

# FH Aachen

**Fachbereich Elektrotechnik und Informationstechnik**

Studiengang Informatik

Bachelorarbeit

## **Integration eines eingebetteten Systems in eine Cloud-Infrastruktur am Beispiel eines autonomen Schachtischs**

vorgelegt von

**Marcel Werner Heinrich Friedrich Ochsendorf**

Matrikel-Nr. **3120232**

Referent:

Prof. Dr.-Ing. Thomas Dey

Korreferent:

Prof. Dr. Andreas Claßen

Datum:

14.06.2021

# **Erklärung**

Hiermit erkläre ich, dass ich die vorliegende Arbeit eigenständig und ohne fremde Hilfe angefertigt habe. Textpassagen, die wörtlich oder dem Sinn nach auf Publikationen oder Vorträgen anderer Autoren beruhen, sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Aachen, den 11.06.2021 \_\_\_\_\_

# **Abstract**

Die Kurzfassung gibt auf ein bis zwei Seiten die wesentlichen Inhalte und Ergebnisse der Abschlussarbeit wieder.

Sie gliedert sich inhaltlich in

- das behandelte Gebiet,
- das Zieler Arbeit,
- die Untersuchungsmethode,
- die Ergebnisse und
- die Schlussfolgerungen.

Die Kurzfassung enthält keine Schlussfolgerungen oder Bewertungen, die über die Inhalte der Kapitel der Arbeit hinausgehen.

Alle Aussagen der Kurzfassung finden sich in ausführlicher Form in der Arbeit wieder.  
Die Kurzfassung erhält keine Kapitelnummer.

## Inhalt

0.1	Einleitung	1
0.1.1	Motivation	1
0.1.2	Zielsetzung	2
0.1.3	Methodik	3
0.2	Analyse bestehender Systeme und Machbarkeitsanalyse	4
0.2.1	Existierende Systeme im Vergleich	4
0.2.2	User Experience	9
0.2.3	Anforderungsanalyse	10
0.2.4	Machbarkeitsanalyse	11
0.3	Grundlegende Verifikation der ausgewählten Technologien	11
0.3.1	Erprobung Buildroot-Framework	11
0.3.2	Verifikation NFC Technologie	12
0.3.3	Schrittmotor / Schrittmotorsteuerung	12
0.3.4	3D Druck für den mechanischen Aufbau	13
0.4	Erstellung erster Prototyp	14
0.4.1	Mechanik	14
0.4.2	Parametrisierung Schachfiguren	16
0.4.3	Schaltungsentwurf	18
0.5	Erstellung des zweiter Prototypens	21
0.5.1	Modifikation der Mechanik	21
0.5.2	Optimierungen der Spielfiguren	21
0.5.3	Änderungen der Elektronik	22
0.5.4	Fazit bezüglich des finalen Prototypens	29
0.6	Entwicklung der Cloud Infrastruktur	30
0.6.1	API Design	30
0.6.2	Service Architektur	32
0.6.3	Vorüberlegungen	32
0.6.4	Backend	32
0.6.5	MoveValidator	34
0.6.6	Entwicklung Webclient	36

0.6.7 AutoPlayer . . . . .	37
0.7 Embedded System Software . . . . .	39
0.7.1 Ablaufdiagramm . . . . .	39
0.7.2 Figur Bewegungspfadberechnung . . . . .	41
0.7.3 Inter Prozess Communication . . . . .	42
0.7.4 Userinterface . . . . .	43
0.8 Fazit . . . . .	47
0.8.1 Ausblick . . . . .	48
<b>Abbildungsverzeichnis</b>	<b>49</b>
<b>Tabellenverzeichnis</b>	<b>50</b>
.1 Anhang . . . . .	51

## 0.1 Einleitung

### 0.1.1 Motivation

Eingebettete Systeme (Englisch “embedded Systems”) sind technische Zusammensetzungen, welche für eine spezifische Funktion entwickelt werden. Im Gegensatz zu Mehrzweksystemen (Englisch “multi-purpose systems”), wie zum Beispiel einem Personal Computer, welcher in der Lage ist, diverse Funktionen auszuführen und nicht zwingend an eine Funktion gebunden ist, dienen eingebettete Systeme einer bestimmten Logik. Daraus resultieren einfacherer und auch Ressourcen-sparendere Systeme, die wesentlich näher an der Technik und der für den Zweck nötigen Komponenten und Software entwickelt werden. Systeme können grünster zusammengesetzt und Fehlerquellen schneller entdeckt und behoben werden. Nicht für den Prozess notwendige Komponenten werden gar nicht erst verwendet. Bei einem Mehrzweksystem wird akzeptiert, dass Komponenten und Schnittstellen existieren, die nicht benötigt werden. Diese verursachen Kosten und können mögliche Fehlerquellen sein.

Dennoch ist die Entwicklung eines solchen Systems nicht banal. Es ist abzuwägen, welche Komponenten derzeit auf dem freien Markt erhältlich sind, welche Eigenschaften diese mitbringen oder ermöglichen und wie diese optimal kombiniert werden können. Es bedarf im Vorhinein intensiverer Recherche und einer größeren Perspektive über mögliche Zusammenhänge. Im Falle eines Merhzweksystems ist die Auswahl simpler, da man den Prozess auch im Nachhinein noch anpassen kann, da zusätzliche Funktionen und Komponenten gegeben sind oder leichter ergänzt werden können. Das eingebettete System muss in der Regel aufgewertet oder sogar völlig ersetzt werden, wenn zu einem späteren Zeitpunkt festgestellt wird, dass Funktionen nicht gegeben oder umsetzbar sind. Fertige Systeme sind komplexer in der Aufwertung.

Die Fähigkeit zur Erstellung eines solchen System ist daher nicht leichtfertig anzunehmen und es ist mir wichtig, zum Abschluss meines Studiums mein gewonnenes Wissen über Systeme, Komponenten, Zusammenhänge und deren Verbindung bis hin zur Programmierung nachzuweisen. Die Auswahl eines fertigen Computers oder sogar das simple Nutzen existierender Betriebssysteme erweckt nicht den gleichen Reiz, wie es die eigene Erstellung dieser Komponenten auf mich hat. Ich halte es für essenziell, möglichst fachlich die Inhalte meines Studiums in Verbindung mit meinen Vorlieben zu bringen, um ein optimales Projekt zu erstellen.

Die Erstellung eines autonomen Schachttischs vereinbart in meinen Augen im großen

Umfang die wesentlichen Komponenten des Informatikstudiums mit meiner Vorliebe zur mechanisch-elektrischen Gestaltung. Angefangen mit den Grundlagen der Informatik, insbesondere mit technischem Bezug, über die Berechnung und Auslegung von Systemkomponenten, zudem die objektorientierte Projektplanung und Architektur von Systemen bis hin zu Datenbanken und Webtechnologien und Softwarenentwicklung. Zudem wird mein Studienschwerpunkt, die technische Informatik, mit einem eingebetteten System manifestiert.

Der Reiz im Schachprojekt liegt in der Bedeutung und der Seltenheit. Schach ist ein bewährtes, ausnahmslos bekanntes und immer logisches Spiel, welches jedoch im kommerziellen Rahmen nie an Bedeutung gewonnen hat. Die Auswahl der verfügbare elektrifizierte und programmgesteuerte Schachtisch ist auffallend gering; zudem sind existierende Lösungen oftmals nicht erschwinglich und bedürfen erhebliche Anpassungen des Spielers an das Spiel. Innerhalb der vergangenen drei Jahrzehnte bewiesen sich immer mehr Konzerne ihre technische Kompetenz und Überlegenheit und die Fähigkeit ihrer Maschinen mittels der Auswertung von Schachalgorithmen und dem möglichst schnellen besiegen derzeitiger Schach-Meister und -Meisterinnen. Die Algorithmen stehen heute in einer Vielzahl frei zugänglich zur Verfügung, jedoch ist das Interesse daran, für Spieler mögliche Anwendungen zu generieren, verschwindend gering und wird oftmals nur von Experten und Enthusiasten genutzt und auch hinterfragt.

Mit dieser Arbeit möchte ich mich diesem Problem stellen und einen möglich günstigen Tisch entwickeln, welcher das Spielerlebnis ohne Einschränkungen dem Spieler transferiert. Zudem möchte ich gewonnene Erkenntnisse und aktuelle Ressourcen wie die Cloud-Infrastruktur einbinden, um das Schachspiel, welches zweier Spieler bedarf, für einen Spieler zu ermöglichen. Das Ergebnis soll nicht nur viele Zeilen Code sein, sondern auch ein handfestes Produkt, dass meine Qualitäten und Enthusiasmus widerspiegeln.

### 0.1.2 Zielsetzung

Das Ziel der nachfolgenden Arbeit ist es, einen autonomen Schachtisch zu entwickeln, welcher in der Lage ist, Schachfiguren autonom zu bewegen und auf Benutzerinteraktionen zu reagieren.

Der Schwerpunkt liegt dabei insbesondere auf der Programmierung des eingebetteten Systems und dem Zusammenspiel von diesem mit einem aus dem Internet erreichbaren Servers, welcher als Vermittlungsstelle zwischen verschiedenen Schachtischen und

anderen Endgeräten dient. Dieses besteht zum einem aus der Positionserkennung und Steuerung der Hardwarekomponenten (Schachfiguren) und zum anderen aus der Kommunikation zwischen dem Tisch selbst und einem in einer Cloud befindlichem Server. Mittels der Programmierung werden diverse Technologien von verschiedenen Einzelsystemen zu einem Gesamtprodukt zusammengesetzt. Insgesamt gilt es, einen für Anwender ansprechenden Schachtisch zu entwickeln, der das Spielerlebnis nicht nur originalgetreu widerspiegelt, sondern das Einzelspieler-Modell zusätzlich noch verbessert.

### 0.1.3 Methodik

Im ersten Abschnitt werden die zum Zeitpunkt existierenden Ansätze und deren Umsetzung beleuchtet. Hier wurde insbesondere darauf geachtet, die Grenzen bestehender Systeme darzulegen und auf nur für dieses Projekt zutreffende Funktionen zu vergleichen. Anschließend werden die zuvor verwendeten Technologien betrachtet, welche bei den beiden darauffolgenden Prototypen verwendet wurden. Hierbei stehen insbesondere solche Technologien im Vordergrund der Untersuchung, welche möglichst einfach zu beschaffen sind und optimaler Weise uneingeschränkt und lizenzunabhängig zur Verfügung stehen.

Das sechste Kapitel widmet sich der Realisierung eines ersten Prototyps des autonomen Schachtisches. Hier werden die Erkenntnisse der zuvor evaluierten Technologien verwendet, um ein Modell zu entwickeln, welches den im ersten Abschnitt erarbeiteten Vorgaben entspricht. Der nach der Implementierung durchgeführte Dauertest soll zudem weitere Risiken, mögliche Probleme und Fehlerquellen aufdecken.

Im anschließenden Kapitel wird auf der Basis des ersten Prototypens und dessen im Betrieb verzeichneten Probleme der finale Prototyp entwickelt.

Hier werden die Schwierigkeiten durch die Vereinfachung der Elektronik sowie der Mechanik gelöst. Die Zuverlässigkeit wurde mittels stetiger Testläufe mit kontrollierten Schachzug-Szenarien überwacht und so ein produktreifer Prototyp entwickelt.

Im darauffolgenden Abschnitt wird die Cloud-Infrastruktur thematisiert, welche für eine Kommunikation zwischen den autonomen Schachtischen entscheidend ist. Auch wird dabei die Software, welche auf dem eingebetteten System ausgeführt wird, im Detail beschrieben und deren Kommunikation mit der Cloud-Infrastruktur, sowie mit den elektrischen Komponenten beleuchtet.

## 0.2 Analyse bestehender Systeme und Machbarkeitsanalyse

### 0.2.1 Existierende Systeme im Vergleich

Im Folgenden werden vier kommerzielle und drei lizenzenabhängige (Open-Source) Schachtische miteinander verglichen. Bei den ausgewählten Tischen handelt es sich um

- Square Off - Kingdome
- Square Off - Grand Kingdom
- DGT Smartboard
- DGT Bluetooth Wenge
- Automated Chessboard (Michael Guerero)
- Automated Chessboard (Akash Ravichandran)
- DIY Super Smart Chessboard

Für die kommerziell käuflichen Schachspiele gibt es kein sehr großes Marktangebot, weswegen für den Vergleich nur zwei Hersteller mit jeweils zwei verschiedenen Modellen gewählt werden konnte. Derzeit integriert nur ein Unternehmen eine Funktion, welche die Figuren unterhalb der Tischplatte mechanisch bewegen kann. Der zweite Hersteller wurde dennoch zum Vergleich der zusätzlichen Funktionen herangezogen.

Die Tische eines Herstellers unterscheiden sich kaum in ihren Funktionen; mit steigendem Preis werden zusätzliche Funktionen in Form von Sensoren oder Verbindungsoptionen implementiert.

Das Angebot lizenzenfreier Produkte hingegen ist signifikanter, jedoch sind die einzelnen Modelle oftmals Kopien oder Revisionen voneinander. Die möglichen Funktionen unterscheiden sich daher kaum. Für die hier dargestellte Übersicht wurden drei Modelle gewählt, welche in ihren Funktionen signifikante Auffälligkeiten und einen hohen Stellenwert und Bekanntheitsgrad aufweisen. Wie bereits aus zum Teil identischen den Namen ersichtlich, streben alle Tische das gleiche Ziel an und unterscheiden sich daher nur in geringen Funktionen, was im Folgenden nun näher erläutert wird.

#### Kommerzielle Produkte

Tabelle 0.1: Auflistung kommerzieller autonomer Schachtische

	Square Off - Kingdom [?]	Square Off - Grand Kingdom [?]	DGT Smart Board [?]	DGT Bluetooth Wenge [?]
Erkennung Figur-Stellung	nein (Manuell per Ausgangsposition)	nein (Manuell per Ausgangsposition)	ja	ja
Abmessungen (LxBxH)	486mm x 486mm x 75mm	671mm x 486mm x 75mm	540mm x 540mm x 20mm	540mm x 540mm x 20mm
Konnektivität	Bluetooth	Bluetooth	Seriell	Bluetooth
Automatisches Bewegen der Figuren	ja	ja	nein	nein
Spiel Livestream	ja	ja	ja	ja
Cloud-Anbindung (online Spiele)	ja (Mobiltelefon + App)	ja (Mobiltelefon + App)	ja (PC + App)	ja (PC + App)
Parkposition für ausgeschiedene Figuren	nein	ja	nein	nein
Stand-Alone Funktionalität	nein (Mobiltelefon erforderlich)	nein (Mobiltelefon erforderlich)	nein (PC)	nein (PC)
Besonderheiten	Akku für 30 Spiele	Akku für 15 Spiele	-	-

Die für den Vergleich gewählten Eigenschaften sind jene, welche die im Projekt angestrebten Funktionen möglichst äquivalent reflektieren. Dennoch schränkt das geringe Angebot an autonomen Tischen die Auswahl stark ein; daher wurde hierbei wertgelegt auf Automation, Cloud-Anbindung und die Abmessungen, welche das Spielerlebnis am deutlichsten beeinflussen.

Die Bretter des Herstellers DGT erkennen die Position der verwendeten Figuren; eine Auskunft, über die die verwendete Technologie erhält, man jedoch nicht. Die Square-

Off-Schachtische verfügen über keine solche Funktion.

Die Abmessungen unterscheiden sich nur beim Hersteller Square Off deutlich; der Grand Kingdom Schachtisch ist rechteckig konstruiert worden, was das Spielerlebnis deutlich verändert. Der simple Kingdom-Tisch wiederum ist kleiner als das vorgegebene Turniermaß, was ebenfalls Einfluss auf das Spielererlebnis hat. Mit den Standardmaßen der DGT-Spielbretter und zudem ihrer geringen Höhe gleichen diese deutlich einem Turniertisch. Die Kombination aus geringer Höhe und Erkennung der Figur-Stellung bei den DGT-Brettern ist auffallend.

