TEKO Schweizerische Fachschule

DIPLOMARBEIT

Entwicklung eines intelligenten Schachbretts mit KI-Integration und digitaler Spielanalyse

Diplomand	Robert Reitmajer
Studiengang	Elektrotechnik HF
Klasse	B-TEL-23-T-a
Art der Arbeit	Diplomarbeit
Betreuungsperson	Fabian Hirter
Abgabedatum	02.11.2026

November 2026

Management Summary

Diese Diplomarbeit behandelt die Weiterentwicklung eines bestehenden elektronischen Schachbretts zu einem intelligenten, vernetzten Schachsystem. Der ursprüngliche Prototyp erfasste die Belegung der 64 Felder über Reed-Schalter und Magnete in den Spielfiguren. Für einen zuverlässigen Spielbetrieb fehlten jedoch eine ausreichend belastbare Sensorverdrahtung, die vollständige Einbindung einer digitalen Benutzeroberfläche sowie eine Schachengine für Analyse und Computerspiel.

Ziel war es, die vorhandene Hardware gezielt zu verbessern und mit einer erweiterten Arduino-Steuerung, Node-RED und der Schachengine Stockfish zu einem funktionsfähigen Gesamtsystem zu verbinden. Dazu wurden die Sensorverbindungen überarbeitet, die Schachregeln einschliesslich der Sonderzüge umgesetzt und eine bidirektionale serielle Kommunikation eingerichtet. Das Node-RED-Dashboard zeigt die Brettstellung, den Zugverlauf und die Zugbewertungen an. Neben Partien zwischen zwei Personen ermöglicht das System Spiele gegen Stockfish. Die vom Computer berechneten Züge werden durch LEDs angezeigt, vom Benutzer auf dem physischen Brett ausgeführt und anschliessend überprüft.

Bei der Konzeptwahl wurde bewusst auf einen vollständigen Neubau sowie auf eine mechanische Bewegung der Figuren verzichtet. Stattdessen standen die Weiterverwendung bestehender Komponenten, die gezielte Beseitigung von Schwachstellen und eine nachvollziehbare Systemintegration im Vordergrund.

Die Überprüfung umfasste Hardwaretests, Softwaretests und vollständige Partien in beiden Spielmodi. Während eines zehntägigen Tests wurden 1'280 Sensorbetätigungen erfolgreich erkannt. Beim ersten mechanischen Belastungstest traten an elf Positionen Fehler auf; nach der Reparatur bestanden alle 64 Positionen den abschliessenden Belastungstest. Auch die geprüften Schachfunktionen, die Kommunikation und der Spieler-gegen-KI-Modus funktionierten in den durchgeführten Tests. Sämtliche sieben MUSS- und sechs SOLL-Anforderungen wurden erfüllt; die fünf KANN-Anforderungen wurden nicht umgesetzt.

Das Projekt zeigt, dass sich ein physisches Schachbrett mit vergleichsweise einfachen Hardwarekomponenten und etablierter Software um digitale Analyse- und KI-Funktionen erweitern lässt. Die Ergebnisse gelten für den geprüften Umfang; statistische Langzeittests und eine formelle Benutzerstudie wurden nicht durchgeführt. Für eine Weiterentwicklung werden insbesondere eine Leiterplatte für die Sensormatrix, eine robustere Erkennung des Brettzustands und ein vom externen PC unabhängiger Betrieb empfohlen.

