Erst aufgrund der Berücksichtigung des Hashwerts des vorherigen Blocks werden die Blöcke miteinander verknüpft und bilden somit eine chronologisch geordnete Kette.

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, welcher für das Abspeichern und Bereitstellen aller bestehenden Daten eines Netzwerks zuständig ist. Denn jeder Knoten in einem solchem Netzwerk ist sowohl Sender als auch Empfänger, sodass jeder Teilnehmer eine Kopie des Datenbestandes besitzt, was einen “Single Point of Failure” völlig ausschließt. Aus diesem Grund sind Daten einer Blockchain nie nur bei einem Knoten abgespeichert, sondern jede Node besitzt 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 abetracht des Konsens aller Parteien im Netzwerk. Diese Eigenschaft wird als *distributed consensus* bezeichnet und ist das Fundament einer jeder Blockchain. Durch die Einbeziehung eines jeden Teilnehmers in der Entscheidungsfindung wird die Notwendigkeit einer zentralen Entscheidungsinstanz obsolet.

Damit löst die Blockchain ein Gedankenexperiment von Lamport aus dem Jahr 1982. Es plant eine Gruppe an byzantinischen Generälen, welche jeweils ver-

schiedene Teile der byzantinischen Armee befehligen, einen Angriff auf eine Stadt, der nur in einem Sieg resultiert wenn alle gemeinsam angreifen. Der einzige Weg der Interkommunikation und Koordination ist via Boten. Somit entsteht eine Problematik, sollten sich ein oder mehrere Verräter unter den Generälen befinden, die falsche Informationen streuen.

Aus diesem Grund bedarf es einem Mechanismus der es erlaubt, trotz Falschinformationen durch potentielle Verräter, einen Konsens unter den Generälen zu schaffen, um gemeinsam anzugreifen.

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 dieses 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. Der Mechanismus welcher dafür zuständig ist diesen Konsens herbeizuführen, ist je nach Implementierung des Blockchainprotokolls unterschiedlich. Eine beliebte Methodik, welche unter anderem von Bitcoin und Ethereum verwendet wird, ist der *Proof-of-Work* Ansatz. Hierbei werden von den sog. “Minern” Iterationen an aufwändigen und komplexen Berechnungen durchgeführt, die sicherstellen sollen, dass die benötigten kryptographischen Berechnungen für eine Transaktion durchgeführt und die Daten einer Transaktion validiert werden.

Eine weitere Besonderheit einer Blockchain ist die Unveränderbarkeit der darauf gespeicherten Daten. So gibt es, im Gegensatz zu den bekannten CRUD-Operationen <sup>1</sup>, welche zur Kommunikation zwischen Client und Server verwendet werden, um Daten zu schreiben, zu downloaden, zu löschen und zu

---

<sup>1</sup>CRUD:Wiki

## 2 Stand der Technik

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 und bedeutet somit volle Transparenz für jeden User.

Die gerade geschilderten Eigenschaften sind einer jeden Blockchain inhärent bzw. in einer abgeänderten Form (z.B. *Proof-of-Stake*<sup>2</sup> anstatt *Proof-of-Work*) vorhanden. Was jedoch erst in neueren Blockchains (Blockchain 2.0) vorzufinden ist, sind die *Smart Contracts*.

Smart Contracts sind Programme welche 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 Mittelmanns, welcher das Vertrauen aller beteiligten Parteien inne hat. Diese "Trust-Komponente" wird durch die Anerkennung aller Parteien von dem Smart Contract übernommen.

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

---

<sup>2</sup>Konsensalgorithmus welcher anstatt von Rechenleistung einen bestimmten Betrag an Ether als Pfand erwartet. Je höher dieser Wert desto wahrscheinlicher ist es, dass der Nutzer die Transaktion als neuen Block bestätigt [btc, Wikb]

### 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 sog. Dapps (Decentralized Apps) betrieben werden können. So wird Ethereum im Vergleich zu Bitcoin auch als Blockchain 2.0 bezeichnet, da es eben nicht nur eine Kryptowährung umfasst, sondern es aufgrund der Smart Contracts es möglich ist Software auf einer Blockchain zu installieren.

Als Ethereum wird genauer genommen das Protokoll titulierte 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 bzw. war 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 ihre Rechenleistung zur Verfügung stellen und als Gegenleistung bzw. Anreiz in Form von Ether vergütet werden.

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 und ermöglicht es dadurch den Smart Contracts Daten auf die Blockchain zu speichern und bietet 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 Orientie-

## 2 *Stand der Technik*

rung, ist der Sprache Javascript angelehnt und ist zudem Turing-Vollständig, was im Bezug auf die Entwicklung von Apps und eine hohe Bandbreite an Anwendungsfällen eröffnet.

Damit die Funktionen der Smart Contracts überhaupt genutzt werden können, muss einem jedem Methodenaufruf bzw. 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 und Belohnung zur Bereitstellung von Rechenleistung übertragen 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 voraussichtlich im Juni 2019 der Wechsel auf einen Proof of Stake Konsensalgorithmus erfolgen.

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 eine 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 haben dazu geführt, dass Algorithmen in vielerlei Bereiche des Alltags Einzug gefunden haben. Vor allem bei Aufgaben bei denen eine große Menge an Daten an vorhanden ist, werden diese Algorithmen verwendet, um relevante Informationen daraus zu extrahieren.

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 Verständnis für eine konkrete Problemstellungen in der realen Welt.

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

Eines davon fällt unter die Begrifflichkeit Data Mining. 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 deren enormen Vielzahl 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.

## 2 Stand der Technik

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 eignet 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 dieser auch als Feature Vektor bekannt. Machine Learning beschreibt somit die Identifikation von Strukturen und Muster 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 Art des Lernprozesses.

Die wohl bekannteste Kategorie ist das Supervised Learning. Algorithmen welche 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 dazu in der Lage bei Eingabe von unbekannten Input-Daten, basierend auf dem angeeigneten Wissen, eine Generalisierung

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. Denn diese können trotz fehlender Labels im Datensatz statistische Strukturen aufdecken bzw. ein Modell für diesen Datensatz erarbeitet. 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, wird als Reinforcement Learning bezeichnet und wird im nächsten Unterkapitel genauer betrachtet.

### 2.2.2 Reinforcement-Learning

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, sondern dieser versucht 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

## 2 Stand der Technik

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, indem der Agent stets auf die, in einem bestimmten Zustand, bestbewertete Aktion zurückgreift. Je nach Algorithmus und Lernproblem variiert das Verhältnis der beiden, welches durch eine (iterative) Justierung der Parameter anzupassen gilt.

Durch die Wechselwirkung der beiden 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 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
- 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 des Experiments 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 (sehr) unterschiedlich. Eine Kategorisierung wird in der Regel anhand der Berechnung der Wertfunktion gemacht und dadurch ergeben sich drei verschiedene Klassen an Reinforcement Algorithmen. Diese differieren durch verschiedene Lösungsansätze in der Berechnung der Wertfunktion.

