

FH Aachen

Fachbereich Elektrotechnik und Informationstechnik

Studiengang Informatik

Bachelorarbeit

Integration eines eingebetteten Systems in eine Cloud-Infrastruktur am Beispiel eines autonomen Schachspiels

vorgelegt von

Marcel Werner Heinrich Friedrich Ochsendorf

Matrikel-Nr. **3120232**

Referent:

Prof. Dr.-Ing. Thomas Dey

Korreferent:

Prof. Dr. Andreas Claßen

Datum:

11.06.2021

1 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 _____

2 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

1	Erklärung	i
2	Abstract	ii
0.3	Einleitung	1
0.3.1	Motivation	1
0.3.2	Zielsetzung	1
0.3.3	Methodik	1
0.4	Analyse bestehender Systeme und Machbarkeitsanalyse	2
0.4.1	Existierende Systeme im Vergleich	2
0.4.2	User Experience	5
0.4.3	Anforderungsanalyse	6
0.4.4	Machbarkeitsanalyse	7
0.5	Grundlegende Verifikation der ausgewählten Technologien	8
0.5.1	Erprobung Buildroot-Framework	8
0.5.2	Verifikation NFC Technologie	8
0.5.3	Schrittmotor / Schrittmotorsteuerung	8
0.5.4	3D Druck für den mechanischen Aufbau	10
0.6	Erstellung erster Prototyp	11
0.6.1	Mechanik	11
0.6.2	Parametrisierung Schachfiguren	11
0.6.3	Schaltungsentwurf	15
0.6.4	Fazit zum ersten Prototypen	17
0.7	Erstellung zweiter Prototyp	17
0.7.1	Modifikation der Mechanik	17
0.7.2	Optimierungen der Spielfiguren	18
0.7.3	Änderungen der Elektronik	19
0.7.4	Fazit zum finalen Prototypen	25
0.8	Entwicklung der Cloud Infrastruktur	26
0.8.1	API Design	27
0.8.2	Service Architektur	28

0.9	Embedded System Software	35
0.9.1	Ablaufdiagramm	35
0.9.2	Figur Bewegungspfadberechnung	37
0.9.3	Userinterface	38
0.10	Fazit	38
0.10.1	Ausblick	39
	Literaturverzeichnis	40
	Abbildungsverzeichnis	44
	Tabellenverzeichnis	45
.1	Anhang	46

0.3 Einleitung

0.3.1 Motivation

- Beginn: Er zieht die Aufmerksamkeit des Lesers durch die Schilderung des Ereignisses auf sich, das zu dem Problem geführt hat.
- Hintergrundinformationen (Herstellung des Kontexts): Gehe tiefer auf das Ereignis ein, indem du mehr Informationen über es vermittelst und dabei auch den Rahmen deiner Forschung skizzierst.
- Brücke zur Problemstellung: Erläutere, inwiefern es sich hierbei um ein Problem handelt, und schlage somit die Brücke zur Problemstellung, die deiner Untersuchung zu Grunde liegt.

0.3.2 Zielsetzung

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

0.3.3 Methodik

Im ersten Abschnitt werden die zum Zeitpunkt existierenden Ansätze und deren Umsetzung beleuchtet. Anschliessend werden die zuvor verwendeten Technologien betrachtet, welche bei den beiden darauffolgenden Prototypen verwendet wurden.

Das sechste Kapitel widmet sich der realisierung des erste Prototypen des autonomen Schachtischs. Dabei werden alle im anschliessenden Kapitel, wird auf der Basis des

ersten Prototyps und seiner auftretenden Probleme, der finale Prototyp entwickelt. Hier werden die Probleme durch ein re-design und Vereinfachung der Elektronik gelöst und so ein zufriedenstellendes Produkt entwickelt.

Im darauffolgenden Abschnitt wird die Cloud-Infrastruktur thematisiert, welche für eine Kommunikation zwischen den Prototypen entscheidend ist.

- controller software
- fazit

0.4 Analyse bestehender Systeme und Machbarkeitsanalyse

0.4.1 Existierende Systeme im Vergleich

- Nischenprodukt, jedoch einige Projekte im OpenSource Bereich verfügbar
- Ein kommerzieller Hersteller

Kommerzielle Produkte

Tabelle 0.1: Auflistung kommerzieller autonomer Schachttische

	Square Off - Kingdom [6]	Square Off - Grand Kingdom [5]	DGT Smart Board [2]	DGT Bluetooth Wenge [1]
Erkennung	nein (Manuell per	nein (Manuell per Ausgangsposition)	ja	ja
Figurstellung	Ausgangsposition)			
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

	Square Off - Kingdom [6]	Square Off - Grand Kingdom [5]	DGT Smart Board [2]	DGT
				Bluetooth Wenge [1]
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 erforderlich)	nein (PC erforderlich)
Besonderheiten	Akku für 30 Spiele	Akku für 15 Spiele	-	-

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.

Open-Source Projekte

Bei allen Open-Source Projekten wurden die Features anhand der Beschreibung und der aktuellen Software extrahiert. Besonders bei work-in-progress Projekten können sich

die Features noch verändern und so weitere Funktionalitäten hinzugefügt werden.

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 [16] oder eines modifizierten 3D-Druckers[8] 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) [4]	Automated Chess Board (Akash Ravichandran) [10]	DIY Super Smart Chessboard [7]
Erkennung Figurstellung	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	nein (PC erforderlich)	ja	ja
Funktionalität	-	Sprachsteuerung (Amazon Alexa)	Zuganzeige über LED Matrix
Besonderheiten	General Public License (GPL) 3+	GPL	-
Lizenz			

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 wird in der Regel eine einfache X- und Y-Achse verwendet, welche von zwei Schrittmotoren bewegt werden. Die Schachfiguren werden dabei mittels eines Elektromagneten über die Oberseite gezogen. Hierbei ist ein Magnet oder eine kleine Metallplatte 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 Figurtypen. Dieses Problem wird durch eine definierte Ausgangsstellung beim Spielstart gelöst. Nach jedem Zug durch den Spieler und der dadurch resultierenden Änderungen in der Figurpositionen in der Matrix können die neuen Figurstellungen berechnet werden.

0.4.2 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[15] beschrieben. Hierbei 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. Hierbei 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[5], 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.

0.4.3 Anforderungsanalyse

- komplettes vollwertiges Produkt

Tabelle 0.3: Auflistung der Anforderungen an den autonomen Schachtisch

	Atomic Chess Table (ATC) – autonomous Chessboard
Erkennung Schachfigurstellung	ja
Konnektivität	WLAN, USB
Automatisches Bewegen der Figuren	ja
Spiel Livestream	ja
Cloud anbindung (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 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.

- abmessungen und gewicht ergeben sich aus der Umsetzung der Anforderungen

Technologien im Makerspace

- stehen diese im makerspace zur verfüfung

0.4.4 Machbarkeitsanalyse

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

0.5 Grundlegende Verifikation der ausgewählten Technologien

0.5.1 Erprobung Buildroot-Framework

- erstellen eines einfachen images für das embedded System
- inkl ssh Server und SFTP
- qt 5 libraries
- eigenes package atctp
- test der toolchain

0.5.2 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

0.5.3 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

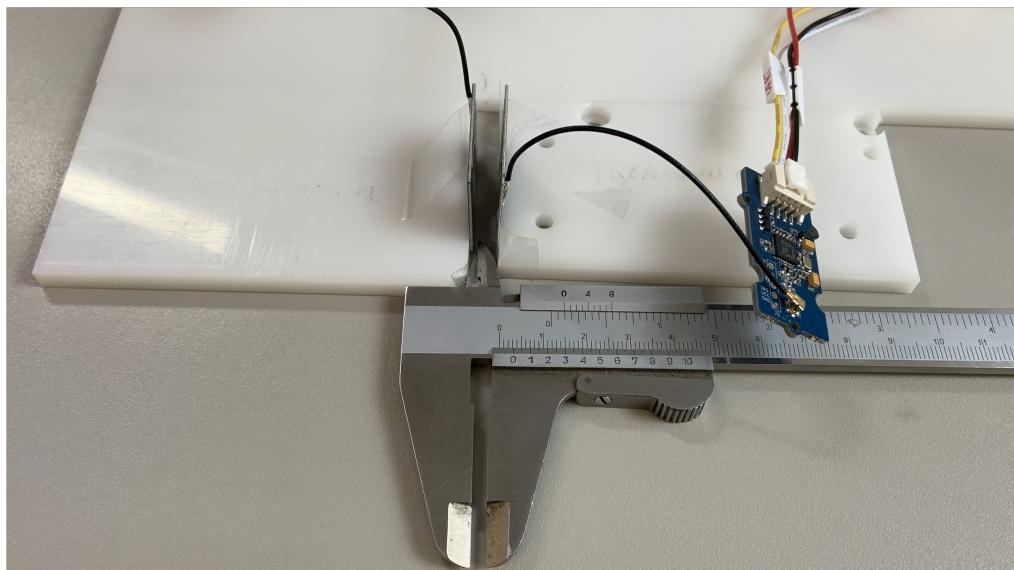


Bild 0-1: Grove PN532 NFC Reader mit Kabelgebundener Antenne

0.5.4 3D Druck für den mechanischen Aufbau

Da es sich hier nur um einen Prototypen handelt, wurde hier auf ein einfach zu handhabendes Filament vom Typ PLA verwendet. Dies 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 Calibrierobjekte gedruckt, an welchen die Toleranzen für die späteren Computer-Aided Design (CAD) Zeichnungen abgeschätzt werden können.