Inhaltsverzeichnis

Management Summary **2**
Inhaltsverzeichnis **3**
Glossar und Abkürzungen **7**
1 Einleitung **9**
1.1 Ausgangslage 9
1.2 Problemstellung 9
1.3 Zielsetzung 10
1.4 Abgrenzung 11
1.5 Nutzen der Arbeit und fachlichen Bezug 11
2 Projektmanagement und Planung **12**
2.1 Projektorganisation 12
2.2 Project vorgehen 12
2.3 Project Phasen 13
2.4 Terminplanung 17
2.5 Meilensteine 18
2.6 Zeitaufwand 18
2.7 Projektkontrolle und wöchentlicher Fortschritt 19
2.8 Risikomanagement 19
3 Anforderungen und Pflichtenheft **21**
3.1 Zweck der Anforderungsanalyse 21
3.2 Herleitung der Anforderungen 21
3.3 Pflichtenheft 22
3.4 Zusätzliche technische Anforderungen 24
3.5 Abnahme- und Verifikationsstrategie 26
4 Konzeptentwicklung und Variantenentscheide **28**
4.1 Ziel der Konzeptphase 28
4.2 Bestehendes System als Grundlage 28
4.3 Gesamtsystemkonzept 29
4.4 Konzept zur Verbesserung der Sensorzuverlässigkeit 31
4.5 Konzept für die Kommunikation zwischen Arduino und PC 32
4.6 Konzept der digitalen Benutzeroberfläche 32
4.7 Auswahl der Schachengine 33
4.8 Konzept zur Ausführung der KI-Spielzüge 33
4.9 Konzept zur Synchronisation des Spielzustands 34
4.10 Bewertung der Konzeptvarianten 35

4.11 Gewähltes Gesamtkonzept 36
4.12 Zusammenfassung 36
5 Systemarchitektur 38
5.1 Hardwarearchitektur 38
5.2 Softwarearchitektur 38
5.3 Datenfluss und Schnittstellen 39
5.4 Kapitelzusammenfassung 39
6 Hardwareentwicklung und Zuverlässigkeit 40
6.1 Ausgangszustand des Sensorsystems 40
6.2 Ghosting und Funktion der Dioden 40
6.3 Mechanische Fehlerursachen und Verbesserungen 41
6.4 Überprüfung der Hardwareverbesserung 41
6.5 Kapitelzusammenfassung 42
7 Embedded-Software auf dem Arduino 44
7.1 Sensorabfrage und Zustandsmodell 44
7.2 Zuglogik und erlaubte Züge 45
7.3 Schach, Schachmatt und Sonderzüge 46
7.4 LED-Visualisierung 47
7.5 Kapitelzusammenfassung 48
8 Kommunikation zwischen Arduino und Node-RED 49
8.1 Serielles Kommunikationsprotokoll 49
8.2 Bidirektionale Kommunikation 50
8.3 Fehlerbehandlung 51
8.4 Kapitelzusammenfassung 52
9 Node-RED und digitale Benutzeroberfläche 53
9.1 Datenverarbeitung 53
9.2 Dashboard 54
9.3 Dokumentierte Entwicklungsprobleme 55
Kapitelzusammenfassung 56
10 Stockfish-Integration und Spielanalyse 57
10.1 UCI-Anbindung 57
10.2 Zugbewertung 58
10.3 Spieler-gegen-KI-Modus 59
10.4 KI-Schwierigkeitsgrad 59
Kapitelzusammenfassung 60
11 Verifikation und Validierung 61
11.1 Teststrategie 61

11.2 Durchgeführte Tests und Ergebnisse ... 62
11.3 Gesamtbewertung der Testergebnisse ... 66
12 Problemlösung und Entwicklungsverlauf ... 67
12.1 Mechanische Instabilität der Reed-Matrix ... 67
12.2 Serial-Parser in Node-RED ... 67
12.3 Move-Log-Darstellung ... 68
12.4 Synchronisation zwischen physischem und digitalem Schachbrett ... 68
12.5 PvAI-Turn-Filter ... 69
Kapitelzusammenfassung ... 69
13 Ergebnisse und Pflichtenhefterfüllung ... 71
13.1 Quantitative Ergebnisse ... 71
13.2 Erfüllung der Anforderungen ... 72
14 Diskussion ... 75
14.1 Technische Grenzen ... 75
14.2 Machbarkeit und Akzeptanz ... 76
15 Empfehlungen und Ausblick ... 77
15.1 Empfehlungen für technische Verbesserungen ... 77
15.2 Mögliche funktionale Erweiterungen ... 78
15.3 Empfohlene Entwicklungsprioritäten ... 79
15.4 Ausblick ... 79
16 Reflexion und gewonnene Erkenntnisse ... 80
16.1 Gewonnene technische Erkenntnisse ... 80
16.2 Projektplanung und Zeitmanagement ... 80
16.3 Fehlersuche und Problemlösung ... 81
16.4 Was ich anders machen würde ... 81
16.5 Persönliche Reflexion ... 81
Kapitelzusammenfassung ... 82
17 Schlusswort und Danksagung ... 83
17.1 Schlusswort ... 83
17.2 Danksagung ... 83
18 Literatur- und Quellenverzeichnis ... 84
19 Hilfsmittel und Tools ... 85
19.1 Einsatz von künstlicher Intelligenz ... 85
20 Eigenständigkeitserklärung ... 86
Anhang A – Testdokumentation ... 87
Anhang B – Schalt- und Verdrahtungsunterlagen ... 88
... 92