Alle Hersteller bieten eine Bluetooth-Schnittstelle an, einzig das Smart-Board des Herstellers DGT nutzt eine serielle, kabelgebundene Schnittstelle.

Bei den DGT-Schachbrettern ist zu beachten, dass diese die Schachfiguren nicht autonom bewegen können. Sie wurden jedoch in die Liste aufgenommen, da diese einen Teil der Funktionalitäten der Square Off Schachbrettern abdecken und lediglich die automatische Bewegung der Schachfiguren fehlt. Die DGT-Bretter können die Position der Figuren erkennen und ermöglichen so auch Spiele über das Internet; diese können sie auch als Livestream anbieten. Bei Schachturnieren werden diese für die Übertragung der Partien sowie die Aufzeichnung der Spielzüge verwendet und bieten Support für den Anschluss von weiterer Peripherie wie z.B. Schachuhren.

Somit gibt es zum Zeitpunkt der Recherche nur einen Hersteller von autonomen Schachbrettern, welcher auch die Figuren bewegen kann.

Ein Spiel-Livestream, eine Darstellung die aktuellen oder vergangenen Spiele über eine Webanwendung, ist mit allen Tischen möglich. Da alle Tische eine Cloud-Anbindung besitzen, in der Regel mittels Applikation auf dem Smartphone oder Computer, wird lediglich das Versetzen von Figuren detektiert und in einer Oberfläche dargestellt.

Auffallend ist, dass nur einer der ausgewählten Tische über eine Parkposition für ausgeschiedene Figuren verfügt. Der Square-Off-Grand, welche Figuren automatische verschieben kann, besitzt dank der rechteckigen Tischform die Möglichkeit, Figuren selbstständig aus dem Spiel zu entfernen und bei Bedarf wieder ins Spiel zurückzuführen.

Ebenfalls erwähnenswert ist, dass keiner der Tische eine Stand-Alone-Funktionalität besitzt. Jeder Tisch benötigt eine Verbindung zu einem externen Gerät, wie einem Smartphone oder Computer, welche die Berechnungen der Spielerzüge vornimmt. Kei-

ner dieser Tische kann ein simples Spiel nach einem verbindungslosen Start ausführen. Ohne Internet verlieren die Tische all ihre Funktionalität.

Beide Square-Off-Modelle verfügen zudem über die Möglichkeit des Spiels ohne Energieversorgung, welche durch eingebaute Akkus ermöglicht werden. Diese Funktionalität sorgt für eine zusätzliche Nutzerzufriedenheit.

Zusammenfassend ist festzustellen, dass alle vier Tische dank unterschiedlicher Ausführung von Spiel-Eigenschaften zu unterschiedlichen Spiel-Erlebnissen führen. Für Nutzer ist eine Entscheidung anhand von Funktionen kaum möglich; letztlich bedarf es der Auswertung von gewünschten und gegebenen Funktionen. Das Ziel soll nun sein, all die positiven Eigenschaften dieser Tische zu vereinbaren und mittels noch zusätzlicher Verbesserungen ein eigenes Produkt zu entwickeln.

### **Open-Source Projekte**

Bei allen Open-Source Projekten wurden die Eigenschaften anhand der Beschreibung und der aktuellen Software extrahiert.

Besonders bei Projekten, welche sich noch in der Entwicklung befinden, können sich die Eigenschaften noch verändern und so weitere Funktionalitäten hinzugefügt werden. Alle Eigenschaften der Projekte wurden zum Zeitpunkt der Recherche analysiert und dokumentiert und mit Beginn der Entwicklung als Struktur-Fixpunkt festgelegt. Nachfolgende Entwicklungen werden zu diesem Zeitpunkt nicht mehr berücksichtigt.

Zusätzlich zu den genannten Projekten sind weitere derartige Projekte verfügbar; in der Tabelle wurde nur jene aufgelistet, welche sich von anderen Projekten in mindestens einem Feature unterscheiden.

Auch existieren weitere Abwandlungen von autonomen Schachbrettern, bei welchem die Figuren von oberhalb des Spielbretts gegriffen bzw. bewegt werden. In einigen Projekten wird dies mittels eines Industrie-Roboters [?] oder eines modifizierten 3D-Druckers[?] realisiert. Diese wurden hier aufgrund der Mechanik, welche über dem Spielbrett montiert werden muss, nicht berücksichtigt.

Tabelle 0.2: Auflistung von Open-Source Schachtisch Projekten

	Automated Chess Board (Michael Guerero) [?]	Automated Chess Board (Akash Ravichandran) [?]	DIY Super Smart Chessboard [?]
Erkennung Figur-Stellung	nein (Manuell per Ausgangsposition)	ja (Kamera / OpenCV)	nein
Abmessungen (LxBxH)	keine Angabe	keine Angabe	450mm x 300mm x 50mm
Konnektivität	Universal Serial Bus (USB)	Wireless Local Area Network (WLAN)	WLAN
Automatisches Bewegen der Figuren	ja	ja	nein
Spiel Livestream	nein	nein	nein
Cloud-Anbindung (online Spiele)	nein	nein	ja
Parkposition für ausgeschiedene Figuren	nein	nein	nein
Stand-Alone Funktionalität	nein (PC erforderlich)	ja	ja
Besonderheiten	-	Sprachsteuerung (Amazon Alexa)	Zuganzeige über LED Matrix
Lizenz	General Public License (GPL) 3+	GPL	-

In den bestehenden Projekten ist zu erkennen, dass ein autonomer Schachtisch sehr einfach und mit simplen Mittel konstruiert werden kann. Hierbei fehlen in der Regel einige Features, wie das automatische Erkennen von Figuren oder das Spielen über das Internet.

Einige Projekte setzen dabei auf eingebettete Systeme, welche direkt im Schachtisch montiert sind, andere hingegen nutzen einen externen PC, welcher die Steuerbefehle an die Elektronik sendet.

Bei der Konstruktion der Mechanik und der Methode, mit welcher die Figuren über das

Feld bewegt werden, ähneln sich jedoch die meisten dieser Projekte. Hier wurden in der Regel einfache X- und Y-Achse verwendet, welche von je einem Schrittmotoren bewegt werden. Die Schachfiguren werden dabei mittels eines Elektromagneten über die Oberseite gezogen. Indes ist ein Magnet oder eine kleine Metallplatte als Gegenpol in den Fuß der Figuren eingelassen worden.

Die Erkennung der Schachfiguren ist augenscheinlich die schwierigste Aufgabe. Hier wurde in der Mehrzahl der Projekte eine Kamera im Zusammenspiel mit einer auf OpenCV basierenden Figur-Erkennung verwendet. Diese Variante ist je nach Implementierung des Vision-Algorithmus fehleranfälliger bei sich ändernden Lichtverhältnissen, auch muss die Kamera oberhalb der Schachfiguren platziert werden, wenn kein transparentes Schachfeld verwendet werden soll.

Eine weitere Alternative ist die Verwendung einer Matrix aus Reed-Schaltern oder Halleffekt-Sensoren. Diese werden in einer 8x8 Matrix Konfiguration unterhalb der Platte montiert und reagieren auf die Magnete in den Figuren. So ist es möglich zu erkennen, welches der Schachfelder belegt ist, jedoch nicht konkret von welchem Figur Typen. Dieses Problem wird durch eine definierte Ausgangsstellung beim Spielstart gelöst. Nach jedem Zug durch den Spieler und der dadurch resultierenden Änderungen in der Figur Positionen in der Matrix können die neuen Figur Stellungen berechnet werden.

### 0.2.2 User Experience

Ein wichtiger Aspekt bei diesem Projekt stellt die User-Experience dar. Diese beschreibt die Ergonomie der Mensch-Maschine-Interaktion und wird durch die DIN 9241[?] beschrieben. Darin geht es primär um das Erlebnis, welches der Benutzer bei dem Verwenden eines Produktes erlebt und welche Erwartungen der Benutzer an die Verwendung des Produktes hat.

Bei dem autonomen Schachtisch soll der Benutzer eine ähnlich authentische Erfahrung erleben wie bei einer Schachpartie mit einem menschlichen Gegenspieler. Der Benutzer soll direkt nach dem Einschalten des Tisches und dem Aufstellen der Figuren in der Lage sein, mit dem Spiel beginnen zu können. Dies soll wie ein reguläres Schachspiel ablaufen; der Spieler vor dem Tisch soll die Figuren mit der Hand bewegen können und der Tisch soll den Gegenspieler darstellen. Dieser bewegt die Figuren der Gegenseite.

Nach Beendigung einer Partie soll das Spielbrett wieder in die Ausgangssituation

gebracht werden. Dies kann zum einem vom Tisch selbst oder vom Benutzer manuell geschehen. Danach ist der Tisch für die nächste Partie bereit, welche einfach per Knopfdruck gestartet werden können sollte.

Dies soll auf für abgebrochene Spiele gelten, welche von Benutzer oder durch das System abgebrochen werden. Indessen soll das Schachbrett sich ebenfalls selbstständig zurücksetzen können.

Ein weiter Punkt, welcher bei der User-Experience beachtet werden soll, ist die zeitliche Konstante. Ein Spiel auf einem normalen Schachspiel hat je nach Spielart kein Zeitlimit, dies kann für das gesamte Spiel gelten oder auch für die Zeit zwischen einzelnen Zügen. Der autonome Schachtisch soll es dem Spieler z.B. ermöglichen ein Spiel am Morgen zu beginnen und dieses erst am nächsten Tag fortzusetzen.

Auch muss sich hier die Frage gestellt werden, was mit den ausgeschiedenen Figuren geschieht. Bei den autonomen Schachbrettern von Square Off[?], werden die Figuren an die Seite auf vordefinierte Felder bewegt und können so wieder bei der nächsten Partie vom System aufgestellt werden. Viele andere Projekte schieben die Figuren auf dem Feld heraus, können diese aber im Anschluss nicht mehr gezielt in das Feld zurückholen. So muss diese Aufgabe vom Benutzer geschehen. Auch wir diese Funktionalität von einigen Projekten nicht abgedeckt und der Benutzer muss die Figuren selbstständig vom Feld entfernen.

### 0.2.3 Anforderungsanalyse

Nach Abschluss der Recherche, kann somit eine Auflistung aller Features angefertigt werden, welche ein autonomer Schachtisch aufweisen sollte. In diesem Projekt werden vor allem Funktionalitäten berücksichtigt, welche die Bedienung und Benutzung des autonomen Schachtisches dem Benutzer einen Mehrwert in Bezug auf die Benutzerfreundlichkeit bieten.

Tabelle 0.3: Auflistung der Anforderungen an den autonomen Schachtisch

Atomic Chess Table (ATC)	
Erkennung Figur-Stellung	ja
Konnektivität	WLAN, USB
Automatisches Bewegen der Figuren	ja
Spiel Livestream	ja

	ATC
Cloudanbindung (online Spiele)	ja
Parkposition für ausgeschiedene Figuren	ja
Stand-Alone Funktionalität	ja (Bedienung direkt am Tisch)
Besonderheiten	visuelle Hinweise per Beleuchtung

Die Abmessungen und das Gewicht des autonomen Schachtisches ergeben sich aus der mechanischen Umsetzung und werden hier aufgrund der zur Verfügung stehenden Materialen und Fertigungstechniken nicht festgelegt. Dennoch wird Wert darauf gelegt, dass das Verhältnis zwischen den Spielfeldabmessungen und den Abmessungen des Tisches so gering wie möglich ausfällt. Auch müssen die Figuren für den Benutzer eine gut handhabbare Größe aufweisen, um ein angenehmes haptisches Spielerlebnis zu gewährleisten. Ebenfalls wird kein besonderes Augenmerk auf die Geschwindigkeit der Figur-Bewegung gelegt, da hier die Zuverlässigkeit und Wiederholgenauigkeit dieser im Vordergrund stehen.

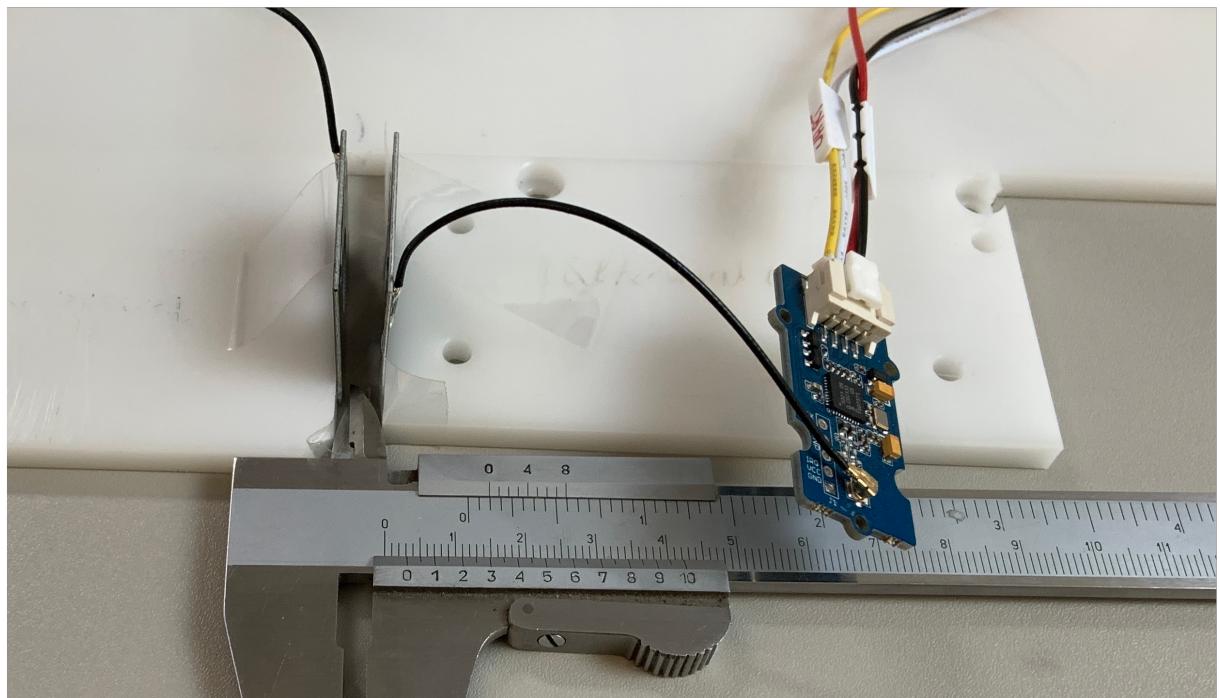
### **0.2.4 Machbarkeitsanalyse**

- welche technologien werden benötigt
- software architektur anforderungen
- hardware anforderungen
- grosse
- wiederholgenauigkeit
- lautstärke
- vorerfahrung in cad ed druck und schaltungsdesign

## **0.3 Grundlegende Verifikation der ausgewählten Technologien**

### **0.3.1 Erprobung Buildroot-Framework**

- Erstellen eines einfachen images für das embedded System



**Bild 0-1:** Grove PN532 NFC Reader mit Kabelgebundener Antenne

- inkl ssh Server und SFTP
- qt 5 libraries
- eigenes package atctp
- test der toolchain

### 0.3.2 Verifikation NFC Technologie

- warum gewählter nfc reader => ndef lesen
- reichweiten test mit 22mm
- test mit benachbarten figuren
- warum kein RFID => keine speicherung von id auf der controller seite
- selbherstellung von eigenen figuren ohne modifikation der controllerseite
  
- test mit figuren nebeneinander

### 0.3.3 Schrittmotor / Schrittmotorsteuerung

- warum => einfache ansteuerung

- keine STEP DIR somit muss embedded nicht echtzeitfähig sein und kann ggf auch andere task abarbeiten
- TMC schrittmotortreiber spi configuration
- und goto move => wait for move finished irw testen
- dafür einfacher python testreiber geschribene
- schrittverlust nicht zu erwarten

### 0.3.4 3D Druck für den mechanischen Aufbau

Da es sich hier nur um einen Prototyp handelt, wurde hier auf ein einfach zu verarbeitendes Filament vom Typ Polylactic Acid (PLA) zurückgegriffen. Dieses ist besonders gut für die Prototypenentwicklung geeignet und kann mit nahezu jeden handelsüblichen Fused Deposition Modeling (FDM) 3D-Drucker verarbeitet werden.