Dies betrifft vor allem die Genuigkeit 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 verarbeitet werden können.

Als Software wurde der Open-Source Slicer Ultimaker Cura [9] verwendet, da dieser zum einen 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.

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

Ender 3 Pro 0.4mm Nozzle	PLA Settings
Layer Height	0.2mm
Infill	50.00%

Ender 3 Pro 0.4mm Nozzle	PLA Settings
Wall Thickness	2.0mm
Support Structure	Tree
Top Layers	4
Bottom Layers	4

- Zeit für den 3D Druck Prozess spielt hier keine Rolle, da selbstbau projekt
- Parameter lassen sich weiter anpassen z.B. Layerheight auf 0.4
- finaler prototyp bietet sich abs oder pteg an

0.6 Erstellung erster Prototyp

- proof of concept
-

0.6.1 Mechanik

- vorgaben IKEA tisch als grundbasis => bereits fertiger grundrahmen in denen die einzelteile integriert werden können
- xy riemen führung
- spiel in Mechanik
- Einabrietung in Fusion360
- Cad design aller bauteile
- standartxy
- erweiterung des spielraums durch zwei Magnete

0.6.2 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

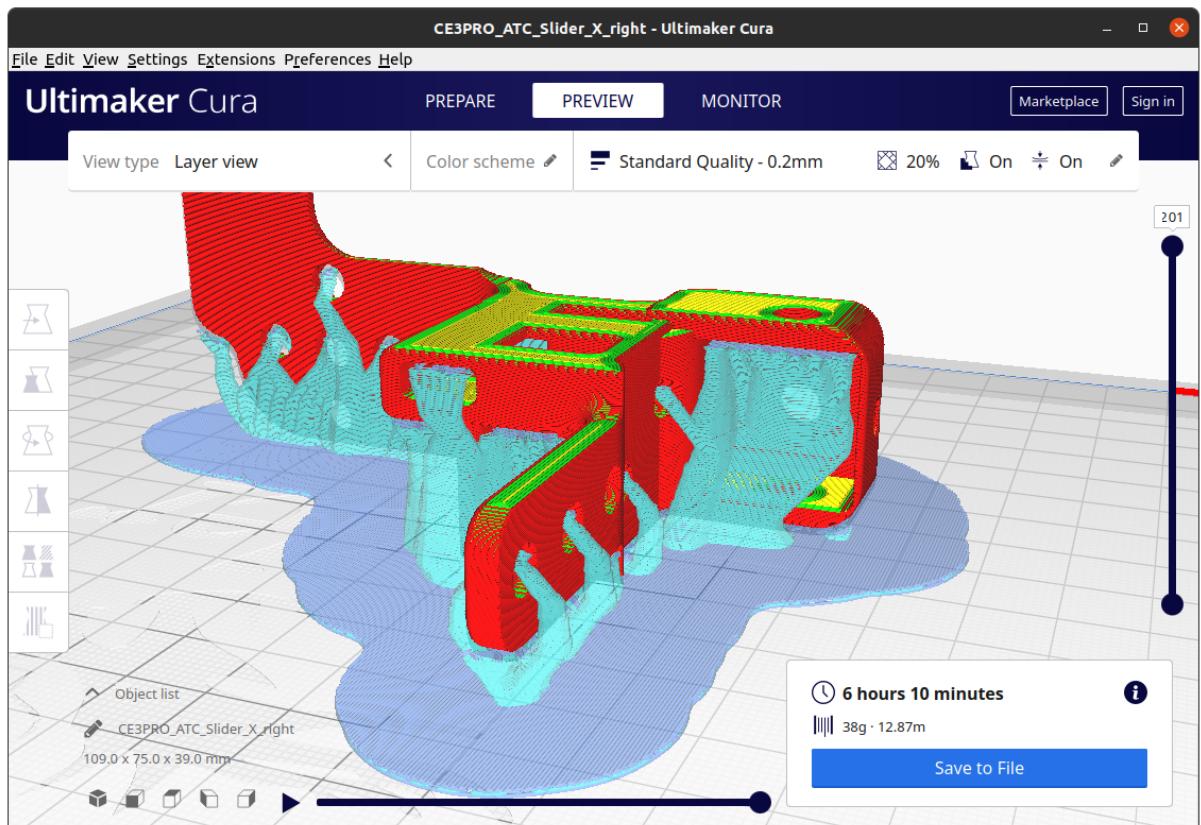


Bild 0-2: 3D Druck: Objekt (rot,gelb,grün),Tree Structure (cyan)



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

Speicher von 180 Byte. Dieser kann über ein NFC-Lese/Schreibgerät mit Daten verschiednster 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 Mobiltelefon und passender App geschehen. Da NDEF Informationen über die Formatierung und der gepeicherten 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 Dateiformaten 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.[12]

Der autonome Schachtisch verwendet den einfachsten NDEF-RTD Typ, welcher der Text-Record ist, und zum Speichern von Zeichenketten genutzt werden kann, ohne dass eine Aktion auf dem Endgerät ausgeführt wird. Jeder Tag einer Schaffigur, welche für den autonomen Schachtisch verwendet werden kann, besitzt diesen NDEF Record an der ersten Position. Alle weiteren eventuell vorhandenen Records werden vom Tisch ignoriert.[11]

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 möglich diesen gegen auslesen oder erneut schreiben mittels einer Read/Write-Protection zu schützen. Diese Funktionalität

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]	=Ñ=
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

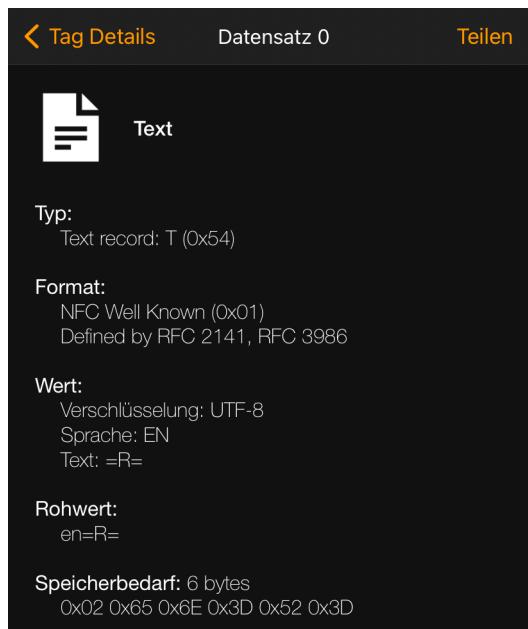


Bild 0-5: Prototyp Hardware: NDEF Text Record Payload für einen weissen Turm

wird nicht verwendet um das Kopieren von Figuren durch den Benutzer zu ermöglichen. Somit kann dieser leicht seine eigenen Figuren erschaffen, ohne auf das Tool angewiesen zu sein. Auch ist es möglich verschiedene Figur-Sets zu mischen, so kann jeder Spieler sein eigenes Set an Figuren mit dem autonomen Schachtisch verwenden.

0.6.3 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
- welche funktion stehen bereit tabelle