Als erste Kategorie ist die Dynamische Programmierung (DP) zu nennen. Diese 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 Markow-Entscheidungsproblem (MEP oder auch MDP *Markov decision process*). Dieses Modell erlaubt die Erarbeitung einer Strategie zur

## *2 Stand der Technik*

Reward-Maximierung, unter der Berücksichtigung der bekannten Übergangswahrscheinlichkeiten und den erwarteten Rewards. Das ist auch der große Nachteil dieser Methodik, da diese Berechnungen sehr rechenaufwändig sind und zudem ein perfektes Umgebungsmodell in den meisten Fällen nicht vorhanden ist.

Unter die nächste Kategorie fallen die Monte Carlo (MC) Methoden. Im Gegensatz zur Dynamische Programmierung berechnen diese 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 bei denen Episoden auch terminieren. Somit ist es 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 den bereits vorangegangene Schätzungen und nähert sich dadurch inkrementell dem “wahren” Wert der Wertfunktion an, ohne dabei jedes mal auf die Terminierung einer Episode warten zu müssen. So ist auch bei dieser Art von Algorithmen kein Umgebungsmodell von Nöten, sondern die Bewertungen der Aktionen bzw. werden 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 3.3 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

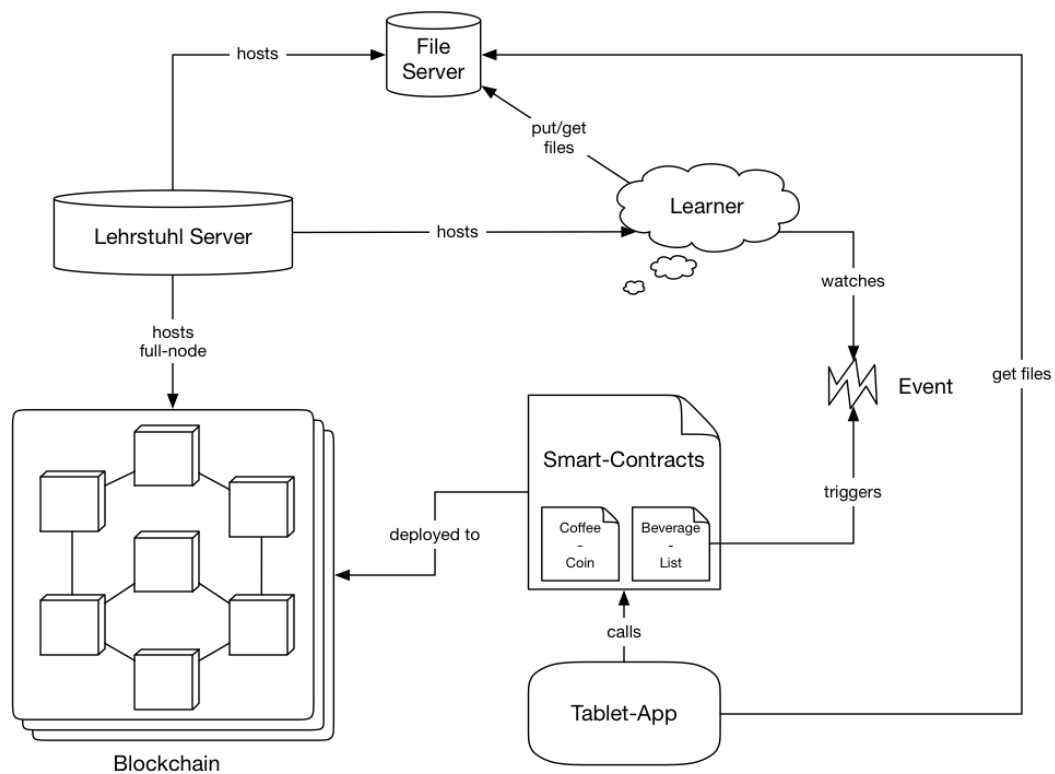


ABBILDUNG 3.1: Systemarchitektur

#### 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 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 Softwareanwen-



derung, dessen 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 diesen eigenständig bearbeiten. Auch wenn jene für sich autark agieren können, 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, welches aber unter `refsubsec:ql` genauer erläutert wird.

Der **Worker** ist einerseits für die Userverwaltung und andererseits für die, in einem festgelegten Intervall, 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 um den Q-Learning Algorithmus zu befüllen und aufgrund diesen das Trinkverhalten zu erlernen.

### 3 System Architektur

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, da 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 Go-Anwendung, welche eine rudimentäre REST [Wike] Api [Wika] zur Verfügung stellt. Von den sogenannten CRUD [Wikd] 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 v.a. beim Testing und beim Setup eine große Erleichterung darstellt. Dieser Broadcast versendet in seinen Nachrichten lediglich die IP-Adresse des Lehrstuhl-Servers und somit auch seine eigene und 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 “subscriben” und können dadurch die IP der Blockchain und des Fileservers

### 3.1 Überblick

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 ??) generiert wird. Die Erläuterungen zu den jeweiligen Key-Value-Pairs sind unter Kap. 3.2.1 zu finden. Das Generieren und das Starten der Blockchain erfolgt auf dem Lehrstuhlserver. Dabei hostet der Server zudem eine sogenannte “full-node” (auch “miner” genannt) der Blockchain, welche dafür zuständig ist Transaktionen zu berechnen und zu bestätigen. Aus Ressourcengründen ist dieser “miner” der einzige im Gesamtsystem, was aus theoretischer Sicht einen “Single Point of Failure” [Wikf] als Nachteil mit sich zieht. Das bedeutet sollte diese “full-node” ausfallen, würden 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. Der große Vorteil besteht allerdings darin, dass Transaktionen sehr schnell bestätigt werden, da es keine weiteren “node’s” gibt, die um die Berechnung eines Block’s konkurrieren. Was vor allem aus Sicht der User-Experience [Wikg] einen großen Mehrwert darstellt, da dieser 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 kommunizieren und dessen Potential in voller Gänze ausschöpfen zu können, werden auf diese sogenannte Smart-Contracts [Dav15] deployed. Im Rahmen der Systemarchitektur sind es zwei dedizierte Smart-Contracts (*Coffe-Coin*, *Beverage-List*), welche komplett unabhängig voneinander agieren.

#### Smart Contracts

Die beiden Smart Contracts welche auf die Blockchain deployed werden, werden mit *Coffe-Coin* und *Beverage-List* betitelt. Diese decken zwei völlig unterschiedliche Aufgabenbereiche ab, weswegen sie keinen Einfluss aufeinander haben und deswegen unabhängig voneinander operieren. So löst nur der *Beverage-List Contract* 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 Crossplattform App [Wikc], 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 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 und vom *Learner* detektiert und der 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 und die zweite beschreibt den eigentlichen Ablauf und das Zusammenwirken der Instanzen.