Zuvor wurden einige Testdrucke durchgeführt, um die Qualität der zuvor gewählten Druckparameter zu überprüfen und diese gegebenenfalls anzupassen. Auch wurden verschiedene weitere Bauteile gedruckt, an welchen die Toleranzen für die späteren Computer-Aided Design (CAD) Zeichnungen abgeschätzt werden können. Dies betrifft vor allem die Genauigkeit der Bohrungen in den gefertigten Objekten, da hier später Bolzen und Schrauben ein nahezu spielfrei eingeführt werden müssen. Ein Test, welcher die Machbarkeit von Gewinden zeigt, wurde nicht durchgeführt, da alle Schrauben später mit der passenden Mutter gesichert werden sollen. So soll eine Abnutzung durch häufige Montage der gedruckten Bauteile verhindert werden.

Bei dem Design der zu druckenden Bauteile wurde darauf geachtet, dass diese den Bauraum von 200x200x200mm nicht überschreiten und somit auch von einfachen FDM 3D-Druckern erstellt werden können.

Als Software wurde der Open-Source Slicer Ultimaker Cura [?] verwendet, da dieser zum einen bereits fertige Konfigurationen für den verwendeten 3D-Drucker enthält und zum anderen experimentelle Features bereitstellt.

Hier wurde für die Bauteile, welche eine Stützstruktur benötigen, die von Cura bereitgestellte Tree Support Structure aktiviert. **0-2** Diese bietet den Vorteil gegenüber anderen Stützstrukturen, dass sich diese leichter entfernen lässt und weniger Rückstände an den Bauteilen hinterlässt. Diese Vorteile wurde mit verschiedenen Testdrucken verifiziert und kommen insbesondere bei komplexen Bauteilen mit innenliegenden Elementen zum Tragen, bei denen eine Stützstruktur erforderlich sind.

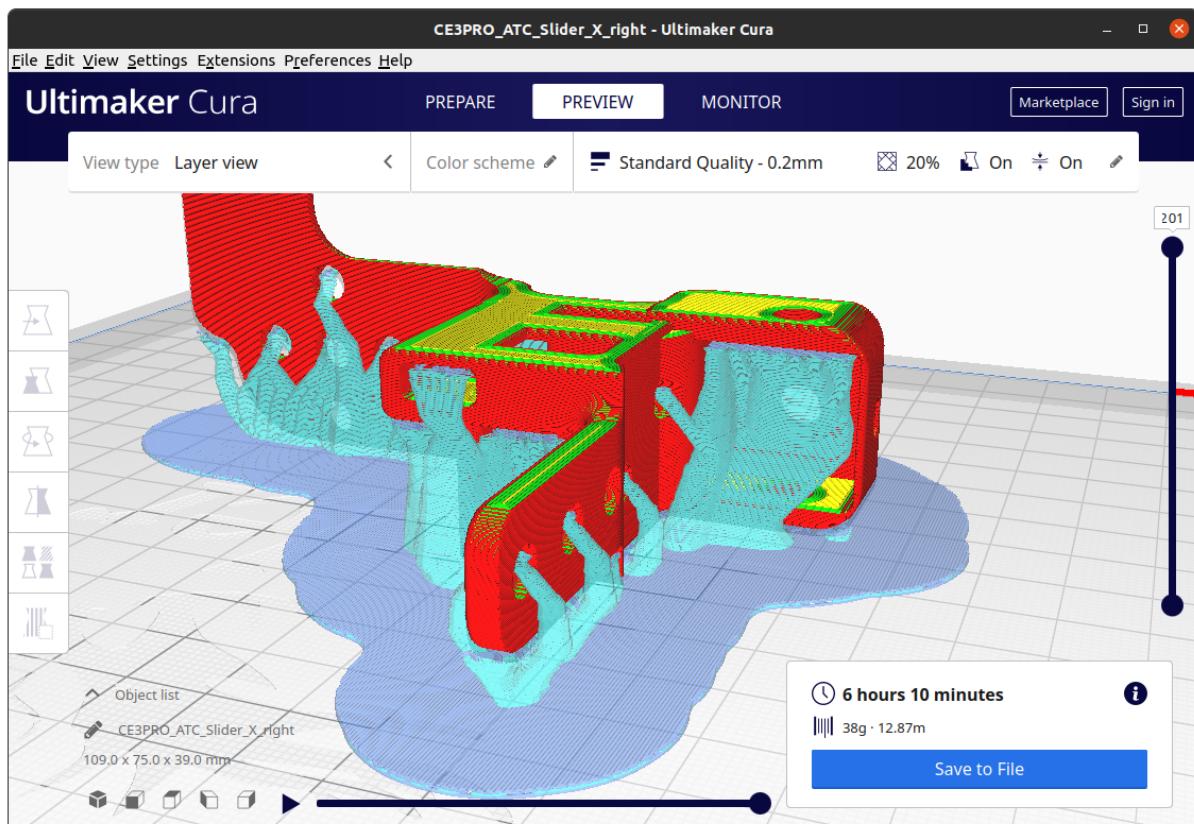


Bild 0-2: 3D Druck: Objekt (rot,gelb,grün),Tree Structure (cyan)

Tabelle 0.4: Verwendete 3D Druck Parameter. Temperatur nach Herstellerangaben des verwendeten PLA Filament.

Ender 3 Pro 0.4mm Nozzle	PLA Settings
Layer Height	0.2mm
Infill	50.00%
Wall Thickness	2.0mm
Support Structure	Tree
Top Layers	4
Bottom Layers	4

Zusätzliche Parameter wie die Druckgeschwindigkeit, sind hierbei individuell für den zu gewählten 3D Drucker zu ermitteln. Allgemein wurden hier die Standardeinstellungen verwendet, welche in diesem Falle einen guten Kompromiss zwischen Qualität und Druckzeit lieferten.

## 0.4 Erstellung erster Prototyp



Bild 0-3: Prototyp Hardware: Erster Prototyp des autonomen Schachtisch

Für die Achsenführung der beiden X- und Y-Achsen wurden konventionelle 20x20mm Aluminium-Profile verwendet, welche mit einfachen Mitteln und wenig Geschick passend zugeschnitten werden können. Allgemein wurde eine X-Y Riemenführung verwendet, wobei jede Achse einen separaten Nema 17 Schrittmotor inklusive des passenden Endschalters montiert hatte. Bei den Schlitten, welche auf den Aluminium-Profilen laufen, wurden fertige Standartkomponenten verwendet, um das Spiel in der Mechanik zu minimieren. Diese stellen jedoch einen großen Posten in der Preiskalkulation dar. Die Vorteile überwogen jedoch, da diese nicht manuell erstellt und getestet werden müssen.

Bereits während des Designprozess konnte anhand einer statischen Simulation des Modells erkannt werden, dass trotz der Optimierung des Fahrweges beider Achsen durch die Verkleinerung der Halterungen der Aluminium-Profile dieser nicht ausreicht. Mit dieser Konstellation können die Figuren nicht ausreichend weit aus dem Spielfeld platziert werden und verbleiben in den äußeren Spielfeldern. Dieser Effekt war unerwünscht und schränkt das Spielerlebnis deutlich ein.

Um dies zu verhindern wurde der zentrale Schlitten der Y-Achse, auf welchem der Elektromagnet für die Figur-Mitnahme platziert ist, um einen weiteren Elektromagnet erweitert. Diese befinden sich nun nicht mehr mittig auf dem Schlitten, sondern wurden um 110mm in Richtung der X-Achse versetzt. So ist es möglich Figuren bis ganz an den Rand verschieben zu können.

Diese Lösung erfordert jedoch einen komplexeren Bahnplanungs-Algorithmus, da die Elektromagneten zwischen einzelnen Zügen gewechselt werden müssen. Dies führt zu einem zeitlich kürzeren Stillstand der Figur auf dem Schachfeld.

Alle selbst-konstruierten Teile wurden anschließend mittels 3D Druck erstellt und konnten in die Tischplattenbasis eingeschraubt werden. Die Verwendung der aus Holz bestehenden Grundplatte erschwerte jedoch eine akkurate Platzierung der Teile und die bereits existierenden Seitenwände schränkten diese noch zusätzlich ein. Somit erforderte der komplette Zusammenbau mehrere Tage und zusätzliche Iterationen des 3D-Designs, um den Einbau spezifischer Teile zu ermöglichen. Das Design stellt jedoch eine solide Grundlage dar, welche für die weitere Software und Hardware-Entwicklung essentiell ist.

### 0.4.2 Parametrisierung Schachfiguren

Da das System die auf dem Feld befindlichen Schachfiguren anhand von Near Field Communication (NFC) Tags erkennt, müssen diese zuerst mit Daten beschrieben werden. Die verwendeten NXP NTAG 21 Chips, besitzen einen vom Benutzer verwendbaren Speicher von 180 Byte. Dieser kann über ein NFC-Lese/Schreibgerät mit Daten verschiedenster Art beschrieben und wieder ausgelesen werden. Moderne Mobiltelefone besitzen in der Regel auch die Fähigkeit mit passenden NFC Tags kommunizieren zu können; somit sind keine Stand-Alone Lesegeräte mehr notwendig.

Der Schachtisch verwendet dabei das NFC Data Exchange Format (NDEF) Dateiformat welches Festlegt, wie die Daten auf dem NFC Tag gespeichert werden. Da diesen ein Standardisiertes Format ist, können alle gängigen Lesegeräte und Chipsätze diese Datensätze lesen. Der im autonomen Schachtisch verwendete Chipsatz PN532 von NXP ist dazu ebenfalls in der Lage.

Um das NDEF Format verwenden zu können, müssen die NFC Tags zuerst auf diese formatiert werden. Die meisten käuflichen Tags sind bereits derart formatiert. Alternativ kann dies mittels Mobiltelefons und passender Applikation geschehen. Da NDEF Informationen über die Formatierung und der gespeicherten Einträge speichert, stehen nach der Formatierung nur noch 137 Bytes des NXP NTAG 21 zur Verfügung.

Per Lesegerät können anschließend mehrere NDEF Records auf den Tag geschrieben werden. Diese sind mit Dateien auf einer Festplatte vergleichbar und können verschiedenen Dateiformate und Dateigrößen annehmen. Ein typischer Anwendungsfall ist der NDEF Record Type Definition (NDEF-RTD) URL Datensatz. Dieser kann dazu genutzt werden eine spezifizierte URL auf dem Endgerät aufzurufen, nachdem der NFC Tag gescannt wurde. [?]

## SETTINGS

### FIGURE TYPE

KING  QUEEN  ROOK  BISHOP  KNIGHT  PAWN

---

### FIGURE COLOR

BLACK  WHITE

---

## RESULT

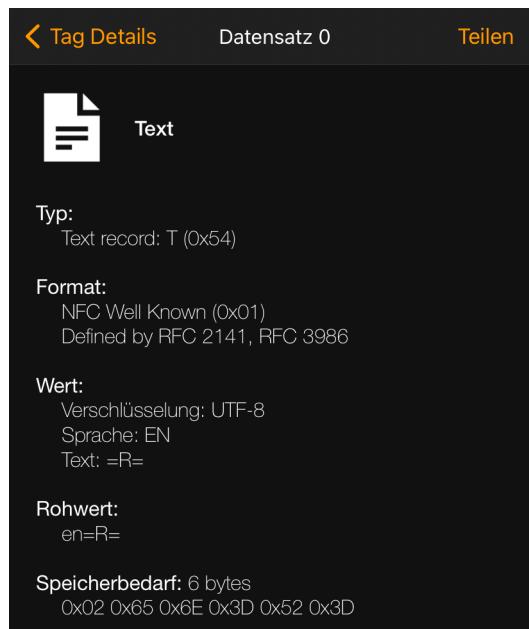
No	DATA	NDEF_RECORD_CONTENT
0	1,1,0,1,0,0,0,0 =[208]	=D=
1	1,1,0,1,0,0,0,1 =[209]	=N=
2	1,1,0,1,0,0,1,0 =[210]	=O=
3	1,1,0,1,0,0,1,1 =[211]	=Ó=
4	1,1,0,1,0,1,0,0 =[212]	=Ô=
5	1,1,0,1,0,1,0,1 =[213]	=Õ=
6	1,1,0,1,0,1,1,0 =[214]	=Ö=
7	1,1,0,1,0,1,1,1 =[215]	=x=

Bild 0-4: Prototyp Hardware: Tool zur Erstellung des NDEF Payloads: ChessFigureID-Generator.html

Der autonome Schachtisch verwendet den einfachsten NDEF-RTD Typ, den sogenannten Text-Record, welcher zum Speichern von Zeichenketten genutzt werden kann, ohne das eine Aktion auf dem Endgerät ausgeführt wird. Jeder Tag einer Schachfigur, welche für den autonomen Schachtisch verwendet werden kann, besitzt diesen NDEF Record an der ersten Speicher-Position. Alle weiteren eventuell vorhandenen Records werden vom Tisch ignoriert. [?]

Um die Payload für den NFC Record zu erstellen wurde ein kleine Web-Applikation erstellt, welche den Inhalt der Text-Records erstellt. Dieser ist für jede Figur individuell und enthält den Figur-Typ und die Figur-Farbe. Das Tool unterstützt auch das Speichern weiterer Attribute wie einem Figur-Index, welcher aber in der finalen Software-Version nicht genutzt wird. 0-4

Nach dem Beschreiben eines NFC Tags ist es zusätzlich möglich, diesen gegen Auslesen mittels einer Read/Write-Protection zu schützen. Diese Funktionalität wird jedoch nicht verwendet, um das Kopieren einzelner Figuren durch den Benutzer zu ermöglichen.



**Bild 0-5:** Prototyp Hardware: NDEF Text Record Payload für einen weißen Turm

chen. Somit kann dieser leicht seine eigenen Figuren erschaffen, ohne auf das Tool angewiesen zu sein. Auch ist es so möglich, verschiedene Figur-Sets zu mischen; somit kann ein Spieler verschiedene Sets an Figuren mit dem autonomen Schachtisch verwenden.