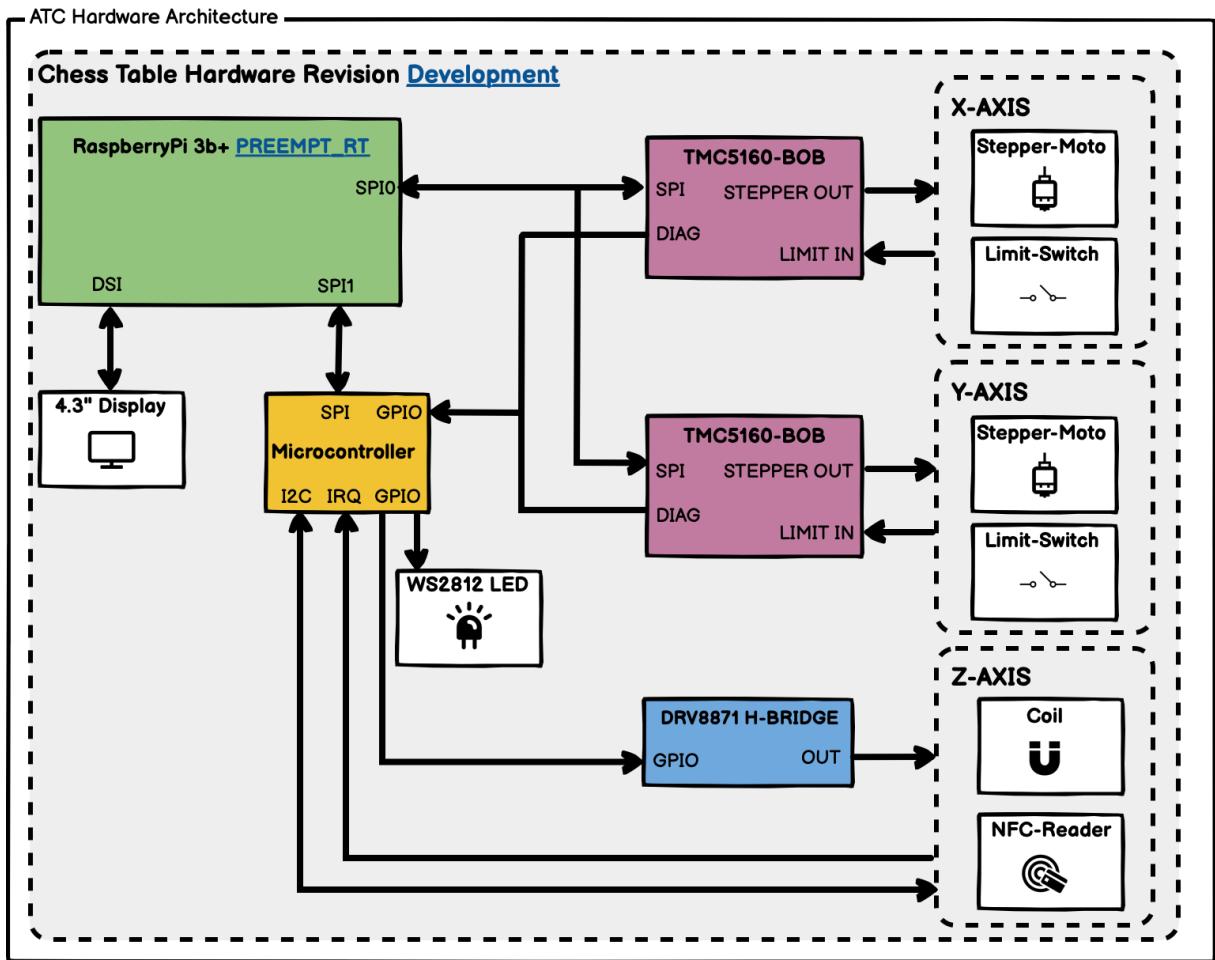


Bild 0-6: Prototyp Hardware: Blockdiagramm



Bild 0-7: Production Hardware: Finaler autonomer Schachtisch

0.6.4 Fazit zum ersten Prototypen

- nicht für production geeignet
- aufbau und calibrierung langwierig
- trotzdem robustes design auf kleinem formfaktor
- verwendeten elektromagnete nicht stark genug, somit über aquusserhalb 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

0.7 Erstellung zweiter Prototyp

0.7.1 Modifikation der Mechanik

- dauertest hat gezeigt dass mechanik zu viel spiel aufweisst
- Motorenhalterung der y achse schränkt des bewegungsspielraum um mehr als 10cm ein, welches zu einem unwesentlichen grösseren verhältnis von Spielfeld-grösse 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

0.7.2 Optimierungen der Spielfiguren

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

Die größte der Figuren kann durch die fest definierte Feldgrösse von 55mm und der verwendeten NFC Tags nicht verändert werden. Nach vielen Testdurchläufen mit dem ersten Prototyp war zu erkennen, dass sich die Figuren je nach aktueller Situation auf dem Spielfeld immernoch magnetisch anziehen. Dies führt je nach Spielverlauf zu Komplikationen, sodass die Figuren manuell wieder mittig auf den Felder platziert werden müssen.

Um dies zu verhindern, wurde einige Figuren zusätzlich mit einer 20mm Unterlegscheibe am Boden versehen, welche diese Problem behob, jedoch das NFC Tag nicht mehr als lesbar erwies. Dies resultierte in der Idee 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 Protoyp wiegen 8 Gramm für die Bauern und 10 Gramm für die restlichen Figuren. Der Test mit der Unterlegscheibe ergab, dass diese mit 4 Gramm genug Gewicht hinzufügte um die magnetische Beeinflussung zu unterbinden.

Testweise wurden eingie Figuren mittels 3D Drucker erstellt um so das Gewicht zu erhöhen. Nach einem erfolgreichen Test wurde das CAD Modell wurde so angepasst, dass sich der Magnet direkt in den Boden der Figure einkleben lässt. Desweiteren wurden bei den Bauern (den leichtesten) Figuren 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 durchgeföhrten Testläufen keine Beeinflussungen mehr statt.

0.7.3 Änderungen der Elektronik

Mit ein grösserer Kritikpunkt, welcher bereits während des Aufbaus des ersten Protoyps 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 grossen 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 andere alternativen in Frage kommen. Zusätzlich konnte die Elektronik nur beschränkt mit anderen System verbunden werden, welches insbesondere durch die verwendete Serial Peripheral Interface (SPI) Schnittstelle geschuldet war.

All diese Faktoren erschweren einen einfachen Zusammenbau des autonomen Schachttischs. Die Lösung stellt die Verwendung von Standardhardware dar. Nach dem Herunterbrechen der elektrischen Komponenten und des mechanischen Aufbaus ist zu erkennen, dass der autonome Schachtisch einer CNC-Fräse bzw eines 3D Drucker ähnelt. Insbesondere die XY-Achsen Mechanik sowie die ansteuerung von Schrittmotoren, wird in diesen Systemen verwendet. Mit den Druchbruch von 3D Druckern im Consumerbereich, sind auch kleine Steuerungen, preisgünstige Steuerungen erhältlich, welche 2-3 Schrittmotoren und einiges anzusätzlicher 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 Aufwahl dieser wurde vor allem auf die Möglichkeit geachtet sogenannte Silent-Schrittmotortreiber verwenden zu können um die Geräuschimmissionen durch die Motoren soweit 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

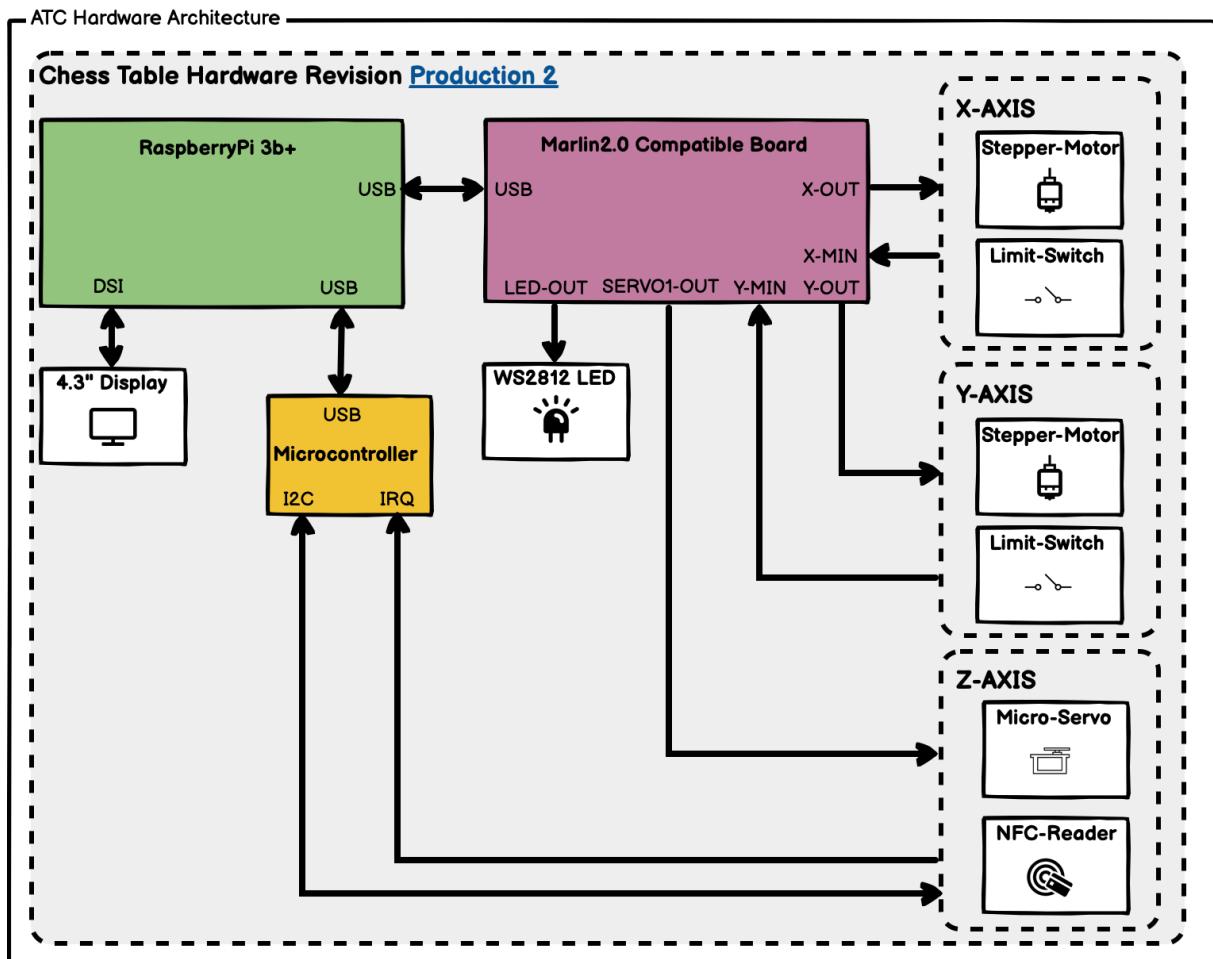


Bild 0-8: Production Hardware: Blockdiagramm

nicht fest verlötet 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 etwas höheren Marktpreises genug Ressourcen auch für spätere Erweiterung bietet und eine Unterstützung für die neuste Version der Marlin-FW[17] 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[17] 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[17] 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.