Anhang C – Software und Node-RED **93**

Anhang D – Entwicklungs- und Fehlerlog **95**

Glossar und Abkürzungen

Begriff / Abkürzung	Bedeutung
Arduino IDE	Entwicklungsumgebung zur Programmierung und Übertragung des Mikrocontroller-Codes.
Arduino Mega 2560	Mikrocontroller-Plattform zur Erfassung der Sensoren, Verarbeitung der Schachlogik und LED-Ansteuerung.
Baudrate	Übertragungsgeschwindigkeit der seriellen Verbindung; im Projekt 115200 Baud.
Bauernumwandlung	Umwandlung eines Bauern beim Erreichen der gegnerischen Grundreihe.
Centipawn	Bewertungseinheit einer Schachengine; 100 Centipawns entsprechen rechnerisch einem Bauernwert.
Dashboard	Digitale Bedien- und Anzeigeoberfläche mit Brettstellung, Zugliste und Auswertung.
Diode	Halbleiterbauteil zur Entkopplung der Matrixsignale und Vermeidung von Fehlauslösungen.
En passant	Schachsonderregel zum Schlagen eines gegnerischen Bauern unmittelbar nach dessen Doppelschritt.
FEN	Forsyth-Edwards Notation; Textformat zur Beschreibung einer Schachstellung.
Ghosting	Fehlerhafte Erkennung zusätzlicher Sensorpositionen bei der Abfrage einer Schaltmatrix.
KI	Künstliche Intelligenz; im Projekt als Bezeichnung für den computergesteuerten Schachgegner verwendet.
LED-Matrix	Anordnung einzeln ansteuerbarer LEDs zur visuellen Rückmeldung am Schachbrett.
MUSS-/SOLL-/KANN-Ziele	Prioritätsstufen der Projektanforderungen: zwingend, erwünscht und optional.
Node-RED	Flow-basierte Entwicklungsumgebung für Datenverarbeitung, Kommunikation und Benutzeroberfläche.
PvAI	Player versus Artificial Intelligence; Spielmodus gegen die Schachengine.
PvP	Player versus Player; Spielmodus für zwei menschliche Personen.
Reed-Schalter	Magnetisch betätigter Kontakt zur Erkennung, ob ein Schachfeld belegt ist.
Reed-Switch-Matrix	Anordnung von 64 Reed-Schaltern in acht Zeilen und acht Spalten zur Felderfassung.
RGB	Rot, Grün, Blau; Grundfarben für die additive Farbmischung der LEDs.
Rochade	Schachsonderzug, bei dem König und Turm unter festgelegten Bedingungen bewegt werden.
Serielle Kommunikation	Übertragung von Daten zwischen Arduino und Node-RED über eine USB-Verbindung.
Stockfish	Schachengine zur Berechnung von Zügen und Analyse von Schachstellungen.