### 0.4.3 Schaltungsentwurf

- auswahl der Motortreiber (leise, bus ansteuerung)
- ansteuerung pn532 und umsetzung auf uart
- platinendesign
- ansterung elektromagnetet

### Implementierung HAL

- ansteuerung des TMC5160
- ansterung des Microncontrollers (PN532, LED)
- integration in controller software

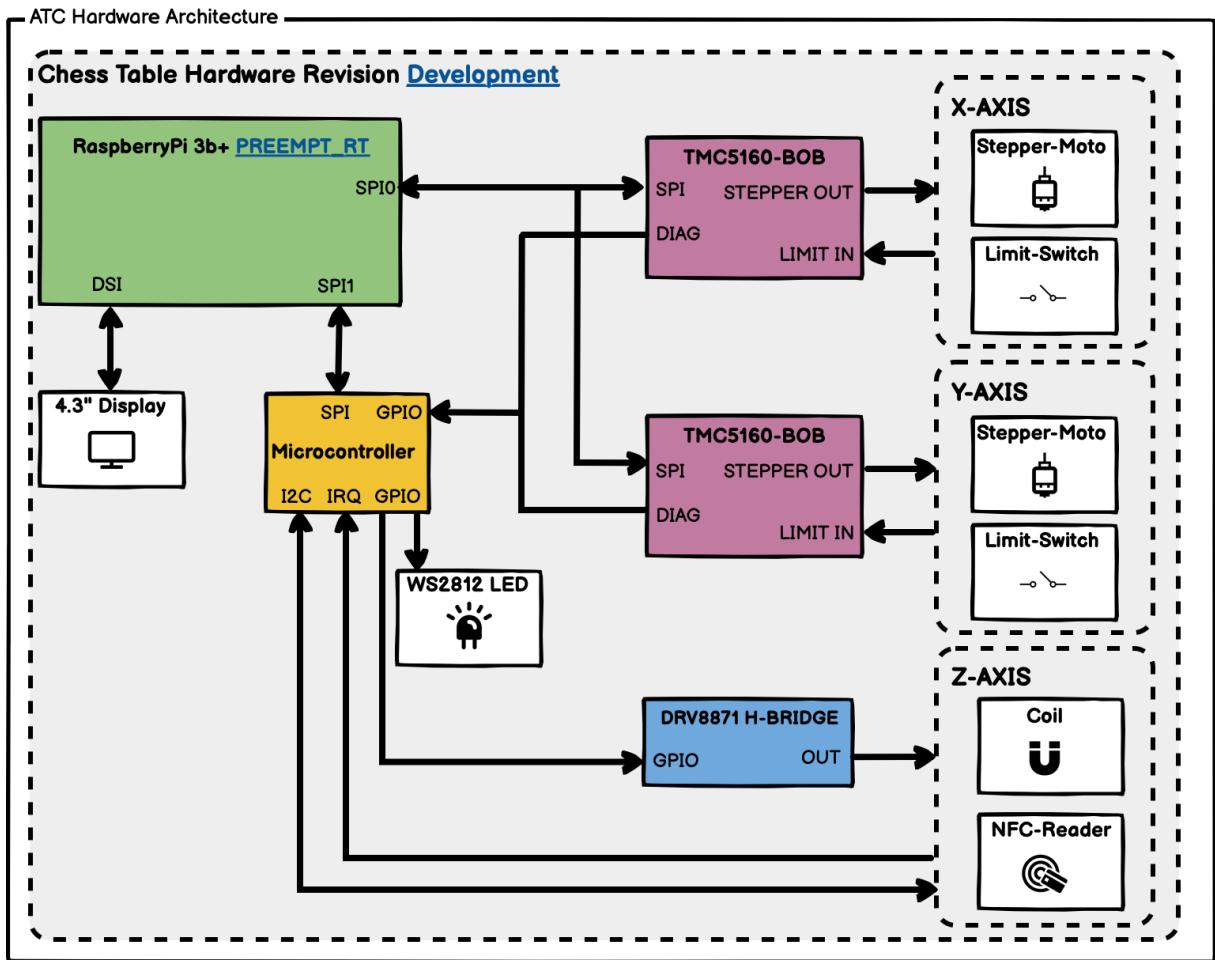


Bild 0-6: Prototyp Hardware: Blockdiagramm



**Bild 0-7:** Production Hardware: Finaler autonomer Schachtisch

- welche funktion stehen bereit tabelle
- step dir interface => erfodert jedoch eine rt fähige ## Fazit bezüglich des ersten Prototypens
- nicht für production geeignet
- aufbau und calibrierung langwierig
- trotzdem robustes design auf kleinem formfaktor
- verwendeten elektromagnete nicht stark genug, somit über aqußerhalb der specs betrieben was zu temeraturproblemen führte
- gewicht der Figuren zu klein bzw magnete zu start
- workarounds in der software nötig durch die beiden magnete
- nicht die beste entscheidung direkt auf grössze zu optimieren

## 0.5 Erstellung des zweiter Prototypens

### 0.5.1 Modifikation der Mechanik

- Dauertest hat gezeigt dass Mechanik zu viel spiel aufweisst
- Motorenhalterung der y achse schränkt den bewegungsspielraum um mehr als 10cm ein, welches zu einem unwesentlichen grösseren verhältnis von Spielfeldgrösse und Abmessungen des Schachtischs
- CoreXY bietet Vorteil:
- Motoren fest am rahmen => weniger kabel + gewicht an der Y Achse
- jedoch komplexerer Aufwand der riemenverlegung so komplexere 3d bauteile
- Tischabmessungen 620x620mm dabei Bewegungsspielraum vom 580x580 zuvor nur 480x480
- langer zusammenbau !!

### 0.5.2 Optimierungen der Spielfiguren

Die bisherigen genutzten vorgefertigten Figuren funktionierten mit dem ersten Prototyp ohne erkennbare Fehler. Sie wiesen aber trotzdem eine zu hohe Fehleranfälligkeit, in Bezug auf das gegenseitige Beeinflussen (abstoßen, anziehen) durch die verwendeten Magnete auf.

Die Größe der Figuren kann durch die fest definierte Schachfeldgröße von 55mm und der verwendeten NFC Tags nicht verändert werden. Nach einigen Testdurchläufen mit dem ersten Prototyp war zu erkennen, dass sich die Figuren je nach aktueller Situation auf dem Spielfeld weiterhin magnetisch anziehen. Um diesen Fehler zu beheben wurden verschiedenen Bewegungsgeschwindigkeiten getestet, ergaben allerdings für diesen Anwendungsfall keine merkliche Verbesserung.

Dies führt je nach Spielverlauf zu Komplikationen, sodass die Figuren manuell vom Benutzer wieder mittig auf den Felder platziert werden müssen.

Um dies zu verhindern, wurde einige Figuren zusätzlich mit einer 20mm Unterlegscheibe am Boden beschwert. Diese behob das Problem, jedoch erwies sich das NFC Tag nicht mehr als lesbar. Dies resultierten aus dem Prozessgedanken, die Schachfiguren

ebenfalls selbst mit dem 3D-Drucker herzustellen und die Magnete direkt in den Boden der Figur einlassen zu können.

Die aktuell verwendeten Figuren des ersten Prototyp wiegen zwischen 8 Gramm für die Bauern und 10 Gramm für die restlichen Figuren. Der Test mit der Unterlegscheibe ergab das diese mit 5 Gramm zusätzlich genug Gewicht hinzufügen, um die magnetische Beeinflussung zu unterbinden.

Testweise wurden einige Figuren mittels 3D Drucker erstellt, um so das Gewicht zu erhöhen. Nach einem erfolgreichen Test wurde das CAD Modell so angepasst, dass sich der Magnet direkt in den Boden der Figur einkleben lässt. Des Weiteren wurden bei den Bauern die Magnete ausgetauscht. Die zuerst verwendeten 10x3mm Neodym-Magnete wurden bei diesen Figuren gegen 6x3mm Magnete getauscht. Somit sind im Design zwei verschiedenen Arten von Magneten notwendig, jedoch traten in den anschließend durchgeführten Testläufen keine Beeinflussungen mehr auf.

### 0.5.3 Änderungen der Elektronik

Mit ein relevanter Kritikpunkt, welcher bereits während des Aufbaus des ersten Prototyps zu erkennen war, ist die Umsetzung der Elektronik. Diese wurde im ersten Prototyp manuell aufgebaut und enthielt viele verschiedene Komponenten.

Die verwendeten Motortreiber stellten sich während der Entwicklung als sehr flexibel heraus, stellten aber auch einen signifikanten Kostenfaktor dar. Nach dem Aufbau und Erprobung des ersten Prototyps wurde ersichtlich, dass hier nicht alle zuerst angedachten Features der Treiber benötigt werden und so auch Alternativen in Betracht gezogen werden konnten. Zusätzlich konnte die Elektronik nur beschränkt mit anderen Systemen verbunden werden, was insbesondere durch die verwendete Serial Peripheral Interface (SPI) Schnittstelle geschuldet war.

All diese Faktoren erschweren einen einfachen Zusammenbau des autonomen Schachtischs. Die Lösung stellt die Verwendung von Standardhardware dar. Nach der Minimierung der elektrischen Komponenten und des mechanischen Aufbaus ist zu erkennen, dass der autonome Schachttisch einer CNC-Fräse bzw. eines 3D Drucker stark ähnelt. Insbesondere die XY-Achsen Mechanik sowie die Ansteuerung von Schrittmotoren wird in diesen Systemen verwendet. Mit dem Durchbruch von 3D Druckern im Konsumer-Bereich sind auch kleine und preisgünstige Steuerungen erhältlich, welche 2-3 Schrittmotoren und diverse zusätzliche Hardware ansteuern können.

Tabelle 0.5: Standardhardware 3D Drucker Steuerungen

	SKR 1.4 Turbo	Ramps 1.4	Anet A8 Mainboard
Stepper Driver	TMC2209	A4988 / TMC2209	A4988
LED Strip Port	WS2811 / RGB	-	-
Firmware	Marlin-FW 2.0	Marlin-FW 1.0	Proprietary

Hierbei existiert eine große Auswahl dieser mit den verschiedensten Ausstattungen. Bei der Auswahl dieser wurde vor allem auf die Möglichkeit geachtet sogenannte Silent-Schrittmotortreiber verwenden zu können, um die Geräuschimmissionen durch die Motoren so weit wie möglich zu minimieren. Im ersten Prototyp wurde unter anderem aus diesem Grund die TMC5160-BOB Treiber ausgewählt. Hierzu wurde der Schrittmotor-Treiber TMC2209 gewählt, welcher diese Features ebenfalls unterstützt und in der Variante als Silent-Step-Stick direkt in die meisten 3D Drucker Steuerungen eingesetzt werden können. Hierbei ist es wichtig, dass auf der gewählten Steuerung die Treiber-ICs nicht fest verlotet sind, sondern getauscht werden können. Ein weiterer Punkt ist die Kommunikation der Steuerung mit dem Host-System. Hierbei setzten alle untersuchten Steuerungen auf die USB Schnittstelle und somit ist eine einfache Kommunikation gewährleistet. Das verwendete eingebettete System im autonomen Schachtisch bietet vier freie USB Anschlüsse, somit ist eine einfache Integration gewährleistet.

Nach einer gründlichen Evaluation der zur Verfügung stehenden Steuerungen, wurde die SKR 1.4 Turbo Steuerung ausgewählt, da diese trotz des geringfügig höheren Markt-preises genug Ressourcen auch für spätere Erweiterung bietet und eine Unterstützung für die neuste Version der Marlin-FW[?] bereitstellt. Somit wurde die Elektronik durch die verwendete Plug&Play stark vereinfacht **0-8**.

### HAL: Implementierung GCODE-Sender

Durch die durchgeführten Änderungen an der Elektronik insbesondere durch die Verwendung einer Marlin-FW[?] fähigen Motorsteuerung, ist eine Anpassung der Hardware Abstraction Layer (HAL) notwendig. Diese unterstützt die Ansteuerung der Motoren und anderen Komponenten (z.B. Spindeln, Heizelemente) mittels G-Code und wird typischerweise in 3D Druckern und CNC-Fräsen eingesetzt. G-Code ist eine Marlin-FW[?] biete dabei einen großen Befehlssatz an G-Code Kommandos an. Bei diesem Projekt werden jedoch nur einige G-Code Kommandos verwendet, welche sich insbesondere auf die Ansteuerung der Motoren beschränken.

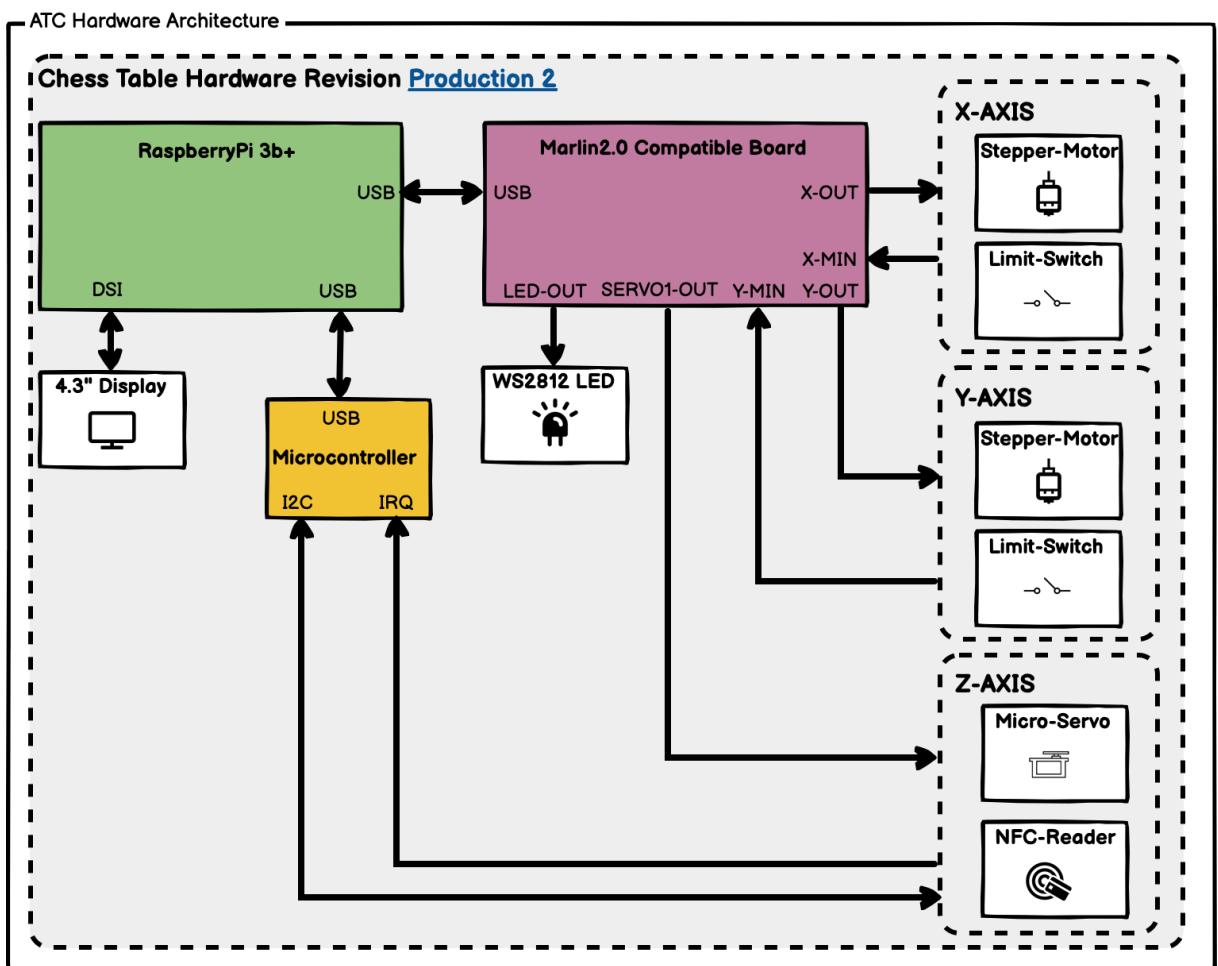


Bild 0-8: Production Hardware: Blockdiagramm

Tabelle 0.6: Grundlegende verwendete G-Code Kommandos

	G-Code Command	Parameters
Move X Y	G0	X Y
Move Home Position	G28	-
Set Units to Millimeters	G21	-
Set Servo Position	M280	P S
Disable Motors	M84	X Y

Die erforderlichen Kommandos wurden auf ein Minimum beschränkt, um eine maximale Kompatibilität bei verschiedenen G-Code-fähigen Steuerungen zu gewährleisten. Die Software unterstützt jedoch weitere Kommandos wie z.B. [M150](#) mit welche speziellen Ausgänge für LEDs gesteuert werden können. Dieses Feature bietet sowohl die verwendete Marlin-FW[?] als auch die verwendete Steuerung an. Sollte die verwendete Steuerung solch ein optionales Kommando nicht unterstützen, so werden diese ignoriert was zur Folge hat, dass auch preisgünstige Steuerungen verwendet werden können.

Die Kommunikation zwischen Steuerung und eingebetteten System geschieht durch eine USB Verbinden. Die Steuerung meldet sich als virtuelle Serielle Schnittstelle im System an und kann über diese mit der Software kommunizieren. Auch werden so keine speziellen Treiber benötigt, da auf nahezu jedem System ein Treiber (USB-CDC) für die gängigsten USB zu seriell Wandler bereits installiert ist. Die Software erkennt anhand der zur Verfügung stehenden USB-Geräte sowie deren Vendor und Product-ID Informationen die verbundene Steuerung und verwendet diese nach dem Start automatisch. Hierzu wurde zuvor eine Liste mit verschiedenen getesteten Steuerungen sowie deren USB-Vendor und Product-ID angelegt.