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 eine 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 welchem spezielle Ausgänge für LEDs gesteuert werden können. Dieses Feature bietet die verwendete Marlin-FW[17], als auch die verwendete Steuerung an. Sollte die Steuerung solch ein optionales Kommando nicht unterstützen, so werden diese ignoriert und somit können auch preisgünstige Steuerungen verwendet werden.

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 gängisten 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 Steuerung automatisch 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 HAI-Basisklasse bereitstellen. Nach Aufruf einer Funktion zum Ansteuern der Motoren, wird aus den übergeben 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 REPONSE
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             {
49                 break;
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 weitere Fall ist die Busy-Antwort [echo:busy](#). Diese Signalisiert, dass das Kommando noch in der Bearbeitung ist und wird im falle des autonomen Schachttisch bei langen und langsamen Bewegungen der Mechanik ausgegeben. Das System wartet diese Antworten ab, bis eine finale [ok](#)-Antwort zurückgegeben wird, erst dann wird das nächste Kommando abgearbeitet.

HAL: I2C Seriell Umsetzer

Druch 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 möglich. Da dieses mittels Inter-Integrated Circuit (I2C) Interface direkt mit dem eingebetteten Systems verbunden war. Diese Möglichkeit besteht weiterhin, 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 hinzugefügt welcher die I2C Schnittstelle zu einer USB Seriell wandelt. Hierzu wurde ein Atmega328p Mikronkontroller eingesetzt, da dieser

weit verbreitet und preisgünstig zu beschaffen ist. Die Firmware des Mikrocontrollers stellt ein einfaches Kommandobasierte 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    }
27    return 0; //VALID TAGS FROM 1-127
28 }
```

Hier 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_`.

```
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(
```

```
        UBC_COMMAND_READNFC);
10     //...
11     //SPLIT STRING -
12     const std::vector<std::string> re = split(readres,
13         UBC_CMD_SEPERATOR);
14     //READ SECTIONS
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     return fig;
31 }
```

0.7.4 Fazit zum finalen Prototypen

- 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 0.8: Eigenschaften des 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

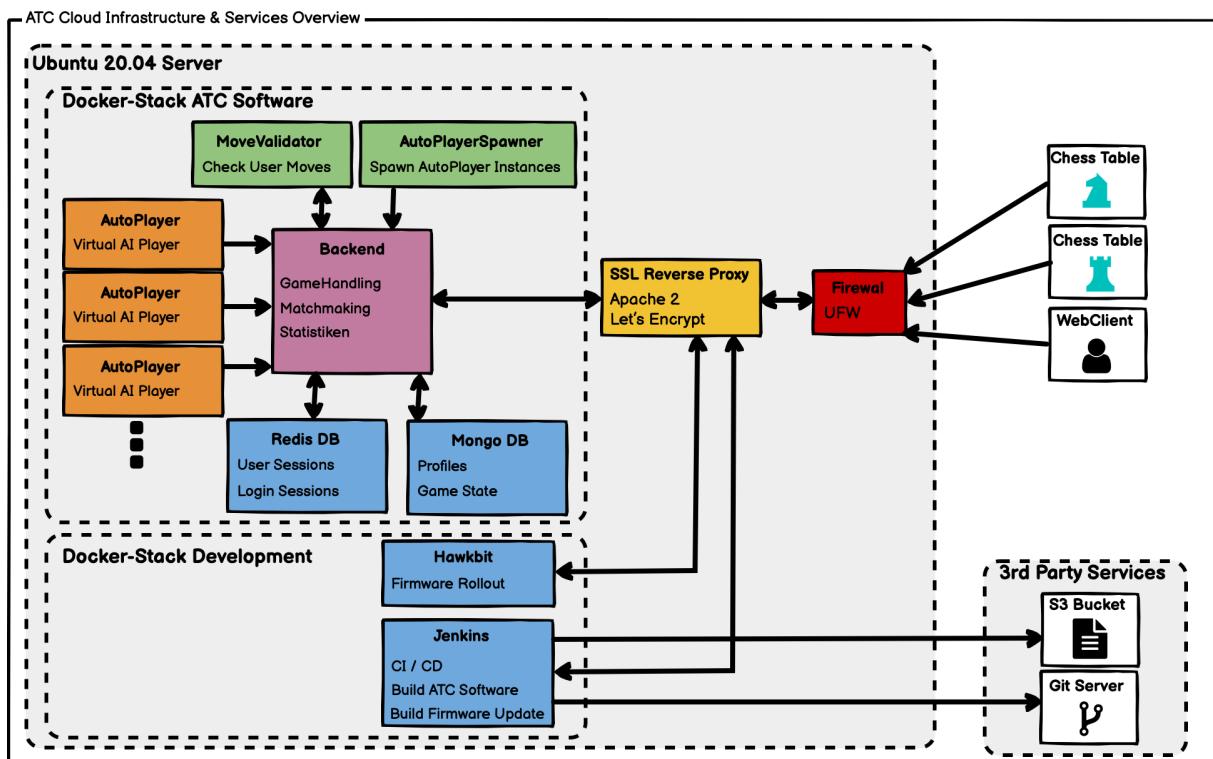


Bild 0-9: Gesamtübersicht der verwendeten Cloud-Infrastruktur

ATC – autonomous Chessboard

Cloud anbindung (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

0.8 Entwicklung der Cloud Infrastruktur

Die erste Phase der Entwicklung des Systems bestand in der Entwicklung der Cloud-Infrastruktur und der darauf laufenden Services. Hierbei stellt die “Cloud”, 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....

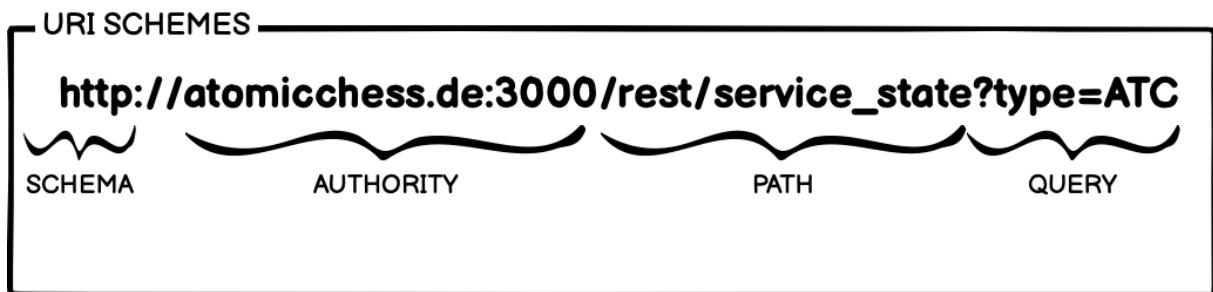


Bild 0-10: Cloud-Infrastruktur: Aufbau einer URI

0.8.1 API Design

Das System soll so ausgelegt werden, dass es im 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 Schachtisch im Browser abbilden kann. Hierzu muss das System eine einheitliche Representational State Transfer (REST)-Schnittstelle bereitstellen.

Eine RESTful API bezeichnet eine API welche HTTP-Requests verwendet um auf Daten zugreifen zu können.

- grafik
- 5 requirements

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 Rückantwort 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. Des Weiteren gibt es viele andere Schema, wie z.B File Transfer Protocol (FTP) welches

Somit stellt die RESTful API eine Interoperabilität zwischen verschiedenen Anwendungen und Systemen bereit, welche durch ein Netzwerk miteinander verbunden sind.

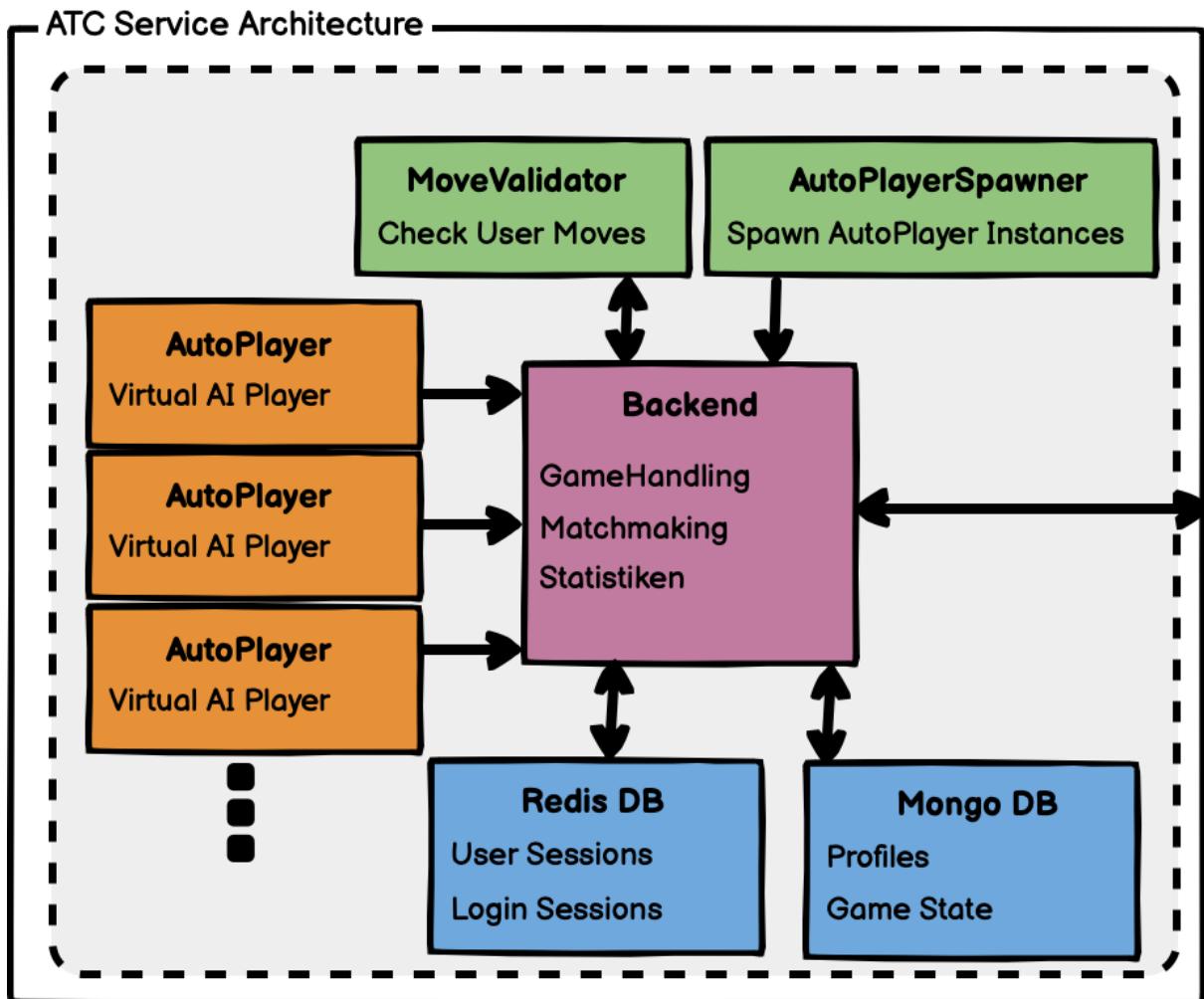


Bild 0-11: Cloud-Infrastruktur: Aufbau der Service Architecture

Dieser Ansatz ist somit geeignet um die verschiedenen Client Systeme (Schachtisch, Webclient) eine Kommunikation mit dem Server zu erlauben.

0.8.2 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 Schachtisches zur Verfügung.

The screenshot shows a POSTMAN interface. At the top, a GET request is made to `http://atomicchess.de:3000/rest/login?hwid=f1ow389djiw&playertype=1`. The 'Params' tab is selected, showing two parameters: `hwid` with value `f1ow389djiw` and `playertype` with value `1`. Below the table, tabs for Body, Cookies, Headers, and Test Results are visible. The status bar indicates a 200 OK response with a time of 130 ms and a size of 865 B. The response body is displayed in JSON format:

```
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

Vorüberlegungen

- welche funktionalitäten müssen abgedeckt werden
- client aktivitendiagramm

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

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

Inhalt

The screenshot shows a POST request to `http://move_validator:5000/rest/execute_move`. The request body is defined as form-data with two fields: `fen` containing the value `rnbqbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQK...
e2e4` and `move` containing the value `e2e4`. The response body is a JSON object:

```
1 {  
2   "err": null,  
3   "move_executed": true,  
4   "new_fen": "rnbqbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR b KQkq - 0 1",  
5   "next_player_turn": 1  
6 }
```

The status is `200 OK` and the time taken is `10 ms`.

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

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 einen neuen Spiel an oder übergibt einen Schachzug inkl. des Spielbretts an den Service.
0-13 Der Response wird dann vom Backend in der Datenbank gespeichert und weiter an die Client-Devices verteilt.

Tabelle 0.9: MoveValidator-Service API Overview

AtomicChess Move-Validator Service 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 verwendet, welcher ein neues Spielbrett in der Forsyth-Edwards-Notation (FEN) Notation zurückgibt, welches zum Start der Partie verwendet wird. Allgemein arbeitet wurde das komplette System so umgesetzt, dass dieses mit einem Spielfeld in einer Zeichenketten/String arbeitet. Dies hat den Vorteil, dass die Spielfeld-Notation leicht angepasst werden kann. Mit diesem Design ist es möglich, auch andere Spielarten im System zu implementieren, nur hier die initialen Spielfelder generiert werden und Züge 30

FEN-TYPE FEN-String

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 Blibliothek [3]. Diese unterstützt zusätzlich die Generierung der für den Benutzer möglichen Schachzügen, welche auf dem aktuellen Brett möglich sind. Diese List wird vom System dazu verwendet um sicherzustellen, das der Benutzer nur gültige Züge tätigen kann. Diese Funktion lässt sich zusätzliche abschalten, falls das Spiel nicht nach den allgemeinen Schachregeln ablaufen soll. Bei der Generierung der möglichen Schachzügen, 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, so sind z.B. auch alle anderen Figur-Züge enthalten, wenn der König sich aktuell im Schach befindet.

Wenn ein Spieler an der Reihe ist, sein getätigter Zug mittels der `/rest/check_move` API überprüft, ob dieser gemäss der Legal-Moves durchführbar ist. Ist dies der Fall, wird der Zug auf das Spielbrett angewendet, welches durch die `/rest/execute_move` API geschieht. Diese führ den Zug aus und ermittelt somit 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 Zusand des Spielbretts vor dem getätigten Zug wiederher. Danach hat der Benutzer die Möglichkeit einen alternativen Zug auszuführen.

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. Dieser wurde dabei komplett in JavaScript (JS) umgesetzt im Zusammenspiel mit Hypertext Markup Language (HTML) und Cascading Style Sheets (CSS) und ist somit komplett im Browser lauffähig.