UCI	Universal Chess Interface; standardisiertes Kommunikationsprotokoll für Schachengines.
USB	Universal Serial Bus; Schnittstelle für die serielle Verbindung zwischen Arduino und Computer.
WS2812B	Digital adressierbare RGB-LED zur Anzeige von Zugmöglichkeiten und Spielinformationen.
Zugvalidierung	Prüfung, ob ein erkannter Schachzug gemäss den geltenden Regeln zulässig ist.

1 Einleitung

1.1 Ausgangslage

Im Rahmen einer vorgängigen Projektarbeit wurde ein elektronisches Schachbrett entwickelt, welches die Position und Bewegung von Schachfiguren mithilfe von Reed-Schaltern und Magneten erfassen kann. Unter jedem der 64 Schachfelder befindet sich ein Reed-Schalter. Die in den Schachfiguren integrierten Magnete betätigen diese Sensoren und ermöglichen dadurch die Erkennung, ob sich eine Figur auf einem bestimmten Feld befindet.

Die Verarbeitung der Sensorsignale erfolgt mit einem Arduino Mega 2560. Für die Visualisierung von Informationen am physischen Schachbrett befinden sich seitlich am Brett zwei adressierbare WS2812B-LED-Matrizen, wobei jedem Spieler eine eigene LED-Matrix zugeordnet ist. Mithilfe dieser LEDs können beispielsweise mögliche Zielfelder einer ausgewählten Schachfigur sowie später die von der Schach-KI berechneten Spielzüge dargestellt werden.

Zusätzlich zum physischen Schachbrett wurde im Rahmen einer früheren Transferarbeit bereits eine grundlegende digitale Benutzeroberfläche mit Node-RED entwickelt. Diese enthielt erste Funktionen zur Darstellung von schachbezogenen Informationen, beispielsweise der aktuellen Brettposition, der ausgeführten Spielzüge sowie weiterer Spielinformationen. Diese Version wurde jedoch hauptsächlich als Softwarekonzept entwickelt und war noch nicht mit dem Arduino beziehungsweise dem physischen Schachbrett verbunden. Ein automatischer Austausch real ausgeführter Schachzüge zwischen dem physischen Brett und der Node-RED-Oberfläche war zu diesem Zeitpunkt somit noch nicht möglich.

Diese beiden vorgängigen Arbeiten bilden die Ausgangsbasis für die Diplomarbeit. Das physische Schachbrett stellt die bereits vorhandene Sensorik und LED-Hardware zur Verfügung, während die Transferarbeit eine erste Grundlage für die digitale Benutzeroberfläche liefert. Die beiden Systeme waren jedoch noch nicht zu einem gemeinsamen funktionsfähigen Gesamtsystem verbunden.

Darüber hinaus bestanden beim bisherigen physischen Schachbrett noch technische Einschränkungen. Insbesondere die Langzeitzuverlässigkeit der Reed-Schalter und der handgefertigten Verdrahtung stellte eine Schwachstelle dar. Auch die Spiellogik war noch eingeschränkt und eine Schachengine zur Analyse der Partie beziehungsweise als computergesteuerter Gegner war noch nicht integriert.

Die Diplomarbeit baut deshalb gezielt auf diesen bestehenden Grundlagen auf. Ziel ist nicht, das Schachbrett oder die digitale Benutzeroberfläche vollständig neu zu entwickeln, sondern die bestehenden Systeme zusammenzuführen, technisch zu verbessern und wesentlich zu erweitern. Dazu gehören insbesondere die Kommunikation zwischen Arduino und Node-RED, die Erweiterung der Schachlogik, die Verbesserung der Hardwarezuverlässigkeit, die Integration der Schachengine Stockfish sowie die bidirektionale Kommunikation zwischen dem physischen Schachbrett und dem Computersystem.

1.2 Problemstellung

Bei einem intelligenten Schachbrett reicht es nicht aus, lediglich zu erkennen, ob einzelne Felder belegt sind. Für einen zuverlässigen Spielbetrieb müssen Sensorik, Spiellogik, Kommunikation und Visualisierung als Gesamtsystem zusammenarbeiten.