#### 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 Überblick

- 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

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:

### 3 System Architektur

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The terminal displays the command 'geth' followed by a series of options on separate lines, each ending with a backslash to indicate it's part of a single command.

```
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 Befehl zum starten der Blockchain

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 aber zugleich auch eine sogenannte *full node* gestartet (*--syncmode*), die durch das “Flag” *--mine* sofort zum “minen” beginnt.

Desweiteren 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 dem 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

### 3 System Architektur

jeweils für 3a) und 3b) das identische Bash-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 dazu verwendet werden ein Go Bindingsfile zu generieren. Im Anschluss wird dann ein Go-Skript ausgeführt, welches auf Grundlage des Bindingsfiles den Smart Contract auf der Blockchain installiert und die zurückgelieferte Adresse und die bereits bekannte ABI 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. Daraufhin werden sowohl der Worker als auch der Watcher in Form von “Goroutines” aktiviert. Der Worker iteriert über die Liste aller User und kontaktiert jeweils den Fileserver ob bereits gelernte Daten für diesen Usern vorhanden sind. Sollte das Fall sein, initialisiert er damit die Parameter des Q-Learning Algorithmus des 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

### *3.1 Überblick*

Daten zu bekommen. Welche im Anschluss dazu verwendet werden 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 (vgl. Abbildung 3.3) an Fileserver

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 gestartet indem dieser auf Tablet ein Getränk auswählt und eine Transaktion auslöst. Zuerst wird dabei der CoffeeCoin 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, Eth-Adresse) und löst damit ein Event aus.

Dieses Event wird vom Watcher detektiert und die Daten (Zeit, Getränk, Wochentag, Eth-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.

#### 3.1.3 Entwicklungsprozess

Abschließend wird der Prozess der Entwicklung geschildert, aus welchem schließlich die finale Version der Systemarchitektur resultierte.

Der Entwicklungsprozess beinhaltete mehrere Iterationen der einzelnen Komponenten bis hin zum derzeitigen Stand. Das Konzept sah primär die Entwicklung von drei dedizierten Software Anwendungen vor, welche aber im Zuge der Iterationen nochmal in kleinere Module aufgeteilt und ausgelagert wurden. Zudem wurden, um den Workflow und das Testen während der Entwicklungsphase zu erleichtern, Anwendungen entwickelt, welche während der Konzeption in der Art nicht vorgesehen waren, aber partiell Bestandteil der Systemarchitektur wurden.

So wurde mit zwei separaten Repos gestartet, einerseits für den Learning-Part, welcher anfänglich auch die Smart Contracts umfasste, und andererseits eines für die Tablet-App, welches bereits vor der eigentlichen Konzeption erstellt wurde, um in erster Linie bestehende Crossplattform Frameworks, auf Basis der Kompatibilität und Funktionstüchtigkeit mit Libraries, welche die Kommunikation mit der Blockchain ermöglichen, zu evaluieren.

Die Wahl fiel letztendlich auf React-Native, welches zwar nur bis zu einer bestimmten Versionsnummer der Web3.js Library von Ethereum vollends kompatibel ist und nur mit einem kleinen Workaround zum Laufen gebracht werden konnte. Jedoch im Vergleich zu anderen Frameworks (z.B. Nativescript) die beste Development-Experience (geringe Lernkurve, gute Dokumentation, CLI) bot und v.a. hinsichtlich der Requirements alle Aufgaben komplett erfüllen konnte, welche die anderen Frameworks in dieser Gänze nicht replizieren konnten.

Nachdem die erste rudimentäre Version der Tablet-App, welche lediglich eine

funktionierende Kommunikation (read/write) mit einem bereits bestehenden Smart-Contract auf einer lokal gehosteten Blockchain bestätigte, erstellt wurde, kam im nächsten Schritt der Learning-Part zum Zuge.

In Anbetracht der kompletten Implementierung des Ethereum Protokolls in Golang und der Schwierigkeiten mit der Javascript Library Web3.js, v.a. im Bezug auf das deployen der Smart-Contracts, aus einem vorangegangenen Projekt, fiel die Wahl für diese Instanz auf Golang.

Zuerst wurde der Q-Learning Algorithmus, welcher für das Erlernen des Kaffee Trinkverhalten zuständig ist, implementiert. Die Problematik bestand zum einen darin mit einer neuen Programmiersprache vertraut zu werden und zum anderen den Workflow hinsichtlich der Problemstellung und des daraus resultierenden Zustandsraums vollends abzubilden. Die Umsetzung des Algorithmus in der Programmiersprache ging relativ einfach von der Hand, was jedoch Probleme bereitete war die Simulation des Workflows, um die Algorithmus Parameter zu justieren und dessen Tauglichkeit bezüglich das Erlernen des Nutzerverhaltens zu testen.

Im Anschluss wurde ein erster Smart-Contract erstellt und via dem “go-ethereum” package deployed, woraus das erste Smart-Contract Bindingsfile resultierte, welches für die Kommunikation mit dem Smart Contract vonnöten ist. Da mit jedem neuem Deployment eines Smart Contracts eine neue Smart Contract Adresse und eine neue ABI hervorgeht, welche wiederum beide im Source Code für die Kommunikation mit dem Smart Contract, über alle Instanzen hinweg, die mit einem Smart Contract interagieren wollen, hinterlegt sein müssen, wurde ein kleiner HTTP-Fileserver entwickelt, auf dem diese Informationen gespeichert und gelesen werden können.

### 3 System Architektur

Bei jedem neuen Deployment werden daraufhin die Smart-Contract Adresse und die generierte ABI in ein JSON-File gepackt und an den Server geschickt. So konnte während der Entwicklung enorm an Zeit gespart werden, da sich sowohl die Learning-Instanz als auch die App, die benötigten Daten vom Server holen und somit ein ständiges “Hardcodieren” dieser Daten vermieden werden konnte.

Aus diesem Grund findet der Fileserver auch Einzug in die finale Systemarchitektur, da er als persistente Datenquelle eine enorme Erleichterung nicht nur im Entwicklungsprozess, sondern auch im “Live-System” darstellt.

Zudem wird der Fileserver auch für die Verwaltung der Algorithmus-Daten verwendet. Dabei wird bei jedem Worker-Durchlauf (vgl. Kap. 3.3.3) für jeden Nutzer ein JSON-File erzeugt, welches folgende Key-Value-Pairs beinhaltet (vgl. Kap. Abbildung 3.3):

- qt: die aktuelle Q-Tabelle des Users
- ep: der aktuelle Epsilon-Wert
- negs: Anzahl der falschen Predictions in der aktuellen Woche
- Wk\_negs: Array von negs über alle Wochen hinweg

Der Vorteil liegt in der Möglichkeit den Learner jederzeit upzudaten ohne die gelernten Daten zu verlieren. Denn wird der Learner gestartet, holt sich dieser zuerst die Files vom Server, liest die Daten aus und initialisiert schon im Vorab die Q-Tabelle und das Epsilon eines jeden Users.

Sollte der Learner aus unbestimmten Gründen abstürzen, ist durch den eben beschriebenen Algorithmus die Erhaltung des Lernfortschrittes trotzdem sichergestellt.