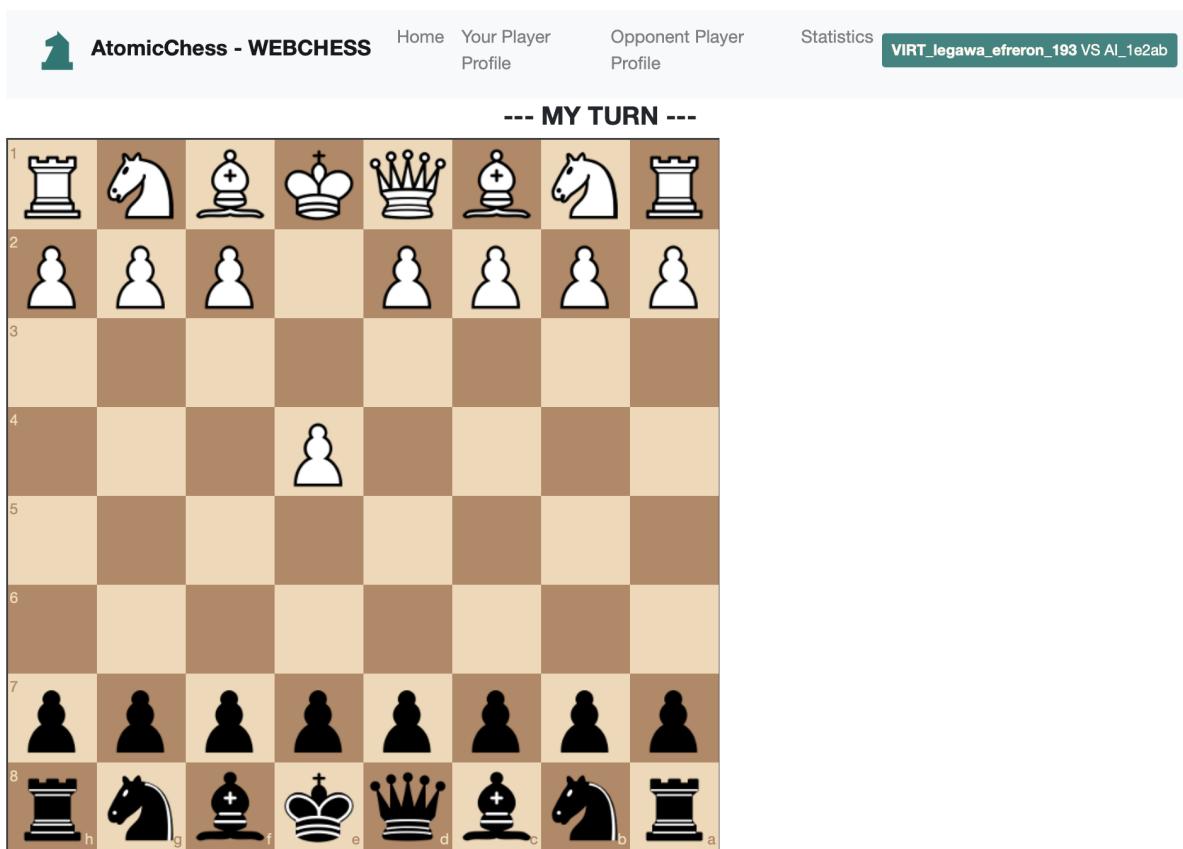
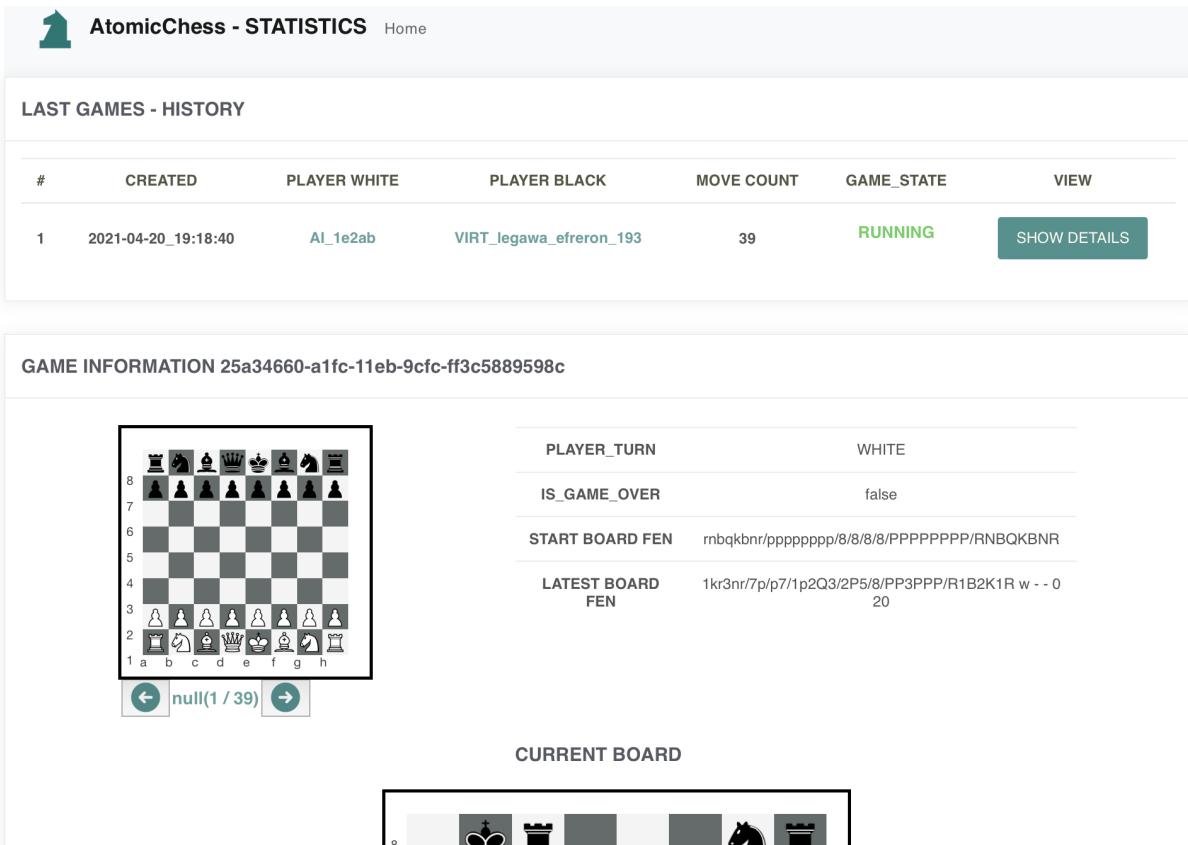


Bild 0-14: WebClient: Spielansicht

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.

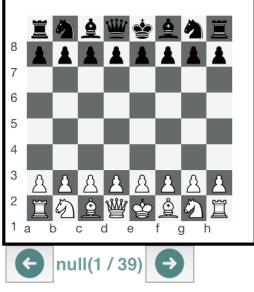
Zusätzlich zu dem verwendeten Vanilla-JS wurde jQuery als zusätzliche JS Bibliothek verwendet, welches eine Manipulation der HTML Elemente stark vereinfacht. Diese bietet insbesondere einfach zu nutzende HTTP-Request Funktionen bzw. Asynchronous JavaScript and XML (AJAX) an, welche für die Kommunikation mit dem Backen-Service verwendet werden. Diese werden im Hintergrund eingesetzt, sodass der WebClient automatisch den neuen Spielzustand dem Benutzer anzeigt. Dies geschieht mittels [polling](#), bei dem der Webbrowser in zyklischen Abständen die aktuellen Spiel-Informationen vom Backen-Service abfragt. Diese Methode wurde verwendet, um eine maximale Kompatibilität mit verschiedenen ggf 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 kann und so eine direkte Kommunikation stattfinden kann ohne Verwendung der [polling](#)-Methode.



The screenshot shows the 'LAST GAMES - HISTORY' section of the web client. It lists one 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	SHOW DETAILS

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



Board FEN: rnbqkbnr/pppppppp/8/8/8/PPPPPPPP/RNBQKBNR

Player Turn: WHITE

Is Game Over: false

Start Board FEN: rnbqkbnr/pppppppp/8/8/8/PPPPPPPP/RNBQKBNR

Latest Board FEN: 1kr3nr/7p/p7/1p2Q3/2P5/8/PP3PPP/R1B2K1R w - - 0
20

Below the board is a navigation bar with arrows and the text 'null(1 / 39)'. The 'CURRENT BOARD' section shows a legend for piece types and colors.

Bild 0-15: Webclient: Statistiken

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

Während der Implementierung wurde der Webclient weiter ausgebaut und es wurde weitere Features 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.

AutoPlayer

Der AutoPlayer-Service stellt den Computerspieler bereit. Jede Service-Instanz stellt einen virtuellen Spieler bereit, welcher die gleiche Schnittstellen wie der Webclient oder der autonome Schachtisch verwendet. Die einzige Änderung an den verwendeten REST-Calls ist der Login-Request. Hier wird das `playertype` Flag gesetzt welches den Spieler als Computerspieler gegenüber des Systems authentifiziert. Somit wird dieser während des Matchmaking-Prozesses erst für ein Match ausgewählt, wenn kein anderer Spieler mehr zur Verfügung steht, welcher vom Typ Webclient oder autonomer Schachtisch ist. Somit ist gewährleistet, dass immer zuerst die Menschlichen-Spieler ein Spiel beginnen.

Eine weitere Modifikation ist die Verwendung einer Schach-AI, da dieser Service als Computerspieler agieren soll. Hierzu kam die Open-Source Chess Engine Stockfish[14] in der Version 11 zum Einsatz. Die Stockfish-Engine bietet noch weitere Features, als nur die nächst besten Züge zu einem gegebenen Schachbrett zu ermitteln. Sie kann auch genutzt werden um Züge zu klassifizieren und bietet Analysemöglichkeiten für Spielstellungen.

Die AutoPlayer-Instanz kommuniziert über das Universal Chess Interface (UCI) Protokoll[13] mit der Executable der Stockfish-Engine. Dieses Protokoll wird in der Regel von Schach-Engines um mit einer Graphical User Interface (GUI) zu kommunizieren und die Kommunikation erfolgt über Klartextbefehle. Um das aktuelle Spielbrett in der Engine zu setzen wird dieses in der FEN Notation mit dem Prefix `position fen` als Klartext an die Engine übergeben.