Tabelle 0.7: Hinterlegte G-Code Steuerungen

Product	Vendor-ID	Product-ID	Board-Type
Bigtreetech SKR 1.4 Turbo	1d50	6029	Stepper-Controller
Bigtreetech SKR 1.4	1d50	6029	Stepper-Controller
Bigtreetech SKR 1.3	1d50	6029	Stepper-Controller

Damit die Software mit der Steuerung kommunizieren kann, wurde eine G-Code Sender Klasse implementiert, welche die gleichen Funktionen wie die HAL-Basisklasse bereitstellen. Nach Aufruf einer Funktion zum Ansteuern der Motoren, wird aus den übergebenen Parametern das passende G-Code Kommando in Form einer Zeichenkette

zusammengesetzt und auf die Serielle Schnittstelle geschrieben.

```
1 //GCodeSender.cpp
2 bool GCodeSender::setServo(const int _index,const int _pos) {
3     return write_gcode("M280 P" + std::to_string(_index) + " S
4                         " + std::to_string(_pos));           //MOVE SERVO
5 }
6 bool GCodeSender::write_gcode(std::string _gcode_line, bool
7                               _ack_check) {
8     //...
9     //...
10    //FLUSH INPUT BUFFER
11    port->flushReceiver();
12    //APPEND NEW LINE CHARAKTER IF NEEDED
13    if (_gcode_line.rfind('\n') == std::string::npos)
14    {
15        _gcode_line += '\n';
16    }
17    //WRITE COMMAND TO SERIAL LINE
18    port->writeString(_gcode_line.c_str());
19    //WAIT FOR ACK
20    return wait_for_ack();
21 }
22 bool GCodeSender::wait_for_ack() {
23     int wait_counter = 0;
24     //...
25     //...
26     while (true) {
27         //READ SERIAL REONSE
28         const std::string resp = read_string_from_serial();
29         //...
30         //...
31         //PROCESS
32         if (resp.rfind("ok") != std::string::npos)
33         {
34             break;
35         }else if(resp.rfind("echo:Unknown") != std::string::
36                   npos) {
37             break;
38         }else if(resp.rfind("Error:") != std::string::npos) {
39             break;
40         }else if (resp.rfind("echo:busy: processing") != std::
41                   string::npos) {
42             wait_counter = 0;
43             LOG_F("wait_for_ack: busy_processing");
44         }else {
45             //READ ERROR COUNTER AND HANDLING
46             wait_counter++;
47             if (wait_counter > 3)
48             {
```

```
47         break;
48     }
49 }
50 }
51 //...
52 //...
53 return true;
54 }
```

Die Steuerung verarbeitet diese und bestätigt die Ausführung mit einer Acknowledgement-Antwort. Hierbei gibt es verschiedenen Typen. Der einfachste Fall ist ein `ok`, welches eine erfolgreiche Abarbeitung des Kommandos signalisiert. Ein weiterer Fall ist die Busy-Antwort `echo:busy`. Diese Signalisiert, dass das Kommando noch in der Bearbeitung ist und wird im Falle des autonomen Schachttisches bei langen und langsam Bewegungen der Mechanik ausgegeben. Das System wartet diese Antworten ab bis eine finale `ok`-Antwort zurückgegeben wird, erst dann wird das nächste Kommando aus der Warteschlange bearbeitet.

### HAL: I2C Seriell Umsetzer

Durch den Wegfall der zuvor eingesetzten Elektronik und der Austausch durch die SKR 1.4 Turbo Steuerung, ist jedoch ein Anschluss des PN532 NFC Moduls nicht mehr direkt möglich, da dieses mittels Inter-Integrated Circuit (I2C) Interface direkt mit dem eingebetteten System verbunden war. Dieses Interface entfällt nun. Dennoch besteht weiterhin die Möglichkeit, jedoch wurde auch hier auf eine USB Schnittstelle gewechselt. So ist es möglich das System auch an einem anderen Host-System zu betreiben, wie z.B. an einem handelsüblichen Computer.

Dazu wurde ein Schnittstellenwandler entwickelt welcher die I2C Schnittstelle zu einer USB Seriell wandelt. Indes wurde ein Atmega328p Mikrokontroller eingesetzt, da dieser weit verbreitet und preisgünstig zu beschaffen ist. Die Firmware des Mikrokontroller stellt ein einfaches kommandobasiertes Interface bereit. Die Kommunikation ist mit der Kommunikation und der Implementierung des G-Code Senders vergleichbar und teilen sich die gleichen Funktionen zur Kommunikation mit der Seriellen Schnittstelle.

```
1 //userboardcontroller.cpp Atmega328p Firmware
2 //simplyfied version
3 char scan_nfc_tag(){
4     //...
5     if (nfc.tagPresent())
6     {
```

```
7     //READ TAG CONTENT
8     NfcTag tag = nfc.read();
9     //READ NDEF PAYLOAD
10    NdefMessage msg = tag.getNdefMessage();
11    if(msg.getRecordCount() > 0){
12        //READ FIRST RECORD
13        NdefRecord record = msg.getRecord(0);
14        const int payloadLength = record.getPayloadLength
15            ();
16        byte payload[payloadLength];
17        //...
18        record.getPayload(payload);
19        //...
20        //...
21        //RETURN FIGURE ID
22        if(payloadLength == 6){
23            return payload[3];
24        }
25    }
26 }
```

In diesem Falle wird nur ein Befehl zum Auslesen des NFC Tags benötigt. Das Host-System sendet die Zeichenkette `_readnfc_` zum Mikrokontroller und dieser versucht über das PN532 Modul ein NFC Tag zu lesen. Wenn dieses erkannt wird und einen passenden Payload enthält, antwortet dieser mit dem String `_readnfc_res_PICTURE-ID_ok_` oder wenn kein Tag gefunden wurde mit `_readnfc_res_empty_`. Auch hier wird wie bei der G-Code Sender Implementierung auf Fehler bei der Kommunikation bzw. einem Abbruch durch einen Timeout reagiert. Das System initialisiert die Serielle Schnittstelle neu und resettet das System durch setzen des DTR GPIO am USB-Seriell Wandler ICs (falls vorhanden).

```
1 //UserBoardController.cpp HOST - SYSTEM
2 //simplyfied version
3 ChessPiece::FIGURE UserBoardController::read_chess_piece_nfc()
4 {
5     ChessPiece::FIGURE fig;
6     fig.type = ChessPiece::TYPE::TYPE_INVALID;
7     //...
8     //READ SERIAL RESULT
9     const std::string readres = send_command_blocking(
10         UBC_COMMAND_READNFC);
11     //...
12     //SPLIT STRING -
13     const std::vector<std::string> re = split(readres,
14         UBC_CMD_SEPERATOR);
15     //READ SECTIONS
16     //...
```

```
15 //...
16 const std::string figure = re.at(3);
17 const std::string errorcode = re.at(4);
18 //CHECK READ RESULT
19 if(errorcode == "ok"){
20     if(figure.empty()){
21         break;
22     }
23     //...
24     //...
25     //DETERM FINAL READ FIGURE
26     const char figure_charakter = figure.at(0);
27     fig = ChessPiece::getFigureByCharakter(
28         figure_charakter);
29 }
30 //...
31 return fig;
32 }
```

Das System erkennt den Anschluss der Hardware beim Start auf die gleiche Art und Weise wie der G-Code Sender. Dafür wurden einige verschiedene Mikrokontroller im System hinterlegt, auf welchen die Firmware getestet wurde.

Tabelle 0.8: Hinterlegte Mikrokontroller

Product	Vendor-ID	Product-ID	Board-Type
Arduino Due [Programming Port]	2341	003d	User-Move-Detector
Arduino Due [Native SAMX3 Port]	2341	003e	User-Move-Detector
CH340	1a86	7523	User-Move-Detector
HL-340	1a86	7523	User-Move-Detector
STM32F411	0483	5740	User-Move-Detector

### 0.5.4 Fazit bezüglich des finalen Prototypens

- modularer hardware aufbau
- einfach/gut verfügbare materialien verwendet
- geänderte Mechanik resultiert in nahezu Spielfreier Mechanik (+- 1mm), welches für diesen Zweck mehr als ausreicht
- 6h dauerstest bestanden

Tabelle 0.9: Eigenschaften die finalen Prototypen

	ATC – autonomous Chessboard
Feldabmessungen (LxBxH)	57x57mm
Abmessungen (LxBxH)	620x620x170mm
Gewicht	5.7kg
Konnektivität	WLAN, USB
Automatisches Bewegen der Figuren	ja
Erkennung Schachfigurstellung	ja
Spiel Livestream	ja
Cloudanbindung (online Spiele)	ja
Parkposition für ausgeschiedene Figuren	ja
Stand-Alone Funktionalität	ja
Besonderheiten	User-Port für Erweiterungen

- alle anforderungen erfüllt
- zulasten der geschwindigkeit insbesondere bei der erkennung des User-Move
- erweiterungsmöglichkeit in hard uns SOFTWARE

## 0.6 Entwicklung der Cloud Infrastruktur

Die erste Phase der Entwicklung des Systems bestand in der Auslegung und Erstellung der Cloud-Infrastruktur und der darauf ausgeführten Services. Die “Cloud” stellt in diesem Zusammenhang einen Server dar, welcher aus dem Internet über eine feste IPv4 und IPv6-Adresse verfügt und frei konfiguriert werden kann. Auf diesem System ist der Schach-Cloud Stack 0-9 installiert, welcher zum einen aus der Schach-Software besteht, welche in einem Docker-Stack ausgeführt wird und zum anderen. . . .

### 0.6.1 API Design

Das System soll so ausgelegt werden, dass es zu einem späteren Zeitpunkt mit verschiedenen Client-Devices mit diesem kommunizieren können. Dazu zählen zum einen der autonome Schachtisch, aber z.B. auch einen Web-Client, welcher die Funktionalität eines Schachtisches im Browser abbilden kann. Hierzu muss das System eine

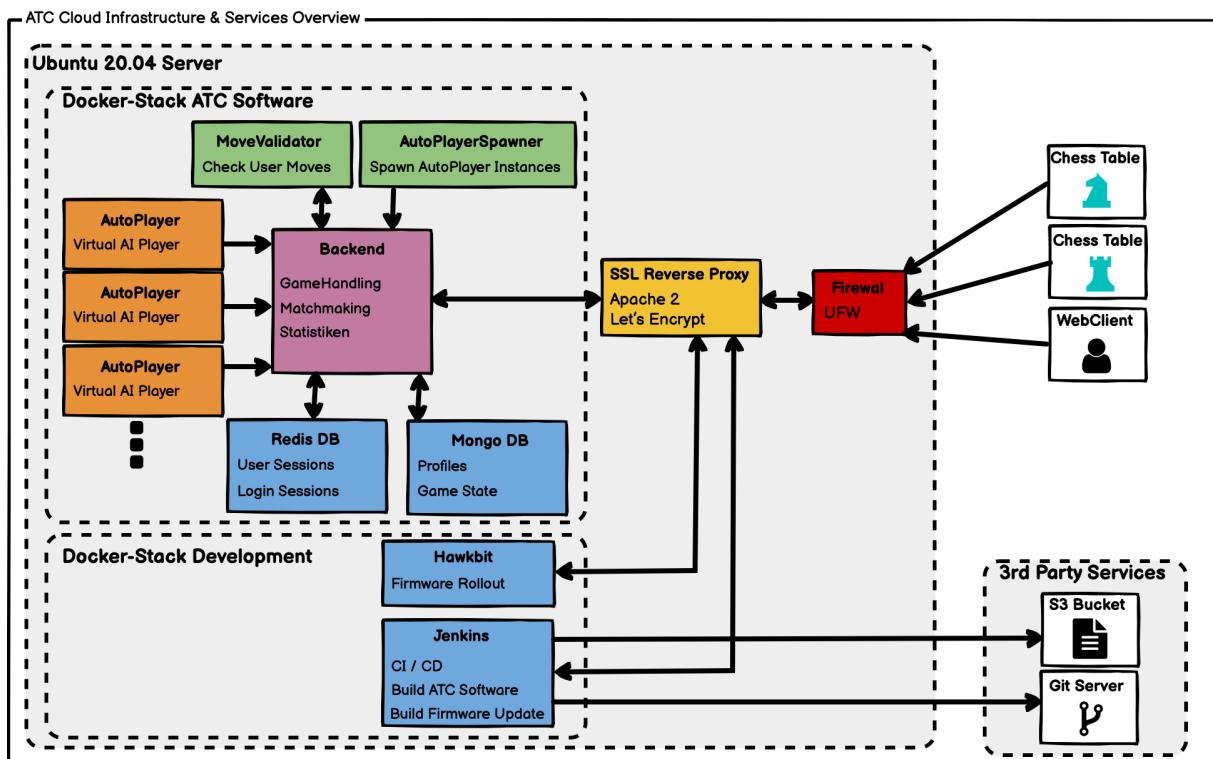


Bild 0-9: Gesamtübersicht der verwendeten Cloud-Infrastruktur

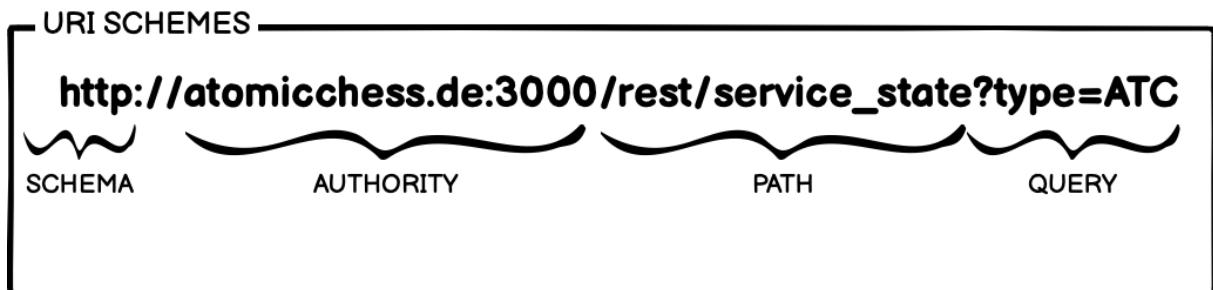


Bild 0-10: Cloud-Infrastruktur: Aufbau einer URI

einheitliche Representational State Transfer (REST)-Schnittstelle bereitstellen.

Die RESTful API stellt verschiedene Ressourcen bereit, welche durch eine URI 0-10 eindeutig identifizierbar sind. Auf diese können mittels verschiedenster HTTP Anfragemethoden (GET, POST, PUT, DELETE) zugegriffen werden. Jeder dieser Methoden stellt einen anderen Zugriff auf die Ressource dar und beeinflusst somit das Verhalten und die Rück-Antwort dieser.

Eine URI besteht dabei aus mehreren Teilen. Das Schema gibt an wie die nachfolgenden Teile interpretiert werden sollen. Dabei wird bei einer RESTful Schnittstelle typischerweise das Hypertext Transfer Protocol (HTTP) Protokoll, sowie Hypertext Transfer Protocol Secure (HTTPS) verwendet. Dabei steht HTTPS für eine verschlüsselte

Verbindung.

Somit stellt die RESTful API eine Interoperabilität zwischen verschiedenen Anwendungen und Systemen bereit, welche durch ein Netzwerk miteinander verbunden sind. Dieser Ansatz ist somit geeignet um die verschiedenen Client Systeme (Schachtisch, Webclient) eine Kommunikation mit dem Server zu erlauben.

- reverse Proxy für https

### 0.6.2 Service Architektur

- was ist ein Service
- microservice ansatz
- Kapselung der Schachspiel spezifischen funktionalitäten
- verwendung von NoSQL Datenbanken somit müssen Tabellen nicht speziell auf Schach spezifische Felder ausgelegt sein
- statelss Diese stellen alle wichtigen Funktionen zum Betrieb des autonomen Schachtischs zur Verfügung.