```
{
  "qt": "f\\{\\\\"weekday\\\\"":1,\\\\"timeslot\\\\"":0,\\\\"Drinkcount\\\\"":
{\\\\"coffeeCount\\\\"":0,\\\\"waterCount\\\\"":0,\\\\"mateCount\\\\"":0}\\":
[0.8507462686509051,-0.16666666666666669],\\
{\\\\"weekday\\\\"":1,\\\\"timeslot\\\\"":1,\\\\"Drinkcount\\\\"":
{\\\\"coffeeCount\\\\"":0,\\\\"waterCount\\\\"":0,\\\\"mateCount\\\\"":0}\\":[0.7,0],\\
{\\\\"weekday\\\\"":1,\\\\"timeslot\\\\"":2,\\\\"Drinkcount\\\\"":
{\\\\"coffeeCount\\\\"":0,\\\\"waterCount\\\\"":0,\\\\"mateCount\\\\"":0}\\":[0,0],\\
{\\\\"weekday\\\\"":3,\\\\"timeslot\\\\"":0,\\\\"Drinkcount\\\\"":
{\\\\"coffeeCount\\\\"":0,\\\\"waterCount\\\\"":0,\\\\"mateCount\\\\"":0}\\":
[0.8375858510195312,-0.6635069219333333],\\
{\\\\"weekday\\\\"":3,\\\\"timeslot\\\\"":0,\\\\"Drinkcount\\\\"":
{\\\\"coffeeCount\\\\"":1,\\\\"waterCount\\\\"":0,\\\\"mateCount\\\\"":0}\\":
[0,-0.9999969885009999],\\{\\\\"weekday\\\\"":3,\\\\"timeslot\\\\"":0,\\\\"Drinkcount\\\\"":
{\\\\"coffeeCount\\\\"":2,\\\\"waterCount\\\\"":0,\\\\"mateCount\\\\"":0}\\":[0.999271,0]},
  "ep": "0.988516",
  "necs": 8,
  "Wk_necs": [18, 17, 13, 15, 12, 9]
}
```

ABBILDUNG 3.3: 0x6ecbe1db9ef729cbe972c83fb886247691fb6beb-ql.json  
Der Name des JSON-Files setzt sich aus der Ethereum-Adresse des Users und der Abkürzung "ql", welches für Q-Learning steht, zusammenhängen.

Als letztes Modul wurde ein kleiner Node-Server entwickelt, dessen Aufgaben darin bestand die Smart-Contracts zu testen und als primitiver Ersatz für die App zu fungieren. Hierbei erzeugte er in einem festgelegten Intervall (7sek) Events mit einem zufällig generierten Daten (User & Getränk) auf der Blockchain, um letztendlich die Event-Erkennung (“Watcher”) des “Learners” zu testen. Dabei wurde sowohl für den “Beverage-List Contract” als auch für den “CoffeeCoin-Contract” eine entsprechende Implementierung angefertigt. In diesen Fällen wurde das Q-Learning-Submodul des Learners gar nicht erst gestartet, da lediglich die Funktionsweise des Watchers getestet werden sollte. Besonders hier zeigte sich die Nützlichkeit des Fileservers, da gerade in der Entwicklungsphase die Smart-Contracts noch häufigen Änderungen unterlagen und dahingehend sehr oft neu deployed werden mussten, was ohne den Fileserver dazu geführt hätte die Smart Contract Daten bei jeder Iteration neu im Sourcecode zu hinterlegen.

Nach einer längeren Testphase, in der eine einwandfreie Kommunikation mit

### 3 System Architektur

den beiden Smart Contracts attestiert werden konnte, wurde mit der eigentlichen Entwicklung der App begonnen, für jene auch Teile der Node-Server Implementierung übernommen werden konnten.

Schwierigkeiten traten dabei erst in der Testphase auf, in der festgestellt wurde, dass zu wenig Events vom Learner detektiert werden. Die Ursache dafür lag an der sehr alten Android Version des Tablets, die nicht ermöglichte eine direkte Verbindung zum Uni-Netzwerk herzustellen. Dies gelang nur mit einem Workaround, bei dem sich das Tablet mit einem öffentlichen Wlan-Netzwerk verband und sich daraufhin über eine VPN-Verbindung in das Uni-Netzwerk einwählen konnte.

Das führte jedoch dazu, dass die Verbindung zum Wlan-Netzwerk in unregelmäßigen Abständen abbrach und dadurch auch zur Blockchain. Da so ein unvorhergesehenes Verhalten wurde in der ersten Implementierung der App nicht vorgesehen war, musste dies in einem Update der App berücksichtigt werden (vgl. Kap. 3.4), sodass keine Daten verloren gingen, sollte die Verbindung abbrechen.

Schlussendlich waren es sechs dedizierte Software Anwendungen, welche jeweils in eigenen Git-Repositories gehostet werden. Dazu zählten:

- Learner
- Tablet-App
- Go Fileserver
- Smart Contracts: Beverage-List, CoffeeCoin
- Web3 Node-Server
- Dockerimage für die Blockchain

Das Dockerimage fand in der Hinsicht keine größere Erwähnung, da es nur zu Test- und Weiterbildungszwecken entwickelt wurde und auch keine Verwendung in der finalen Architektur fand.

## 3.2 Blockchain

Das folgende Kapitel erläutert im Detail den Setup der privaten Blockchain, sowie die beiden Smart Contracts welche ebenso ein 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, welcher Accounts zugeordnet und somit Transaktionen durchgeführt werden können.

Die Konfiguration dessen erfolgt durch ein sog. “genesis.json file” (3.5). 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.

### 3 System Architektur

Das in 3.5 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 allerdings kurz erläutert:



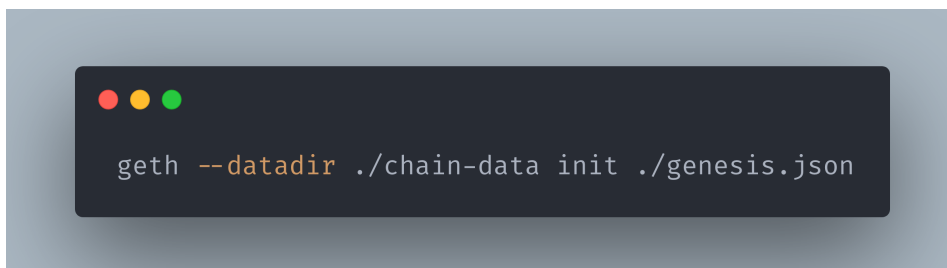
```
{
  "config": {
    "chainId": 50,
    "homesteadBlock": 1,
    "eip150Block": 2,
    "eip150Hash": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "eip155Block": 3,
    "eip158Block": 3,
    ...
  },
  "difficulty": "0x1",
  "nonce": "0x0",
  "timestamp": "0x5af1ffac",
  "gasLimit": "0x4c4b400000",
  "mixHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "coinbase": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "alloc": {
    "0xe8816898d851d5b61b7f950627d04d794c07ca37": {
      "balance": "0x56BC75E2D63100000"
    },
    "0x5409ed021d9299bf6814279a6a1411a7e866a631": {
      "balance": "0x56BC75E2D63100000"
    },
    ...
  },
  "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000"
}
```

ABBILDUNG 3.4: genesis.json  
JSON-File welches zur Erstellung des Genesis Blocks verwendet wurde.

- 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 sollten 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 eines Testnets ist es deshalb ratsam einen sehr niedrigen Wert zu wählen
- `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:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The command `geth --datadir ./chain-data init ./genesis.json` is entered in the terminal.