Im Kontext des AutoPlayer-Service wird der Engine nur das aktuelle Spielbrett übermittelt und der nächst beste Zug ausgelenkt. 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 jedoch 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 Backend-Configuration 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 Matching von zwei AutoPlayer-Instanzen, können automatisierte Schachpartien ausgeführt werden um die Funktionsfähigkeit des restlichen Systems zu testen. Diese Feature wurde insbesondere bei der Entwicklung des Webclienten und der Steuerungssoftware für den autonomen Schachtisch verwendet.

Authetifizierung

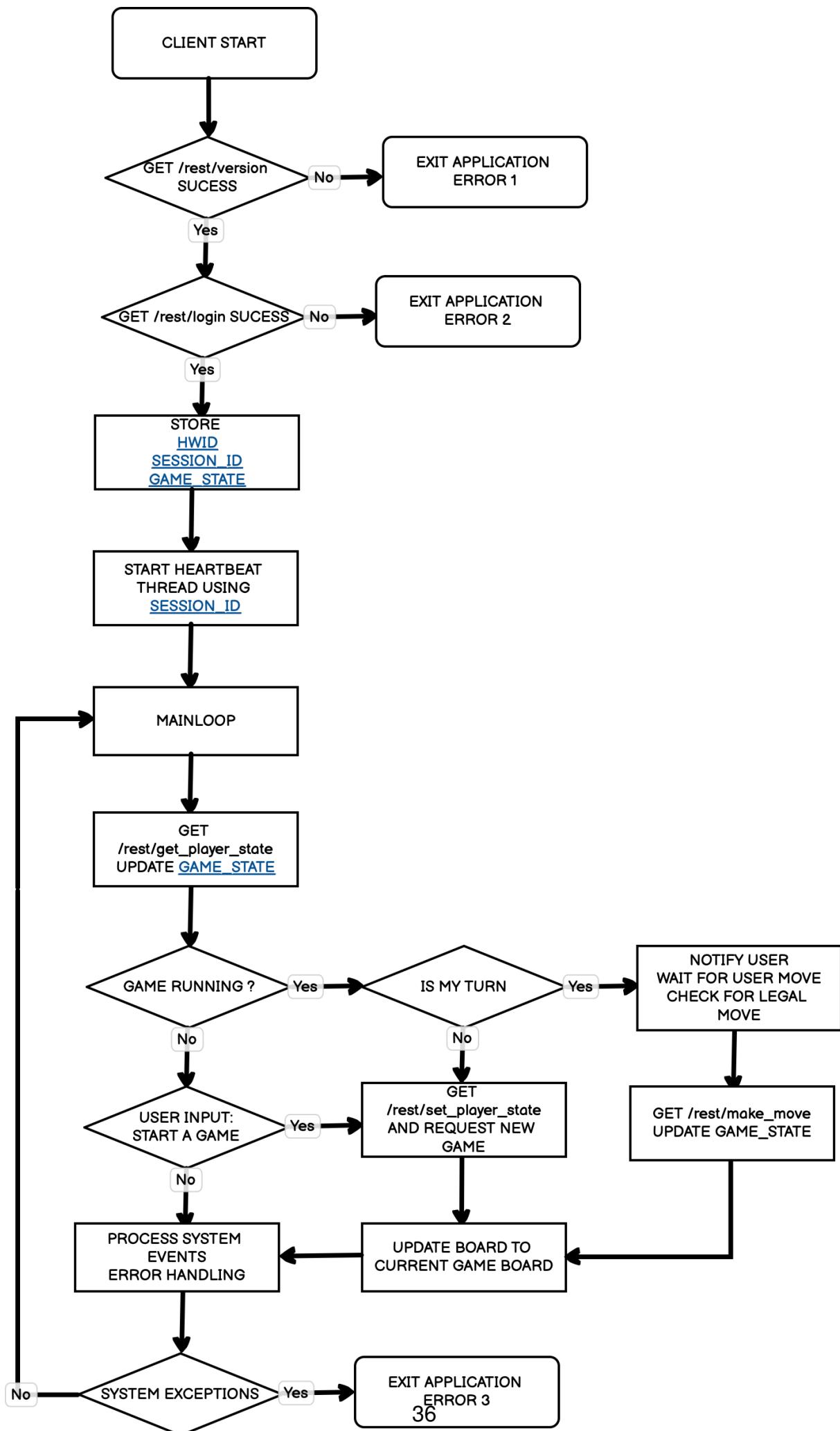
- authetifizierung
- https only
- Zertifikate auf Clientseite generiert jedoch nicht Abgefragt

0.9 Embedded System Software

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

0.9.1 Ablaufdiagramm

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



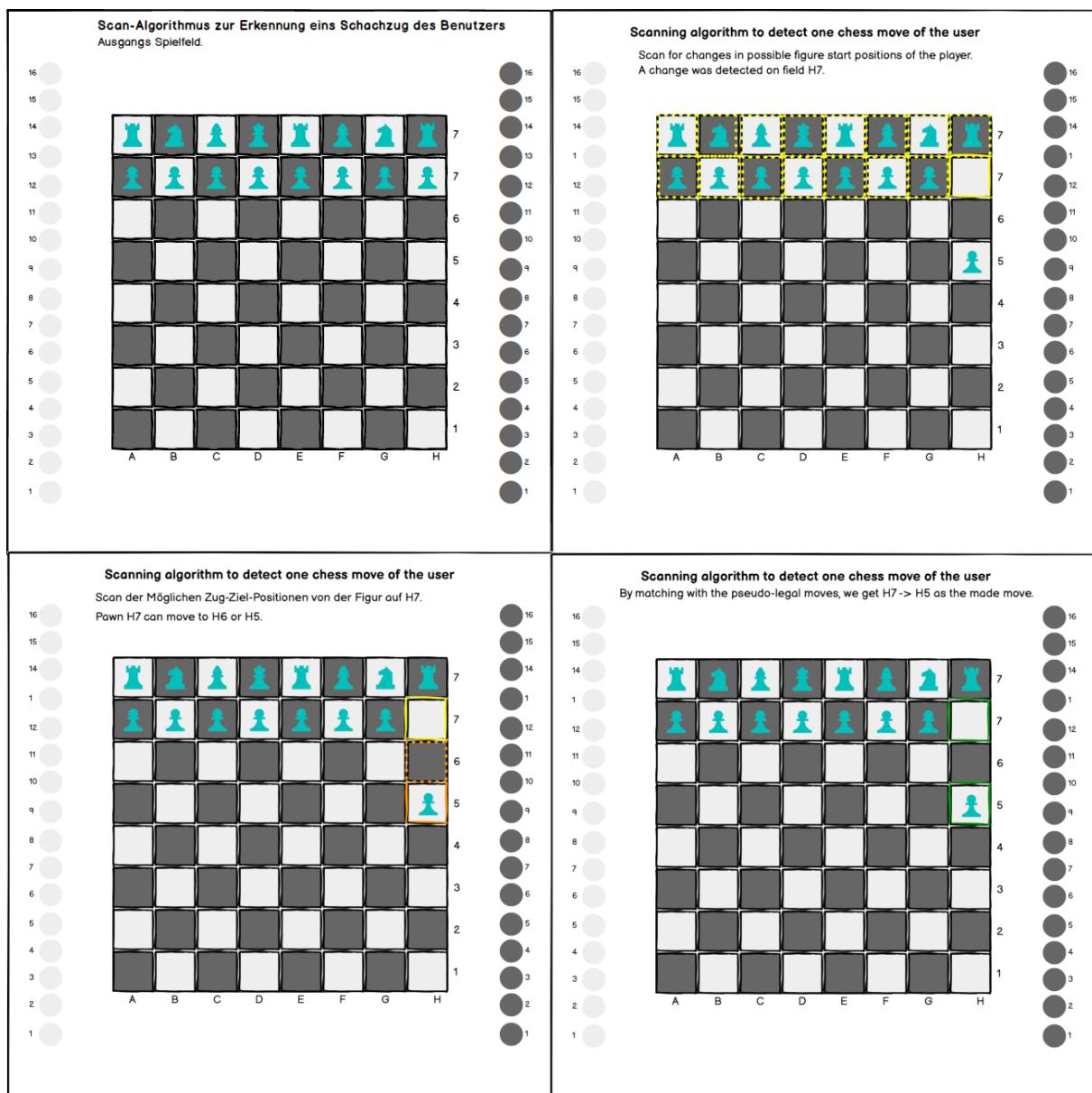


Bild 0-17: Embedded System Software: Schachfeld Scan Algorithmus Ablauf

0.9.2 Figur Bewegungspfadberechnung

- Algorithmus zur Umsetzung eines Schachzugs
- Auftrennung 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 Schachzug
- Scan der Schachfeld-Veränderungen, durch Vergleich des vorherigen Schachfelds und der möglichen Züge