### 0.6.3 Vorüberlegungen

- welche funktionalitäten müssen abgedeckt werden
- client aktivitendiagramm

### 0.6.4 Backend

- matchmaking schachlogik
- zentraler zugriffspunkt auf das System und stellt diese abi bereit
- stellt spielerprofile aus datenbanken bereit bereit
- authentifizierung der clients und deren sessions
- weiterleitung der von spielerinteraktionen an move validator
- spielfelder werden als string übermittelt = hier fen representation; einfach zu parsen; standart

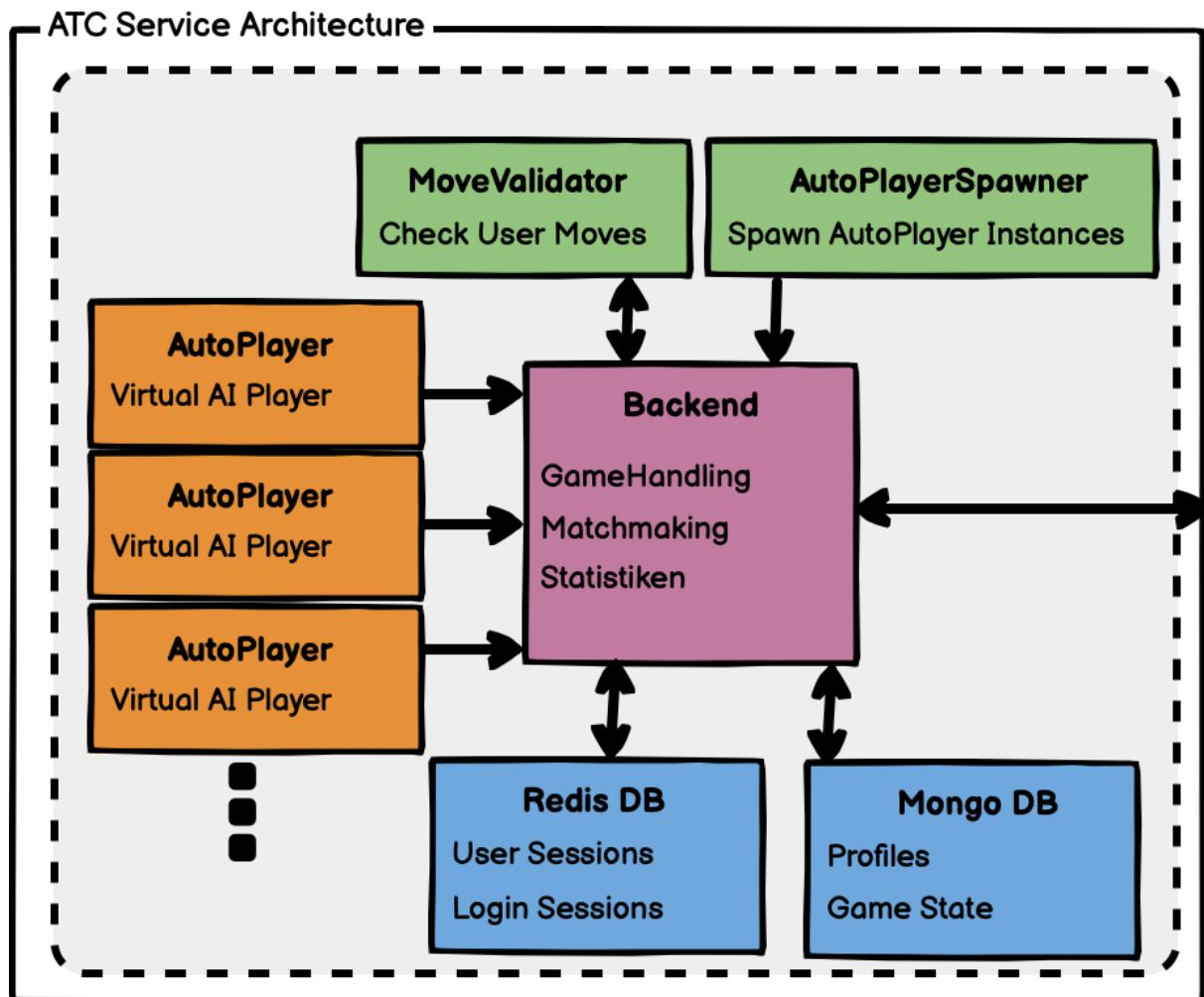


Bild 0-11: Cloud-Infrastruktur: Aufbau der Service Architecture

A screenshot of the Postman application interface. The top bar shows 'GET' and the URL 'http://atomicchess.de:3000/rest/login?hwid=f1ow389djiw&playertype=1'. The 'Params' tab is selected, showing two checked parameters: 'hwid' with value 'f1ow389djiw' and 'playertype' with value '1'. The 'Body' tab is selected, showing a JSON response:

```

1  {
2      "err": null,
3      "status": "ok",
4      "sid": "daada700-a38a-11eb-9c30-7907ebbd50ac",
5      "profile": {
6          "profile_config": {
7              "SETTINGS": null,
8              "USER_DATA": null
9          },
10         "logs": [],
11         "hwid": "f1ow389djiw"
}

```

Bild 0-12: Cloud-Infrastruktur: Backend Login-Request und Response

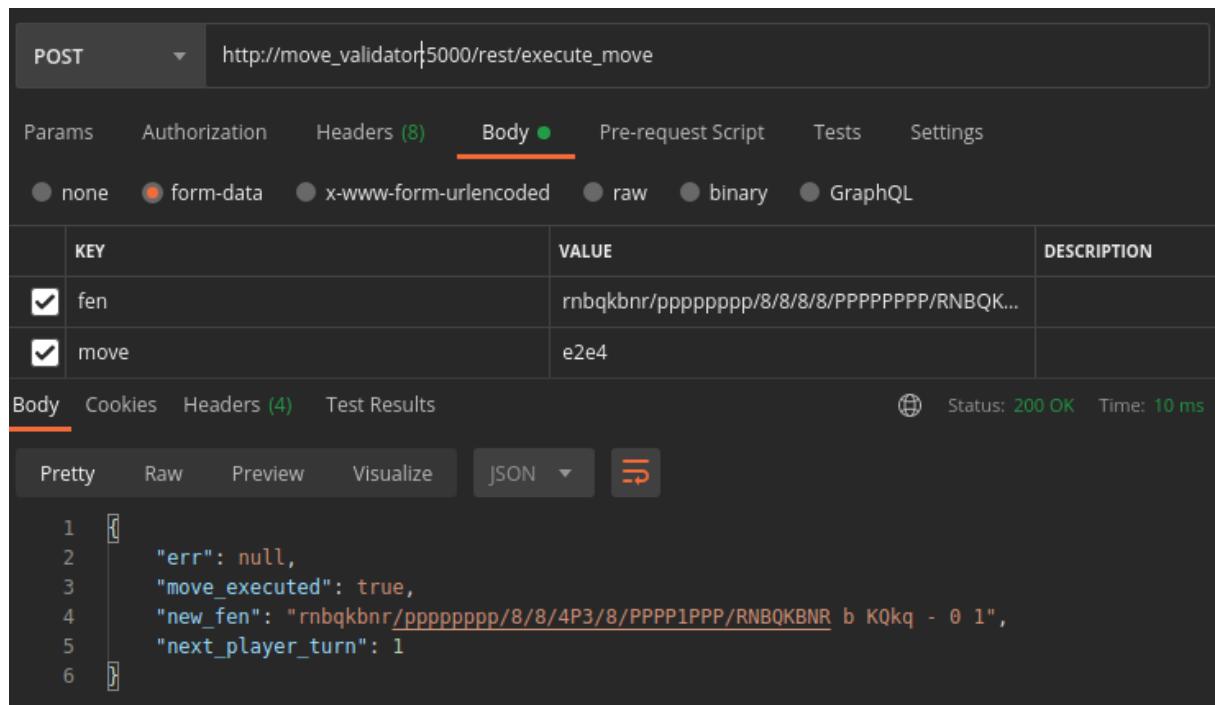


Bild 0-13: MoveValidator: Beispiel Request zur Ausführung eines Zuges auf einem gegebenen Schachbrett

## 0.6.5 MoveValidator

Der MoveValidator-Service bildet im System die eigentliche Schachlogik ab. Die Aufgabe ist es, die vom Benutzer eingegebenen Züge auf Richtigkeit zu überprüfen und auf daraufhin neuen Spiel-Status zurückzugeben. Dazu zählen unter anderem das neue Schachbrett und ob ein Spieler gewonnen oder verloren hat.

Bevor ein Spiel begonnen wird, generiert der MoveValidator das initiale Spielfeld und bestimmt den Spieler, welcher als erstes am Zug ist.

Der Backend-Service fragt ein neues Spiel an oder übergibt einen Schachzug inkl. des aktuellen Spielbrett-Aufbaus an den Service.**0-13** Der Response wird dann vom Backend in der Datenbank gespeichert und weiter an die Client-Devices verteilt.

Tabelle 0.10: MoveValidator-Service API Overview

MoveValidator API	API-Route	Method	Form-Data
Check Move	/rest/check_move	POST	* fen * move * player
Execute Move	/rest/execute_move	POST	fen * move
Validate Board	/rest/validate_board	POST	fen
Init Board	/rest/init_board	GET	

Allgemein geschieht die Kommunikation über vier API Calls, welche vom MoveValidator-Service angeboten werden. Als erstes wird vom Backend der [/rest/init\\_board](#) Request gestartet, um die initialen Spielbrettsituationen zu erhalten. Danach folgt die FEN-Notation (FEN) Notation, welche die aktuelle Schachbrett-Situation beschreibt. Anschließend wird der [/rest/check\\_move](#) Request gestartet, um den nächsten Zug zu überprüfen. Schließlich wird der [/rest/execute\\_move](#) Request gestartet, um den Zug auszuführen.

Tabelle 0.11: Vergleich FEN - X-FEN

FEN-TYPE	FEN-String
FEN	rnbqkbnr/pp1pppp/8/2p5/4P3/5N2/PPPP1PPP/RNBQKB1R
X-FEN	rnbqkbnr/pp1pppp/8/2p5/4P3/5N2/PPPP1PPP/RNBQKB1R b KQkq - 1 2
SCHEMA	Board Player-Color Rochade En-Passant Halfturn Turn-Number

Alle gängigen Schachprogramme und Bibliotheken unterstützen das Laden von Spielbrettern in der FEN bzw X-FEN Schreibweise, ebenso die für den MoveValidator Service verwendete Python-Chess Bibliothek [?]. Diese unterstützt zusätzlich die Generierung der für den Benutzer möglichen Schachzügen, welche auf dem aktuellen Brett möglich sind.

Diese Liste wird vom System dazu verwendet, um sicherzustellen, dass der Benutzer nur gültige Züge tätigen kann. Diese Funktion lässt sich zusätzliche abschalten, falls das Spiel nicht nach den allgemeinen Schachregeln verlaufen soll. Bei der Generierung der möglichen Schachzüge muss zwischen den Legal-Moves und den Pseudo-Legal Schachzügen unterschieden werden. Die Legal-Moves beinhalten nur die nach den Schachregeln möglichen Zügen, welche von Figuren des Spielers ausgeführt werden können. Die Pseudo-Legal Schachzüge sind alle Schachzüge, welche von den Figuren auf dem aktuellen Schachbrett möglich sind; darin sind unter anderem auch alle anderen Figur-Züge enthalten, solange der König des aktuellen Spielers sich aktuell auf dem Schachbrett befindet.

Wenn ein Spieler an der Reihe ist und einen Zug getätigter hat, wird sein getätigter Zug mittels der `/rest/check_move` API überprüft und festgestellt, ob dieser gemäß der Legal-Moves durchführbar war. Ist dies der Fall, wird der Zug auf dem onlie-Spielbrett angewendet. Dies geschieht durch die `/rest/execute_move` API. Diese führt den Zug aus, ermittelt anschließend das neue Spielbrett und überprüft zusätzlich, ob das Spiel gewonnen oder verloren wurde.

Hat der Benutzer jedoch einen ungültigen Zug ausgeführt, wird dieser vom System gestrichen und der Client des Benutzers stellt den Zustand des Spielbretts vor dem getätigten Zug wieder her. Danach hat der Benutzer die Möglichkeit einen alternativen Zug auszuführen.



Bild 0-14: WebClient: Spielansicht

### 0.6.6 Entwicklung WebClient

Der WebClient wurde primär dazu entwickelt, um das System während der Entwicklung zu testen. Dieser simuliert einen autonomen Schachtisch und verwendet dabei die gleichen HTTP Requests. Um das zu ermöglichen wurde dieser vollständig in JavaScript (JS) umgesetzt im Zusammenspiel mit Hypertext Markup Language (HTML) und Cascading Style Sheets (CSS) und ist somit komplett im Browser ausführbar.

Ausgeliefert werden die statischen Dateien zur Einfachheit durch den Backend-Service; es wurde kein gesonderter Frontend-Service angelegt. Durch die Implementierung des WebClienten in JS ist dieser sogar lokal über einen Browser ausführbar, ohne dass die benötigten Dateien über einen Webserver ausgeliefert werden müssen.

Zusätzlich zu dem verwendeten Vanilla-JS wurde jQuery als zusätzliche JS Bibliothek verwendet, welches eine Manipulation der HTML Elemente stark vereinfacht. Diese bietet insbesondere einfach zu nutzende HTTP-Request Funktionen bzw. Asynchronous JavaScript and XML (AJAX) an, welche für die Kommunikation mit dem Backen-Service verwendet werden. Diese werden im Hintergrund eingesetzt, sodass der WebClient auto-

matisch den neuen Spielzustand dem Benutzer anzeigt. Dies geschieht mittels `polling`, bei dem der Webbrower in zyklischen Abständen die aktuellen Spiel-Informationen vom Backen-Service abfragt. Diese Methode wurde verwendet, um eine maximale Kompatibilität mit verschiedenen gegebenenfalls älteren Web-Browsern sicherzustellen. Eine moderne alternative ist die Verwendung von Web-Sockets, bei welcher der Web-Browser eine direkte TCP-Verbindung zum Webserver (in diesem Fall der Backend-Service) aufnehmen und so eine direkte Kommunikation stattfinden kann ohne Verwendung der `polling`-Methode.

Der Hauptanwendungsfall des Webclienten während der Entwicklung ist es, weitere Spieler zu simulieren und so ein Spiel mit nur einem autonomen Schachtisch testen zu können. Durch den Webclient ist zusätzliche möglich, gezielt Spiele und Spielzüge zu simulieren. Hierzu gehören vor allem Sonderzüge wie die Rochade oder der En-Passant Zug. Auch können durch den Webclient ungültige Züge simuliert werden, welche durch die Verwendete Schach-AI nicht getätigter werden.