```
geth --datadir ./chain-data init ./genesis.json
```

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

### 3 System Architektur

Entwicklung eines Smart Contracts begonnen werden.

Die nächsten beiden Unterkapitel befassen sich mit den Smart Contracts, welche im Zuge dieser 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 eine zusätzliche Währung 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 weit verbreitetste Standard ist der ERC-20, bei welchem es folgende Funktionen und Events zu implementieren gilt:

A screenshot of a code editor with a dark background and light-colored text. The code defines the ERC20Interface contract. It includes functions for totalSupply, balanceOf, allowance, transfer, approve, and transferFrom, as well as Transfer and Approval events. The code is as follows:

```
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
- `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:

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.



ABBILDUNG 3.6: CoffeeCoin Interface Auszug

Desweiteren 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 soll vor allem die Möglichkeit einer solchen Bezahlungsmethode aufgezeigt werden, weswegen die 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.

Der Sourcecode ist im Anhang unter REF SO UND SO zu finden.

#### 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 Getränkliste abzubilden, in welcher jede Getränktransaktion 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 dem Learner schlussendlich ermöglichen das Kaffeetrinkverhalten des Users zu erlernen. Damit der Learner ohne großen Aufwand auf diese Informationen zugreifen kann, löst der Smart Contract, mit den eben genannten Daten beinhaltend, ein Event aus, sobald seine Methode “setDrinkData” aufgerufen wird. Diese Funktion hinterlegt die übergebenen Daten (Eth-Adress, Zeit, Wochentag, Getränk) in 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 sogenannte “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.

Dabei werden zuerst der Zustandsraum, also alle möglichen Zustände, alle Aktionen des “Agenten” und der Reward für ausgeführte Aktionen eruiert. Hier 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:
  - 7-9 Uhr (T0)
  - 10-12 Uhr (T1)
  - 13-15 Uhr (T2)
  - 16-18 Uhr (T3)
  - 19-6 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 (13-15 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 Kaffee's nur in Timeslot 2 getrunken worden sein oder in jedem Timeslot (0,1,2) jeweils einen. Andererseits besteht auch die Möglichkeit, dass im 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 Kaffe-Anzahl eine Getränkeanzahl vor, bei der zum Kaffee auch die Quantität der getrunkenen "Clube Mate" und "Wasser" berücksichtigt werden sollten. Der Grund dafür liegt in dem

### 3 System Architektur

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.

TO-DO: feature selection to add user versteht zu jeder warum algorithmus so handelt

#### 3.3.2 Q-Learning

Der Q-Learning Algorithmus fällt unter die Methodik des TD-Learnings (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, hingegen beurteilt es die Aktion in diesem Zustand. Hierbei wird von der Aktion-Wert Funktion  $Q$  gesprochen.

Die Berechnung von  $Q$  erfolgt nach folgender Formel (3.3.2):

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

- $Q(s_t, a_t)$ : beschreibt den aktuellen Q Wert bzw. Q-Value der Aktion a welche zum Zeitpunkt t im Zustand s ausgeführt wurde
- $\alpha$ : 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
- $\gamma$ : ist der Diskontierungsfaktor und wird als Gewichtung für zukünftige Rewards verstanden
- $\max_a Q(s_{t+1}, a)$ : 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. 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.