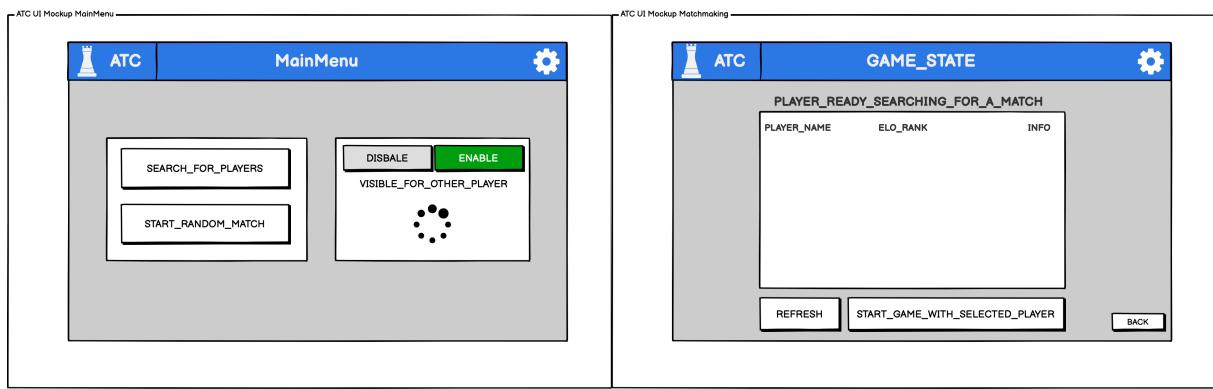


Bild 0-18: Embedded System Software: User-Interface Mockup

0.9.3 Userinterface

Das Userinterface ist mit das zentrale Element mit welchem der Benutzer interagiert. Hierbei soll dieses nur die nötigsten Funktionen bereitstellen, welche zur Bedienung des Schachtisches nötig sind.

- grosse schaltflächen
 - kleine Menutiefe max. ein untermenü
-
- QT Quick UI, als Package in Buildroot integriert
 - IPC Bibliothek zur Kommunikation mit der controller-Software Instanz
 - JSON basiert => einfaches Debugging
 - Steuerung über andere Endgeräte möglich z.B Handy-App welche im selben Netzwerk befindet.

0.10 Fazit

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

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

- lässt Spiel für Erweiterungen
-

0.10.1 Ausblick

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

Literaturverzeichnis

- [1] DGT: *DGT Bluetooth Wenge.* <https://www.topschach.de/bluetooth-wenge-eboard-figuren-p-3842.html>. Version: 28.03.2021
- [2] DGT: *DTG Smart Board.* <https://www.topschach.de/smart-board-p-3835.html>. Version: 28.03.2021
- [3] FIEKAS, Niklas: *Python-Chess Library.* <https://github.com/niklasf/python-chess>. Version: 28.03.2021
- [4] GUERERO, Michael: *Automated Chess Board.* <https://create.arduino.cc/projecthub/maguerero/automated-chess-board-50ca0f>. Version: 28.03.2021
- [5] INC., SquareOff: *Grand Kingdom Set.* <https://squareoffnow.com/product/gks>. Version: 28.03.2021
- [6] INC., SquareOff: *Kingdom Set.* <https://squareoffnow.com/product/kds>. Version: 28.03.2021
- [7] MACHINES, DIY: *DIY Super Smart Chessboard.* <https://www.instructables.com/DIY-Super-Smart-Chessboard-Play-Online-or-Against-/>. Version: 28.03.2021
- [8] McEvoy, Brian: *PRINT CHESS PIECES, THEN DEFEAT THE CHESS-PLAYING PRINTER.* <https://hackaday.com/2021/03/06/print-chess-pieces-then-defeat-the-chess-playing-printer/>. Version: 28.03.2021
- [9] McEvoy, Brian: *Ultimaker Cura Slicer.* <https://github.com/Ultimaker/Cura>. Version: 28.03.2021

- [10] RAVICHANDRAN, Akash: *Automated Chess Board*. <https://create.arduino.cc/projecthub/automaters/automated-chess-5dbd7a>. Version: 28.03.2021
- [11] SEMICONDUCTOR, Nordic: *NFC library and modules: Generating messages and records*. https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v12.0.0%2Fnfc_ndef_format_dox.html. Version: 28.03.2021
- [12] SEMICONDUCTOR, Nordic: *NFC library and modules: NDEF message and record format*. https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v12.0.0%2Fnfc_ndef_format_dox.html. Version: 28.03.2021
- [13] STEFAN-MEYER KAHLEN, ShredderChess: *UCI protocol*. <http://wbec-ridderkerk.nl/html/UCIProtocol.html>, <https://www.shredderchess.com/download/div/uci.zip>. Version: 28.03.2021
- [14] TORD ROMSTAD, Joona K. Marco Costalba C. Marco Costalba: *Stockfish - Strong Open Source Chess Engine*. <https://stockfishchess.org/>. Version: 28.03.2021
- [15] V., Deutsche I. e.: *DIN EN ISO 9241-220 Ergonomie der Mensch-System-Interaktion - Teil 220: Prozesse zur Ermöglichung, Durchführung und Bewertung menschzentrierter Gestaltung für interaktive Systeme in Hersteller- und Betreiberorganisationen*. <https://www.din.de/de/mitwirken/normenausschuesse/naerg-veroeffentlichungen/wdc-beuth:din21:289443385>. Version: 04.04.2021
- [16] YOUSIFNIMAT: *Chess Robot*. <https://www.instructables.com/Chess-Robot/>. Version: 28.03.2021
- [17] ZALM, Erik van d.: *Marlin Firmware*. <https://marlinfw.org/>, <https://github.com/MarlinFirmware/Marlin>. Version: 28.03.2021

Akronyme

AJAX Asynchronous JavaScript and XML. 32

API Application Programming Interface. 30, 31

ATC Atomic Chess Table. 7, 25, 26

CAD Computer-Aided Design. 10, 18

CSS Cascading Style Sheets. 31

FDM Fused Deposition Modeling. 10

FEN Forsyth-Edwards-Notation. 30, 31, 34

FTP File Transfer Protocol. 27

GPL General Public License. 4

GUI Graphical User Interface. 34

HAL Hardware Abstraction Layer. 21

HTML Hypertext Markup Language. 31, 32

HTTP Hypertext Transfer Protocol. 27, 31

HTTPS Hypertext Transfer Protocol Secure. 27

Akronyme

I2C Inter-Integrated Circuit. 23

JS JavaScript. 31, 32

NDEF NFC Data Exchange Format. 13

NDEF-RTD NDEF Record Type Definition. 13

NFC Near Field Communication. 11, 13, 18, 23, 24

REST Representational State Transfer. 27, 34

SPI Serial Peripheral Interface. 19

UCI Universal Chess Interface. 34

USB Universal Serial Bus. 4, 7, 20, 21, 23, 25

WLAN Wireless Local Area Network. 4, 7, 25

X-FEN Extended Forsyth-Edwards-Notation. 30, 31

Abbildungsverzeichnis

0-1	Grove PN532 NFC Reader mit Kabelgebundener Antenne	9
0-2	3D Druck: Objekt (rot,gelb,grün),Tree Structure (cyan)	12
0-3	Prototyp Hardware: Erster Prototyp des autonomen Schachtisch	12
0-4	Prototyp Hardware: Tool zur Erstellung des NDEF Payloads: ChessFigureIDGenerator.html	14
0-5	Prototyp Hardware: NDEF Text Record Payload für einen weissen Turm	15
0-6	Prototyp Hardware: Blockdiagramm	16
0-7	Production Hardware: Finaler autonomer Schachtisch	17
0-8	Production Hardware: Blockdiagramm	20
0-9	Gesamtübersicht der verwendeten Cloud-Infrastruktur	26
0-10	Cloud-Infrastruktur: Aufbau einer URI	27
0-11	Cloud-Infrastruktur: Aufbau der Service Architecture	28
0-12	Cloud-Infrastruktur: Backend Login-Request und Response	29
0-13	MoveValidator: Beispiel Request zur Ausführung eines Zuges auf einem gegebenen Schachbrett	30
0-14	Webclient: Spielansicht	32
0-15	Webclient: Statistiken	33
0-16	Embedded System Software: Ablaufdiagramm	36
0-17	Embedded System Software: Schachfeld Scan Algorithmus Ablauf	37
0-18	Embedded System Software: User-Interface Mockup	38

Tabellenverzeichnis

0.1 Auflistung kommerzieller autonomer Schachttische	2
0.2 Auflistung von Open-Source Schachttisch Projekten	4
0.3 Auflistung der Anforderungen an den autonomen Schachttisch	7
0.4 Verwendete 3D Druck Parameter. Temperatur nach Herstellerangaben des verwendeten PLA Filaments.	10
0.5 Standardhardware 3D Drucker Steuerungen	19
0.6 Grundlegende verwendete G-Code Kommandos	21
0.7 Hinterlegte G-Code Steuerungen	22
0.8 Eigenschaften des finalen Prototypen	25
0.9 MoveValidator-Service API Overview	30
0.10 Vergleich FEN - X-FEN	30

.1 Anhang