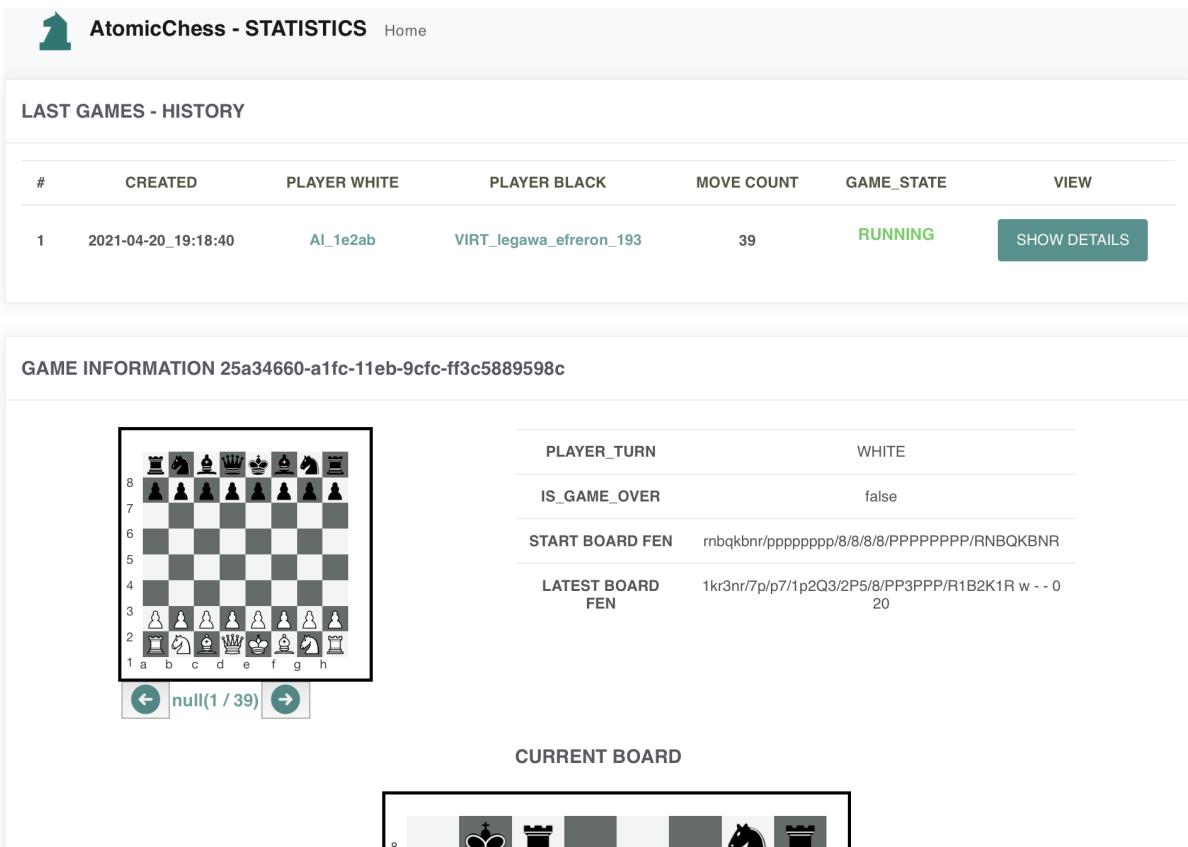
Während der Implementierung wurde der Webclient weiter ausgebaut und es wurde weitere Eigenschaften ergänzt. Dazu zählt zum einen eine Übersicht über vergangene und aktuell laufende Spiele. In dieser können Spiele Zug um Zug nachvollzogen werden und weitere Information über den Spielstatus angezeigt werden.?? Auch ist es möglich, aktuell laufende Spiele in Echtzeit anzeigen zu lassen; somit wurde eine Livestream-Funktionalität implementiert.

### 0.6.7 AutoPlayer

Der AutoPlayer-Service stellt den Computerspieler bereit.

Jede Service-Instanz stellt einen virtuellen Spieler bereit, welcher die gleichen Schnittstellen wie der Webclient oder der autonome Schachtisch verwendet. Die einzige Änderung an den verwendeten REST-Calls ist der Login-Requst. Hier wird das `playertype` Flag gesetzt welches den Spieler als Computerspieler gegenüber dem System authentifiziert. Daraus resultierend wird dieser während des Matchmaking-Prozesses erst für ein Match ausgewählt, wenn keine anderen menschlichen Spieler mehr zur Verfügung stehen. Dieser digitale Gegenspieler ist vom Typ Webclient oder autonomer Schachtisch. Dieser Prozess gewährleistet zudem, dass immer zuerst die menschlichen Spieler ein Spiel beginnen und die digitalen nur Alternativen darstellen.

Eine weitere Modifikation ist die Verwendung einer Schach-AI, da dieser Service als

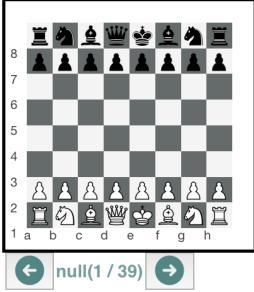


The screenshot shows the 'LAST GAMES - HISTORY' section of the AtomicChess statistics page. It lists a single game entry:

#	CREATED	PLAYER WHITE	PLAYER BLACK	MOVE COUNT	GAME_STATE	VIEW
1	2021-04-20_19:18:40	AI_1e2ab	VIRT_legawa_efreron_193	39	RUNNING	<a href="#">SHOW DETAILS</a>

Below this is the 'GAME INFORMATION' section for game ID 25a34660-a1fc-11eb-9cfc-ff3c5889598c. It displays the current board position and game details:

**CURRENT BOARD**



Board FEN: rnbqkbnr/pppppppp/8/8/8/PPPPPPP/RNBQKBNR  
Move: 1kr3nr/7p/p7/1p2Q3/2P5/8/PP3PPP/R1B2K1R w - - 0  
Turn: WHITE  
Is Game Over: false

**CURRENT BOARD**



Bild 0-15: Webclient: Statistiken

Computerspieler agieren soll. Hierzu kam die Open-Source Chess Engine Stockfish[?] in der Version 11 zum Einsatz. Die Stockfish-Engine bietet noch weitere Features, als nur die nächstbesten Züge zu einem gegebenen Schachbrett zu ermitteln.

Die AutoPlayer-Instanz kommuniziert über das Universal Chess Interface (UCI) Protokoll[?] mit der Engine. Dieses Protokoll wird in der Regel von Schach-Engines verwendet, um mit einer Graphical User Interface (GUI) zu kommunizieren.

Um das aktuelle Spielbrett in der Engine zu setzen wird dieses in der X-FEN Notation mit dem Prefix `position fen` als Klartext an die Engine übergeben und sendet daraufhin eine List möglicher Züge zurück. Der erste Index dieser Liste ist dabei der am besten bewerteten Zug der Engine.

Im Kontext des AutoPlayer-Service wird der Engine nur das aktuelle Spielbrett übermittelt und der nächstbeste Zug auslesen. Dies wird ausgeführt, wenn der AutoPlayer am Zug ist. Nachdem die Engine einen passenden Zug gefunden hat, wird das Ergebnis über den `make_move` REST-API Call übermittelt.

Wenn das Match beendet wird, beendet sich auch die Service-Instanz. Diese wird je-

doch wieder gestartet, wenn die Anzahl der zur Verfügung stehenden Computerspieler unter einen definierten Wert fallen. Somit ist dafür gesorgt, dass das System nicht mit ungenutzten AutoPlayer-Instanzen gebremst wird. Diese Anzahl ?? ist in der Konfiguration des Backend-Service frei wählbar und kann je nach zu erwarteten Aufkommen angepasst werden.

Allgemein skaliert das System durch diese Art der Ressourcenverwaltung auch auf kleinen Systemen sehr flexibel. Durch die Art der Implementierung, dass sich der AutoPlayer-Service wie ein normaler Spieler verhält, sind auch andere Arten des Computerspieler möglich. So ist es zum Beispiel möglich, die Spielstärke je Spieler anzupassen oder einen Computerspieler zu erstellen, welcher nur zufällige Züge zieht.

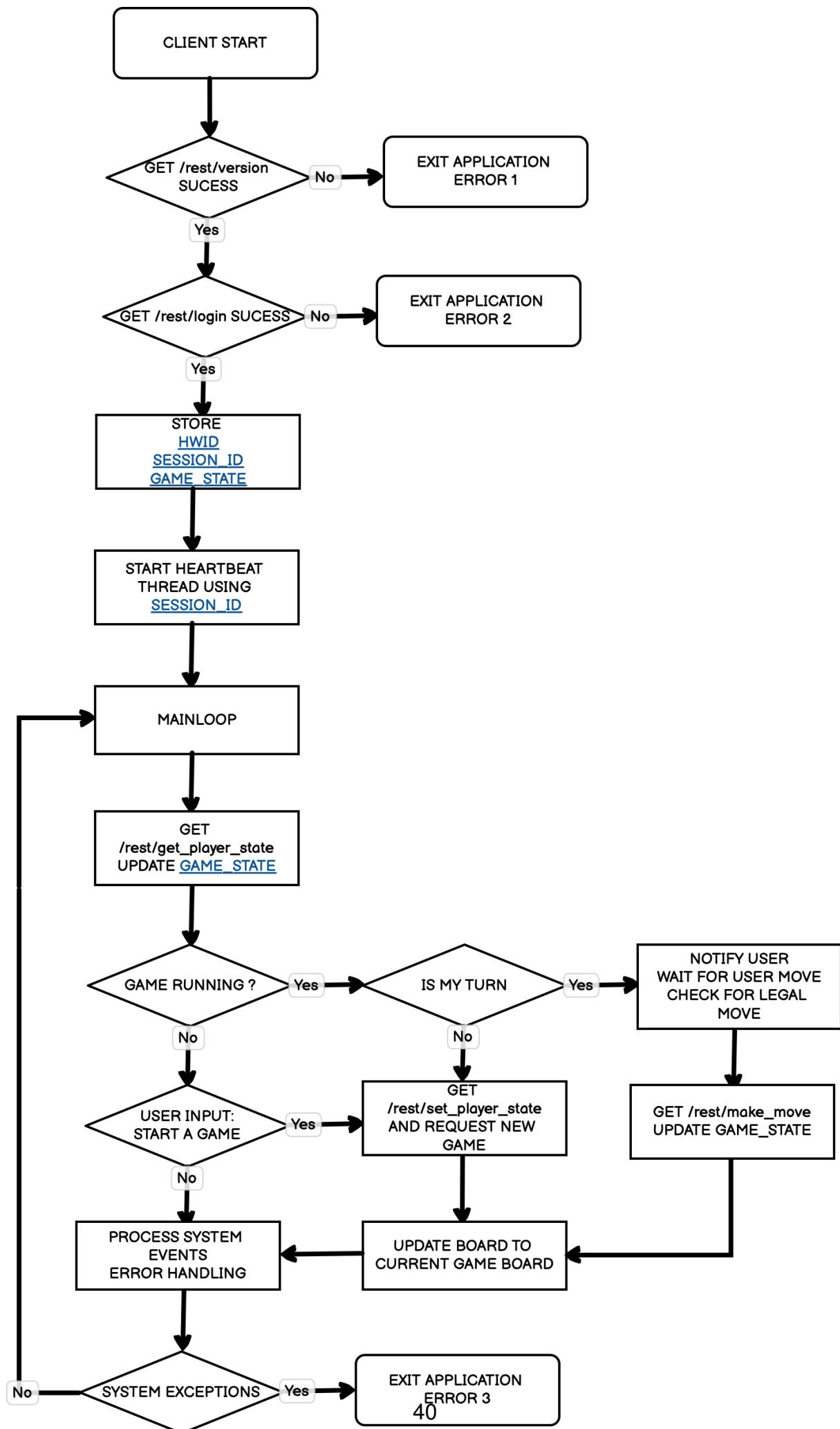
Ein weiterer Anwendungsfall für den AutoPlayer-Service, ist das Testen des weiteren Systems insbesondere des Backend-Service. Durch das Erstellen eines Spiels mit zwei AutoPlayer-Instanzen, können automatisierte Schachpartien ausgeführt werden um die Funktionsfähigkeit des restlichen Systems zu testen. Diese Feature wurde insbesondere bei der Entwicklung des WebClient und der Steuerungssoftware für den autonomen Schachtisch verwendet.

## 0.7 Embedded System Software

- Hauptsoftware zur Steuerung der Elektrik/Mechanik
- Kommunikation mit dem Cloud-Server

### 0.7.1 Ablaufdiagramm

- dummer/thin Client
- Synchronisierung von gegeben Schachfeld mit dem lokalen Schachfeld
- getätigte züfe werden direkt an den schachserver geschickt und dieser generiert daraufhin das neue schachbrett welches von beiden Partner sync



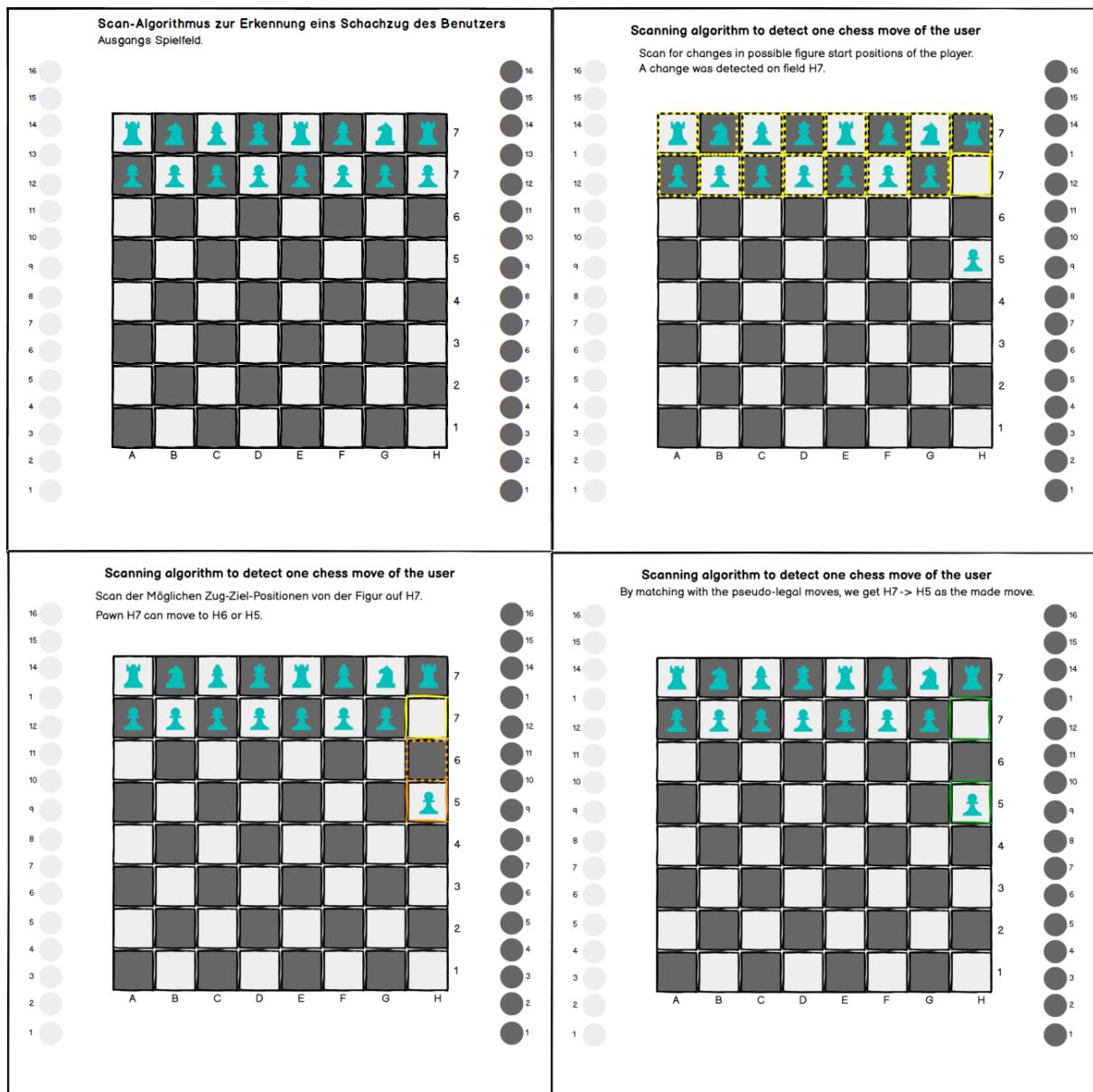


Bild 0-17: Embedded System Software: Schachfeld Scan Algorithmus Ablauf

### 0.7.2 Figur Bewegungspfadberechnung

- Algorithmus zur Umsetzung eines Schachzugs
- Auf trennung in current und target Board
- vier Schritte (entfernen, bewegen, hinzufügen, bewegen) ## Schachfeld Scan Algorithmus zur Erkennung von Schachzügen
- Benutzer bestätigt, dass er Schachzug gemacht hat
- Ermittlung des getätigten Schachzugs
- Scan der Schachfeld-Veränderungen, durch Vergleich des vorherigen Schachfelds und der möglichen Züge

### 0.7.3 Inter Prozess Communication

Bei der Entwicklung des Systems wurde darauf geachtet, dass sich das User-Interface austauschen lässt. Somit ist es auch möglich, ein webbasiertes User-Interface zu integrieren. Dazu wurde ein zusätzliches Interprocess Communication (IPC) Layer hinzugefügt, welches eine Abstraktion der von der User-Interface Software verwendeten Funktionen auf der Controller-Software Ebene bereitstellt.