---

**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
  
```

---

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 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. ??) 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 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  $\epsilon$ -greedy. Dabei beschreibt die Variable  $\epsilon$  einen Wert zwischen '0' und '1' und gibt dadurch das Verhältnis zwischen Exploration und Exploitation an - in der Regel ist der Wert von  $\epsilon$  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  $\epsilon$ -greedy jedoch, wird zu einem gewissen Anteil - definiert durch das  $\epsilon$ - nicht immer die Aktion ausgewählt welche den größten Reward verspricht, sondern eine Aktion zufällig selektiert, obgleich des erwarteten Rewards. Somit wird eine Erforschung des Zustandsraum durch den Lernagenten sichergestellt und eine Beschränkung auf lediglich bekannte Aktionen und Zustände vermieden.

Eine Besonderheit der Implementierung liegt in der Vakanz der Berücksichtigung von Folgezuständen in 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 Independenz der Zustände untereinander, 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.

Diese Form der Problemstellung wird auch als *n-armiger 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 es seine Belohnung zu maximieren (Exploitation) und wird deswegen nach jeder Aktion vor die Entscheidung gestellt, ob der Wechsel zu einem anderen Automat einen größe-

ren 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 anhand von nachfolgenden Aktionen bewertet, sondern einzig anhand der Belohnung nach ihrer Ausführung. Außerdem handelt es sich in beiden Fällen um ein nichtdeterministisches Lernproblem, bei welchen die Ausführung einer Aktion in einem bestimmten Zustand stets einen Reward nach 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 der Hinsicht von großer Bedeutung, da für diese Übergänge ein Feedback für den Agenten notwendig ist, um jeweils 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”.

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.

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, indes 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 6 Uhr (T4) nicht in der Zustandsraummodiellerung 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 jedes Tages

### 3 System Architektur

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 Getränk- und Userdaten auf die Blockchain schreibt.

Die entwickelte App basiert auf dem Crossplattform 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. Diese Library (Web3.js) wurde von Ethereum dafür entwickelt, um sog. DApp's ("decentralized apps") [DApp] 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 bzw. zur Blockchain, so findet der User folgende Startseite vor:

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:

Hier besteht eine Auswahl aus folgenden Getränken: Club Mate, Wasser und Kaffee. Ausgewählt kann jedoch immer nur eines werden (vgl. Abbildung 3.12).

### 3 System Architektur

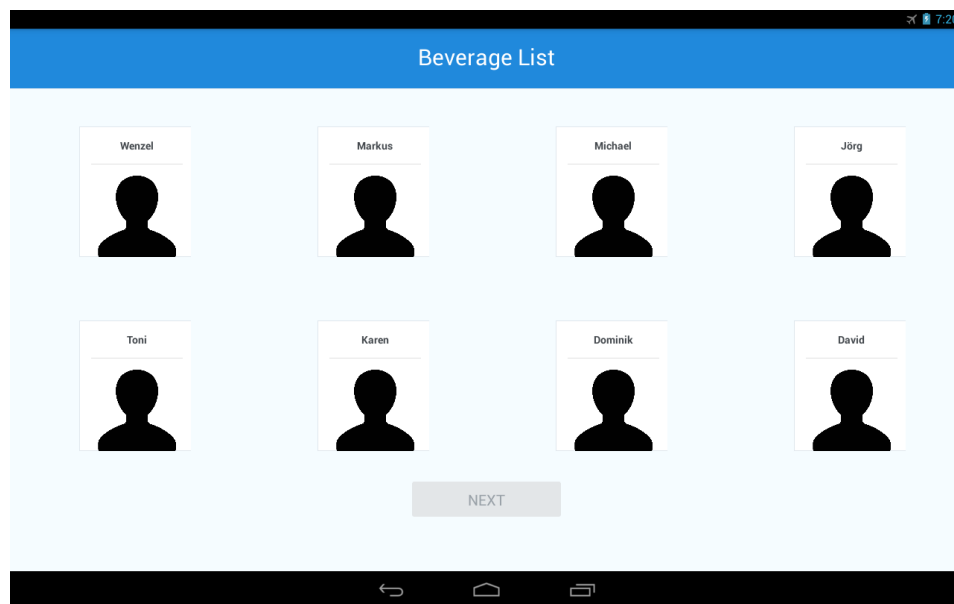


ABBILDUNG 3.7: Mitarbeiter Page: kein Mitarbeiter ausgewählt

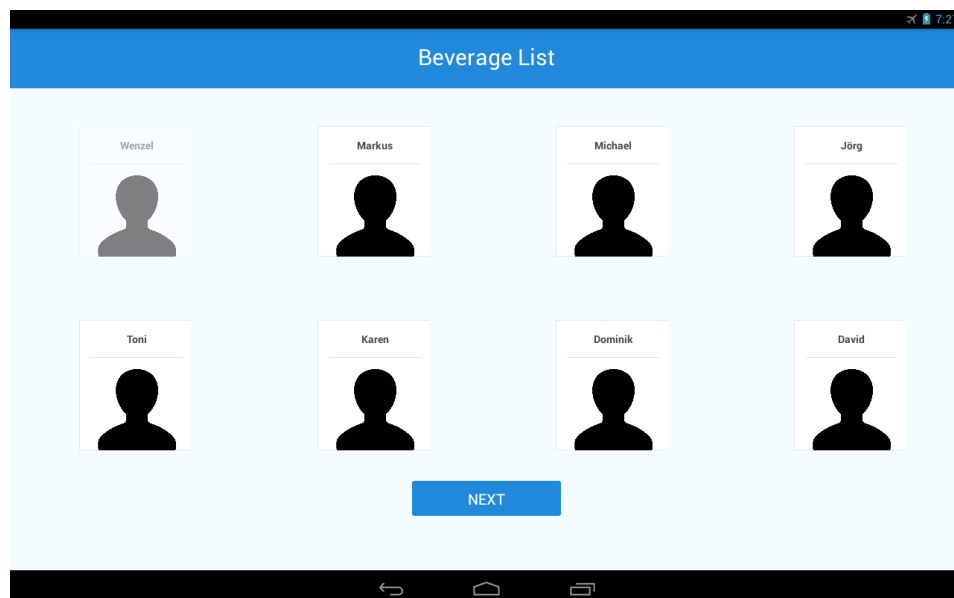


ABBILDUNG 3.8: Mitarbeiter Page: Mitarbeiter ausgewählt

Das Prinzip der Selektion und Deselektion ist identisch mit dem der vorherigen Seite. So wird der "SUBMIT" Button aktiv, sobald ein Getränk ausgewählt

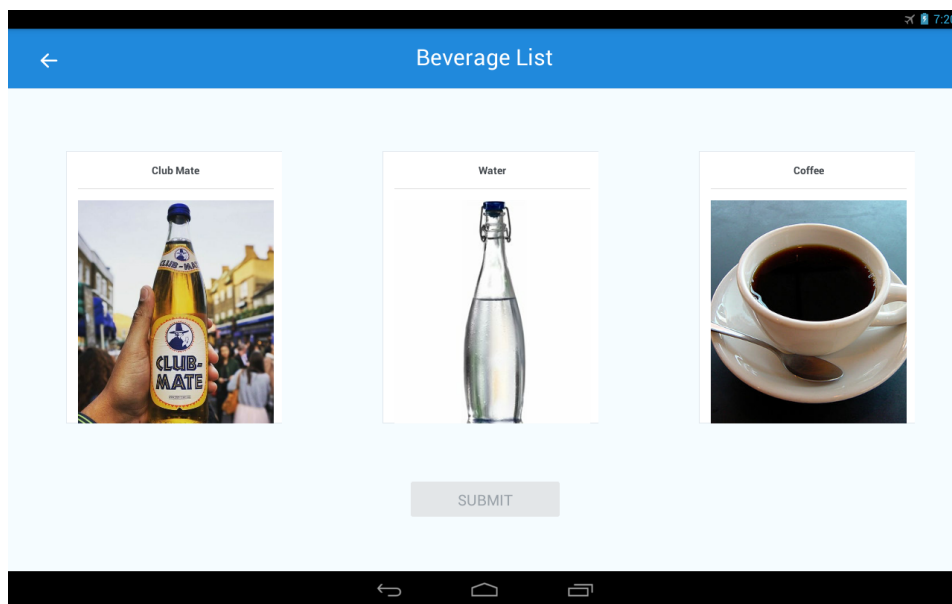


ABBILDUNG 3.9: Drinks Page: kein Getränk ausgewählt

ist und inaktiv wenn das Getränk wieder deselektiert wird (vgl. Abbildung 3.9).

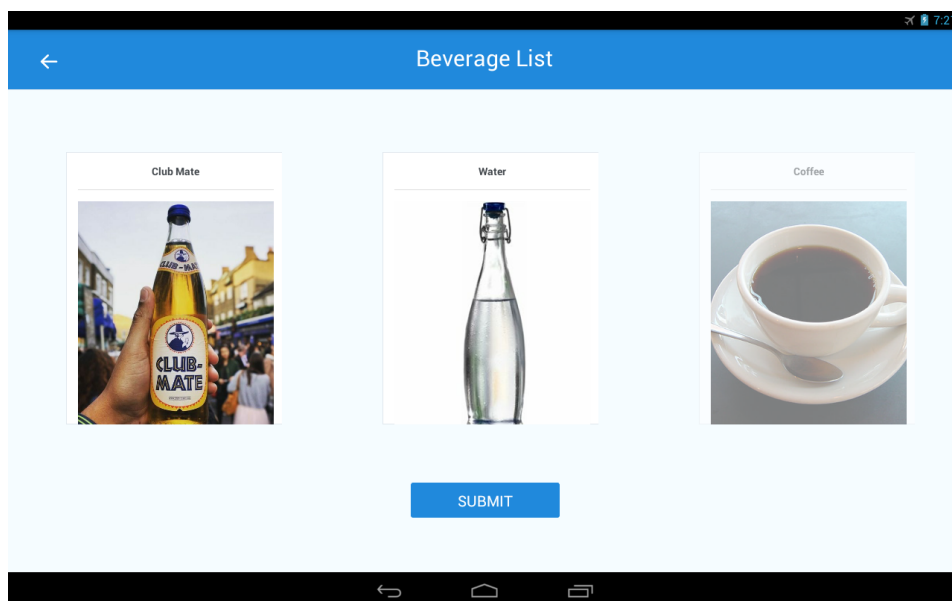


ABBILDUNG 3.10: Drinks Page: Getränk ausgewählt

### 3 System Architektur

Wird schließlich der “SUBMIT” Button gedrückt und die Transaktion als erfolgreich bestätigt erscheint folgendes Overlay:

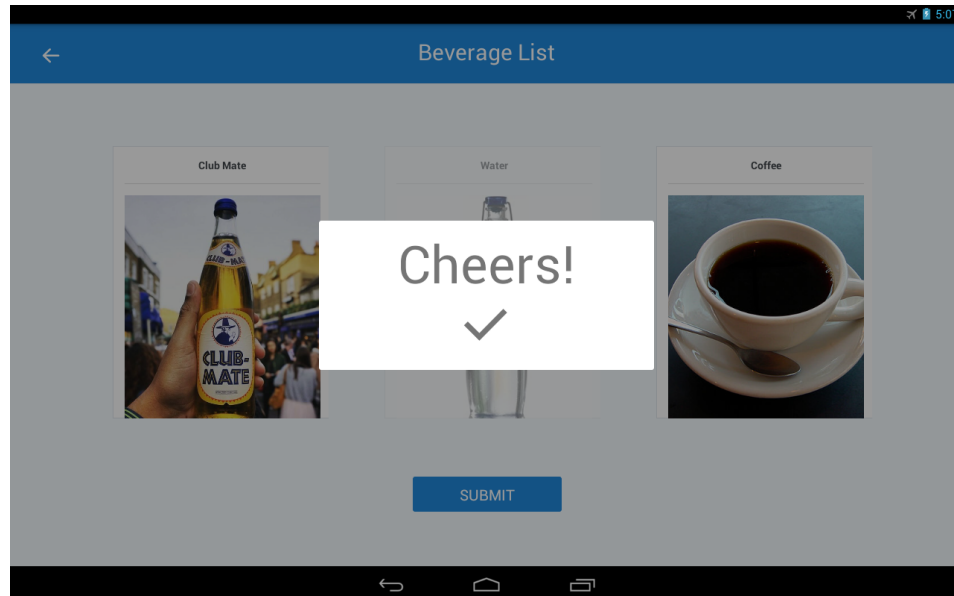


ABBILDUNG 3.11

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

Aufgrund der Tatsache, dass die Verbindung zum Uni-Netzwerk (eduroam) nur über einen Workaround hergestellt werden konnte, bei dem die Verbindung zum Netzwerk trotzdem nach einer unbestimmten Zeit immer wieder abgebrochen ist, wurde eine weitere Komponente entwickelt.

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 Verbin-

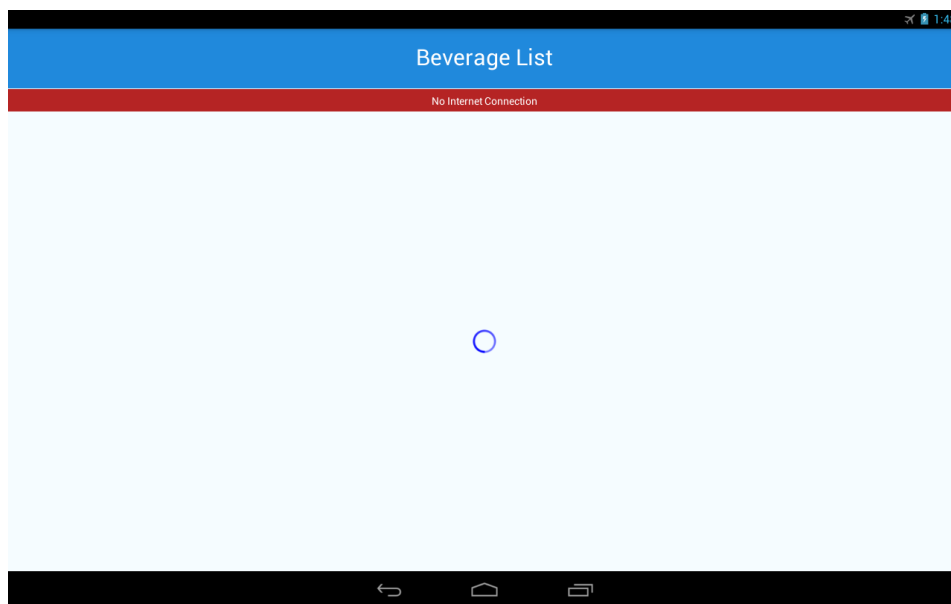


ABBILDUNG 3.12

derung 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

### 3 System Architektur

#### 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

#### 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, dabei werden zuerst die Smart Contract Files vom Fileserver runtergeladen und mit den beinhaltenden Daten das Web3.js Modul initialisiert. Damit steht Verbindung und die Kommunikation mit den Smart Contracts und es wird im Anschluss über alle vorhandenen Transaktion Files iteriert. Dabei wird zuerst der CoffeeCoin Contract und dann der Beveragelist Contract aufgerufen. Wenn beide ihre Transaktionen bestätigt haben, wird die Datei gelöscht, andernfalls bleibt diese 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 klickt “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 getrunkenen 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. Es wird zuerst ein “Gas Estimate” für beide Smart Contract

### 3 System Architektur

Transaktionen durchgeführt. Ein “Gas Estimate” ist, wie der Name bereits impliziert, eine grobe Schätzung wie viel Gas beim Funktionsaufruf eines Smart Contracts benötigt wird. Abschließend werden 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: *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 wird das gerade erstellte File wieder gelöscht. Somit soll sichergestellt werden, dass keine Transaktionen aufgrund von Verbindungsabbrüchen verloren gehen.

Anschließend erfolgt eine “Pseudo-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 dadurch 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 betrachten 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) in jedem Timeslot (T0-T4) unter Berücksichtigung einer vordefinierten Abweichung und Erwartungswert (siehe 4.1) die Anzahl der getrunkenen Kaffee ermittelt. Für jeden Kaffee wird zufällig (daraufhin) 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.

#### 4 Studie

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:

---

#### 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 selben (Wochen) Trainingsset konfrontiert wird. Stattdessen wird mit jeder neuen Episode ein Trainingsset erzeugt, welches komplett neue Uhrzeiten beinhaltet, 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 ca. 3,5 Wochen und umfasste 8 Teilnehmer vom Lehrstuhl. Der Versuchsaufbau entsprach dem der Systemarchitektur aus Kapitel 3.

Wurde also ein Getränk durch einen Mitarbeiter in der App ausgewählt und bestätigt, erfolgte eine Transaktion auf der Blockchain, welche wiederum vom Learner detektiert, gefiltert und die Daten zur Modellierung des Kaffeetrinkverhaltens verwendet wurde. Zudem wurde jede Prediction und dessen Evaluierung geloggt und 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, welche 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 Blockchain kommunizieren bzw. Transaktion auslösen zu können.

Durch einen Workaround (freies WLAN + VPN-Client) konnte das Tablet mit

## 4 Studie

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 wieder 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 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.

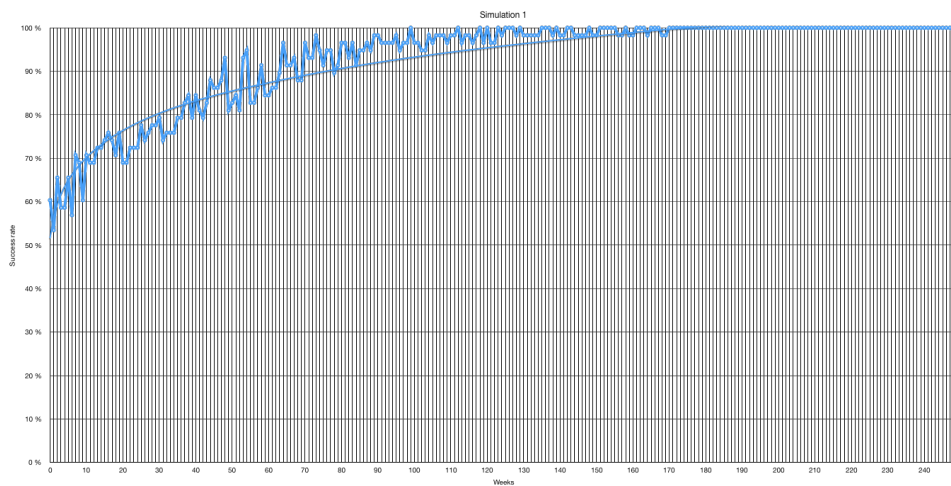


ABBILDUNG 4.1

#### Simulation 2

Der Graph der 2. Simulation 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 die Ergebnisse der Ersten lediglich nochmals untermauert werden

## 4 Studie

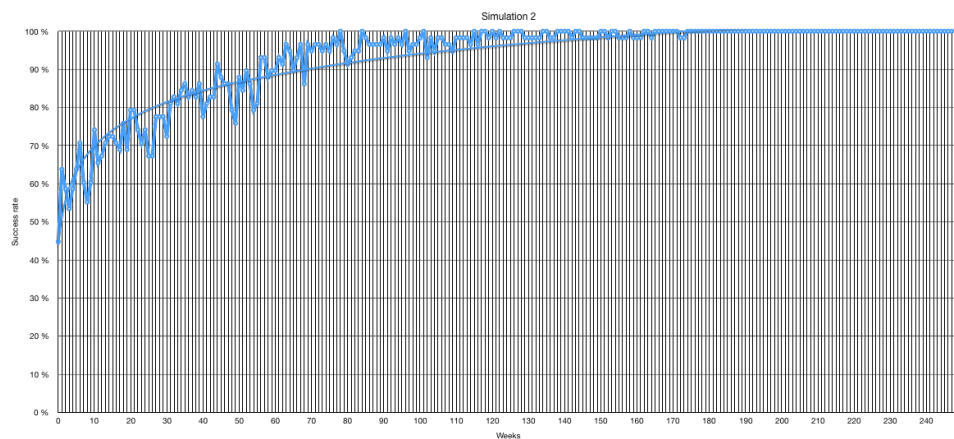


ABBILDUNG 4.2

und die einwandfrei Funktionalität der Implementierung des Q-Learning Algorithmus zeigen.