Dazu wurde eine einfache IPC Bibliothek implementiert, welche dem Controller- als auch dem User-Interface als Shared-Library zur Verfügung steht. Diese stellt einfache Funktionen zum Senden und Empfangen von Events bereit und erzeugt nach der Initialisierung einen separaten Thread in welcher die Kommunikation mit den anderen IPC Instanzen verwaltet wird.

Der Haupthread des Programms kann anschließend über eine First In – First Out (FIFO) Message Queue, die von den anderen Instanzen empfangenen Events in einer Polling-Loop abfragen und Events an die anderen Instanzen absetzen. Diese können mit der gleichen Vorgehensweise

Die Kommunikation zwischen den IPC Instanzen geschieht hierbei über eine Transmission Control Protocol (TCP) Socket-Verbindung. Es wurde keine Shared Memory (Speicherbasierte) Implementierung verwendet, da hier nur eine Kommunikation auf Betriebssystemebene möglich ist.

Durch die Socket Basierende Implementierung ist es möglich die andern IPC Instanzen auszulagern und auf verschiedenen Endgeräten ausführen zu können.

```
1  {
2    "event":12, //BEGIN_BTN_SCAN
3    "type":2, //CLICKED
4    "dest_process_id":"ui_qt_01",
5    "origin_process_id":"controller_sw_01",
6    "is_ack":false
7 }
```

Über die TCP Verbindung werden ausschließlich Daten im JavaScript Object Notation (JSON) Format übertragen. Dies macht ein einfaches Debugging und Steuerung über einen Webbrower möglich, welches die Implementierung während der Entwicklungsphase vereinfachte.

Zusätzlich kann über die Acknowledgement-Funktionalität sichergestellt werden, dass

die anderen IPC Instanzen dieses Event erhalten haben. Diese müssen nach Erhalt das empfangene Event quittieren, was mittels des `is_ack` Flag zurückgemeldet wird.

```
1 //IPC guicommunicator.cpp
2 //Simplyfied example calls
3
4 //INIT IPC SERVER
5 guicommunicator gui;
6 gui.start_recieve_thread();
7 //CHECK OTHER PROCESS REACHABLE
8 while (!gui.check_guicommunicator_reachable()){
9     gui_wait_counter++;
10    if (gui_wait_counter > GUI_WAIT_COUNTER_MAX){
11        break;
12    }
13 }
14 //...
15 //CHECK OTHER PROCESS VERSION NUMBER
16 if(gui.check_guicommunicator_version()){
17     LOG_F(WARNING, "guicommunicator version check failed");
18 }
19
20 //SWITCH MENU ON SCREEN TO PLEASE WAIT SCREEN
21 gui.createEvent(guicommunicator::GUI_ELEMENT::SWITCH_MENU,
22                 guicommunicator::GUI_VALUE_TYPE::PROCESSING_SCREEN);
23 //FLIP SCREEN ORIENTATION
24 gui.createEvent(guicommunicator::GUI_ELEMENT::
25                 QT_UI_SET_ORIENTATION_180, guicommunicator::GUI_VALUE_TYPE
26                 ::ENABLED);
27
28 //GET EVENT FROM OTHER PROCESSES STORED IN EVENT QUEUE
29 guicommunicator::GUI_EVENT ev = gui.get_gui_update_event();
30 if (!ev.is_event_valid){
31     gui.debug_event(ev, true);
32     continue;
33 }
34 //CHECK FOR USER INPUT
35 if(ev.event == guicommunicator::GUI_ELEMENT::BEGIN_BTN_SCAN &&
36    ev.type == guicommunicator::GUI_VALUE_TYPE::CLICKED) {}
```

### 0.7.4 Userinterface

Das User-Interface ist eines des zentralen Elements mit welchem der Benutzer interagiert. Hierbei soll dieses nur die nötigsten Funktionen bereitstellen, welche zur Bedienung des Schachttisches nötig sind. Durch die kleinen Abmessungen des Displays mit 4.3 Zoll, wurde alle Bedienelemente in ihrer Größe angepasst, sodass der Benutzer auch von einer weiter entfernten Position den Zustand direkt erkennen kann. Auch

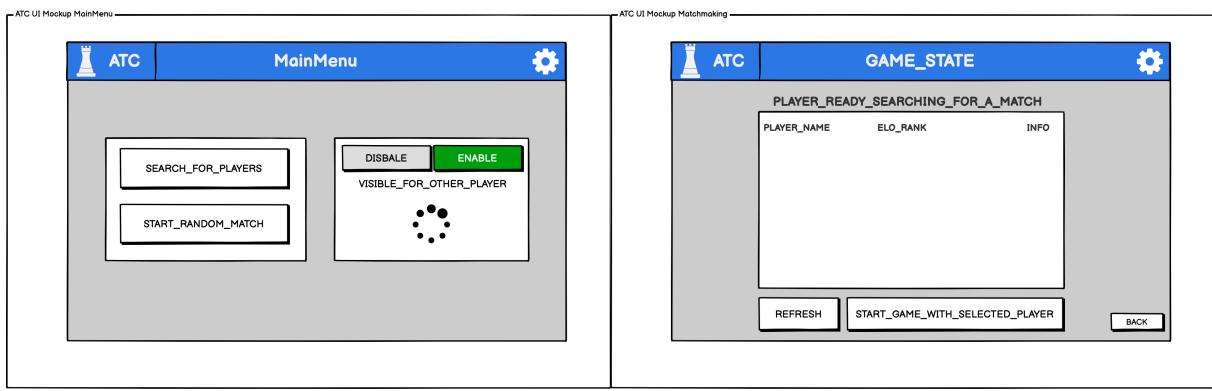


Bild 0-18: Embedded System Software: User-Interface Mockup

wurden die maximale Anzahl an Bedienelementen in einer Ansicht auf drei begrenzt. Die Spielansicht stellt zudem nur die eigene Spielerfarbe, sowie welcher Spieler gerade am Zug ist dar, somit soll der Spieler nicht vom Spiel abgelenkt werden. Nach dem Spielstart findet keine weitere Interaktion mit dem User-Interface mehr statt.

Trotz der Einfachheit der Bedienung und der meist nur also Informationsquelle über den Spielstand dienenden User-Interface, bietet diese viele Möglichkeiten der Konfiguration des Systems. Somit kann auf ein weiteres Eingabegerät, wie z.B. einem Mobiltelefon verzichtet werden, da alle relevanten Einstellungen im Optionen-Menu vorgenommen werden können.

Als Framework wurde hier das Qt[?] verwendet, da dieses bereits im Buildroot-Framework in der Version 5.12 hinterlegt ist. Somit musste kein anderes derartiges Framework aufwändig in das Buildroot-Framework integriert werden.

Das User-Interface wurde gegen Ende der Entwicklung der Controller-Software begonnen, somit waren alle benötigten Ansichten und Funktionen definiert, trotzdem wurden im Vorfeld bereits mögliche Ansichten und Menüstrukturen mittels Wireframing festgehalten und konnten anhand dieser schnell umgesetzt werden.

Das Qt[?] bietet dazu einen separaten Editor [Qt Design Studio](#) an, in denen die zuvor erstellen Wireframe-Grafiken importiert wurden und anschliessen mit den Bedienelementen ersetzt werden könnten. Dieser Prozess gestaltete sich als sehr effizient und so konnte das komplette UI mit moderatem Zeitaufwand umgesetzt werden.

```
1 // WINDOW.qml User-Interface ATC
2 import QtQuick 2.15
3 import QtQuick.Controls 2.15
4 //...
5 Rectangle {
```

```
6      id: window
7      objectName: "window"
8      width: 800
9      height: 480
10     //BACKEND LOGIC INIT => CREATES INSTANCE OF THE
11     MenuManager CLASS
12     MenuManager{
13         id:main_menu
14         objectName: "mainmenu"
15     }
16     //...
17     // MAIN MENU CONTAINER
18     Rectangle {
19         id: mm_container
20         objectName: "mm_container"
21         property var headline_bar_name:"Main Menu"
22         //START AI MATCH BUTTON
23         Button {
24             id: mm_start_random_btn
25             x: 40
26             y: 183
27             width: 207
28             height: 55
29             text: qsTr("START AI MATCH")
30             //CONNECT BUTTON EVENTS TO BACKEND LOGIC
31             Connections {
32                 target: mm_start_random_btn
33                 function onClicked(_mouse){
34                     //CALL A FUNCTION IN BACKEND LOGIC
35                     INSTANCE
36                     main_menu.
37                     mm_search_for_players_toggled(true)
38                 }
39             }
40         }
41     }
42 
```

Die anschließende Implementierung der Backend-Logik des Unter-Interface bestand in der Verbindung, der in QML erstellten Bedienelementen durch den [Qt Design Studio](#)-Editor und der User-Interface Backend Logik. Diese beschränkt sich auf die Initialisierung des Fensters und dem anschließenden laden und darstellen des QML Codes. Die Backend-Logik Funktionalitäten in einem QML Typ [MenuManager](#) angelegt, welcher vor dem Laden des eigentlichen User-Interface QML Codes registriert werden muss.

```
1 // main.cpp User-Interface ATC
2 #include <QGuiApplication>
3 #include <QQmlApplicationEngine>
4 #include "menumanager.h" //BACKEND LOGIC
5 int main(int argc, char *argv[])
6 {
```

```
7     QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
8     //...
9     //CREATE WINDOW
10    QWindow window;
11    window.setBaseSize(QSize(800,480));
12
13    //REGISTER MainMenu COMPONENT
14    qmlRegisterType<MenuManager>("MenuManager", 1, 0, "MenuManager"
15        );
15    //LOAD User-Interface QML
16    QQuickView view;
17    //...
18    view.engine()->addImportPath("qrc:/qml/imports");
19    view.setSource(QUrl("qrc:/qml/WINDOW.qml"));
20    view.engine()->rootContext()->setContextProperty("app", &app
21        );
21    //...
22    //IMPORTANT STEP: AFTER INIT THE MainMenu COMPONENT HAS NO
23    //PARENT
23    //SO WE NEED TO SET IT MANUALLY TO MAKE C++ -> QML FUNCATION
24    //CALLS WORKING
24    QObject *object = view.rootObject();
25    QObject *rect = object->findChild<QObject*>("mainmenu");
26    if (rect){
27        rect->setParent(object);
28    }
29    //FINALLY SHOW MENU ON SCREEN
30    view.show();
31 }
```

Da das User-Interface ein separates Programm ist, welches auf dem System ausgeführt wird, muss dieses in der Lage sein mit der Controller-Software zu kommunizieren. Hierzu wurde die zuvor erstellte IPC Bibliothek in das Projekt importiert, jedoch wurde in der Makefile das `USES_QT` Define-Flag gesetzt. Wenn dieses gesetzt ist, wird die Bibliothek in den Client-Modus versetzt und stellt somit das Gegenstück zu der Instanz dar, welche in der Controller-Software läuft. Somit werden auch die Funktionen zum Senden von `gui.createEvent()` umgekehrt, sodass ein Event in der Controller-Software ausgelöst wird. Dies kann z.B. durch eine Benutzereingabe oder wenn das User-Interface die von der Controller-Software geforderten Zustand angenommen hat.

```
1 // menumanager.cpp User-Interface ATC
2 #include "menumanager.h"
3
4 MenuManager::MenuManager()
5 {
6     //START IPC THREAD
7     guiconnection.start_recieve_thread();
8     //...
9 }
```

```
10
11 //METHOD CALLED FROM QML ELEMENT ss_calboard_btn
12 void MenuManager::ss_calboard_btn(){
13     //SEND EVENT TO CONTROLLER SOFTWARE
14     guiconnection.createEvent(guicomunicator::GUI_VALUE_TYPE
15         ::START_CALBOARD_PROC);
16
17 //PROCESSES EVENTS COMMING FROM THE INTER PROCESS
18 // COMMUNICATION AND SHOWS MENUS OR SET IMAGES/LABES
19 // MenuManager::updateProgress() CALLED BY SPERATE THREAD
20 void MenuManager::updateProgress()
21 {
22     //GET LATEST EVENT FROM IPC
23     const guicomunicator::GUI_EVENT ev = guiconnection.
24         get_gui_update_event();
25     if(!ev.is_event_valid){return;}
26     //PROCESS EVENTS
27     //SWITCH MAIN MENU REQUEST
28     if(ev.event == guicomunicator::GUI_ELEMENT::SWITCH_MENU){
29         switch_menu(ev.type);
30     }
31 }
```

## 0.8 Fazit

Zusammenfassend lässt sich feststellen, dass das Ziel der Arbeit erreicht wurde. Es wurde ein Prototyp eines autonomen Schachttischs entwickelt.

- mit am weitesten forgeschrittenen open-source autonomes Schachttisch Projekt
- vom versierten Benutzer selbständig aufbaubar
- leichte bedienung
- lässt spiel für erweiterungen
-

### **0.8.1 Ausblick**

- Einbindung in existierende Schach-Clouds z.B. <https://lichess.org/>
- user-port für Erweiterungen (z.B. DGT Schachur)

# Abbildungsverzeichnis

<b>0-1</b>	Grove PN532 NFC Reader mit Kabelgebundener Antenne . . . . .	12
<b>0-2</b>	3D Druck: Objekt (rot,gelb,grün),Tree Structure (cyan) . . . . .	14
<b>0-3</b>	Prototyp Hardware: Erster Prototyp des autonomen Schachtisch . . . . .	15
<b>0-4</b>	Prototyp Hardware: Tool zur Erstellung des NDEF Payloads: ChessFigureIDGenerator.html . . . . .	17
<b>0-5</b>	Prototyp Hardware: NDEF Text Record Payload für einen weißen Turm . . . . .	18
<b>0-6</b>	Prototyp Hardware: Blockdiagramm . . . . .	19
<b>0-7</b>	Production Hardware: Finaler autonomer Schachtisch . . . . .	20
<b>0-8</b>	Production Hardware: Blockdiagramm . . . . .	24
<b>0-9</b>	Gesamtübersicht der verwendeten Cloud-Infrastruktur . . . . .	31
<b>0-10</b>	Cloud-Infrastruktur: Aufbau einer URI . . . . .	31
<b>0-11</b>	Cloud-Infrastruktur: Aufbau der Service Architecture . . . . .	33
<b>0-12</b>	Cloud-Infrastruktur: Backend Login-Request und Response . . . . .	33
<b>0-13</b>	MoveValidator: Beispiel Request zur Ausführung eines Zuges auf einem gegebenen Schachbrett . . . . .	34
<b>0-14</b>	Webclient: Spielansicht . . . . .	36
<b>0-15</b>	Webclient: Statistiken . . . . .	38
<b>0-16</b>	Embedded System Software: Ablaufdiagramm . . . . .	40
<b>0-17</b>	Embedded System Software: Schachfeld Scan Algorithmus Ablauf . . . . .	41
<b>0-18</b>	Embedded System Software: User-Interface Mockup . . . . .	44

# Tabellenverzeichnis

0.1	Auflistung kommerzieller autonomer Schachttische . . . . .	5
0.2	Auflistung von Open-Source Schachttisch Projekten . . . . .	8
0.3	Auflistung der Anforderungen an den autonomen Schachttisch . . . . .	10
0.4	Verwendete 3D Druck Parameter. Temperatur nach Herstellerangaben des verwendeten PLA Filament. . . . .	14
0.5	Standardhardware 3D Drucker Steuerungen . . . . .	23
0.6	Grundlegende verwendete G-Code Kommandos . . . . .	25
0.7	Hinterlegte G-Code Steuerungen . . . . .	25
0.8	Hinterlegte Mikrocontroller . . . . .	29
0.9	Eigenschaften die finalen Prototypen . . . . .	30
0.10	MoveValidator-Service API Overview . . . . .	34
0.11	Vergleich FEN - X-FEN . . . . .	35

## **.1 Anhang**