### 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 wurde.

### User 1

User 1 wurde erst in der zweiten Woche aktiv, weswegen die Teilnahmedauer nur 2,5 Wochen betrug. Betrachtet man den Graphen des Wochenüberlicks,



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, Freitag) ein Anstieg der Erfolgsrate um ca. 30% zu erkennen.

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

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

## 4 Studie

### User 2

User 2 war drei Wochen aktiv und startete am 4. Tag der ersten Woche. Bei der Wochenansicht (vgl. ??) ist in den Wochen 1-3 ein Anstieg von 55% auf 68% zu erkennen, auf welche ein Abfall auf 44% Erfolgsrate folgt.

Bis auf den dritten Tag ist in allen anderen Fällen mindestens einmal ein Anstieg der Erfolgsrate im Vergleich zur Vorwoche zu beobachten. Gerade der Graph des dritten Tages, 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 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 als auch von User 2 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 von 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.

Schlussendlich bedarf es einer weitaus längeren Studiendauer, um einen eindeutigen Beweis zu erlangen, welcher die Forschungsfrage in voller Gänze beantwortet.

# 5 Zusammenfassung

## 5.1 Weiterführende Forschungsfragen

- anderer algorithmus sarsa
- Decision Trees besseres ergebnis?
- offline learning from bchain logs -> reinforcement oder unsupervised
- algorithmus daten auch auf bchain speichern -> transparenz
- auf der testchain
- eos
- 
- 

## 5.2 Ausblick

# Literaturverzeichnis

- [Ant17] Andreas M. Antonopoulos. *Mastering Bitcoin: Programming the Open Blockchain*. O'Reilly Media, Inc., 2nd edition, 2017.
- [btc] btc-echo. Was ist proof-of-stake?
- [Bus18] Enée Bussac. Bitcoin, ethereum co. - praxiswissen kryptowährungen und blockchain. 2018.
- [DAp] Dapp.
- [Dav15] David Tuesta. Smart contracts: the ultimate automation of trust?, 2015-10-15.
- [ES18] Christoph Engemann and Andreas Sudmann, editors. *Machine Learning - Medien, Infrastrukturen und Technologien der Künstlichen Intelligenz*. Digitale Gesellschaft. transcript, Bielefeld, [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.

## *Literaturverzeichnis*

- [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.
- [Ste31] Greg Sterling. 90 percent use multiple screens during the same day, 2012-08-31.
- [Wei01] Gerhard Weiss, editor. *Multiagent systems*. MIT Press, Cambridge, Mass. [u.a.], 3. printing edition, 2001.
- [Wika] Wikipedia. Application programming interface - wikipedia.
- [Wikb] Wikipedia. Cross-platform software - wikipedia.
- [Wikc] Wikipedia. Cross-platform software - wikipedia.
- [Wikd] Wikipedia. Crud - wikipedia.
- [Wike] Wikipedia. Representational state transfer - wikipedia.
- [Wikf] Wikipedia. Single point of failure - wikipedia.
- [Wikg] Wikipedia. User experience - wikipedia.
- [Zhe18] Alice Zheng. *Feature engineering for machine learning*. O'Reilly, First edition ; Beijing, April 2018.

# Abbildungsverzeichnis

3.1	Systemarchitektur . . . . .	18
3.2	geth Befehl zum starten der Blockchain . . . . .	27
3.3	0x6ecbe1db9ef729cbe972c83fb886247691fb6beb-ql.json Der Name des JSON-Files setzt sich aus der Ethereum-Adresse des Users und der Abkürzung "ql", welches für Q-Learning steht, zusammenhängen. . . . .	35
3.4	genesis.json JSON-File welches zur Erstellung des Genesis Blocks verwendet wurde. . . . .	38
3.5	ERC-20 Interface . . . . .	40
3.6	CoffeeCoin Interface Auszug . . . . .	42
3.7	Mitarbeiter Page: kein Mitarbeiter ausgewählt . . . . .	54
3.8	Mitarbeiter Page: Mitarbeiter ausgewählt . . . . .	54
3.9	Drinks Page: kein Getränk ausgewählt . . . . .	55
3.10	Drinks Page: Getränk ausgewählt . . . . .	55
3.11	. . . . .	56
3.12	. . . . .	57
4.1	. . . . .	65
4.2	. . . . .	66



# Tabellenverzeichnis

3.1	Exmplarische Q-Tabelle . . . . .	47
-----	----------------------------------	----





# **A Anhang A**



## **B Anhang B**