Eine besondere Herausforderung stellt dabei die Synchronisation zwischen dem realen und dem digitalen Spielzustand dar. Wird eine Figurenbewegung nicht korrekt erkannt oder verarbeitet, kann der

intern gespeicherte Brettzustand vom tatsächlichen Zustand des physischen Schachbretts abweichen. Dadurch können nachfolgende Spielzüge nicht mehr korrekt verarbeitet werden.

Auch die Zuverlässigkeit der Hardware spielt dabei eine entscheidende Rolle. Die Reed-Schalter sowie deren Dioden und Verdrahtung wurden beim bestehenden Prototyp von Hand aufgebaut. Während der bisherigen Entwicklung zeigte sich, dass insbesondere mechanische Belastungen einzelner Verbindungen zu zeitweisen Ausfällen führen können.

Neben der Verbesserung dieser Zuverlässigkeit soll das System funktional erweitert werden. Zwei Personen sollen eine vollständige Partie direkt auf dem physischen Schachbrett spielen können. Zusätzlich soll eine Schach-KI integriert werden, gegen welche ebenfalls direkt auf dem Brett gespielt werden kann. Dazu muss eine bidirektionale Kommunikation entstehen: Das Schachbrett übermittelt den ausgeführten Zug an das Computersystem, während der von der KI berechnete Gegenzug wieder an das Schachbrett übertragen und dort mithilfe des LED-Systems dargestellt wird.

Parallel dazu soll der aktuelle Spielzustand digital visualisiert und die gespielte Partie analysiert werden.

Daraus ergibt sich folgende zentrale Fragestellung:

Wie kann das bestehende elektronische Schachbrett zu einem zuverlässigen intelligenten Schachsystem weiterentwickelt werden, welches reale Figurenbewegungen verarbeitet, vollständige Schachpartien ermöglicht, mit einer Schach-KI kommuniziert und den Spielverlauf digital visualisiert und analysiert?

1.3 Zielsetzung

Das Hauptziel der Diplomarbeit besteht darin, den bestehenden Prototyp zu einem zuverlässig funktionierenden intelligenten Schachbrett weiterzuentwickeln.

Die Zielsetzung wird entsprechend der genehmigten Themeneingabe in **MUSS-, SOLL- und KANN-Ziele** gegliedert.

Die MUSS-Ziele bilden die zwingend zu erreichenden Kernfunktionen des Projekts. Dazu gehören insbesondere die zuverlässige Erkennung von Figurenbewegungen auf allen 64 Feldern, die Verbesserung der Langzeitzuverlässigkeit der Reed-Schalter und deren Verdrahtung, die Umsetzung eines Spielmodus für zwei Spieler sowie die Integration einer Schach-KI. Weiterhin muss eine Kommunikation zwischen dem Schachbrett und der KI zur Verarbeitung von Spielzügen in Echtzeit realisiert werden. Die von der KI berechneten Gegenzüge sollen über das LED-System des Schachbretts dargestellt werden. Abschliessend müssen die entwickelten Funktionen systematisch getestet und die Ergebnisse dokumentiert werden.

Die SOLL-Ziele erweitern das System insbesondere auf der digitalen Ebene. Dazu gehören eine digitale Darstellung der aktuellen Brettposition, die Synchronisation zwischen dem physischen Schachbrett und der digitalen Darstellung sowie ein vollständiges Zugprotokoll. Zusätzlich sollen die ausgeführten Spielzüge mithilfe der Schach-KI bewertet und in verschiedene Qualitätskategorien eingeteilt werden. Weitere Analyseinformationen sollen auf der digitalen Benutzeroberfläche dargestellt werden.

Die KANN-Ziele beschreiben zusätzliche Erweiterungen, welche nur umgesetzt werden, wenn die MUSS- und SOLL-Ziele vollständig realisiert und ausreichend getestet wurden. Dazu gehören beispielsweise ein Robotersystem zur automatischen Bewegung der Schachfiguren, eine automatische Ausführung der von

der KI berechneten Gegenzüge, eine automatische Erkennung und Ausführung von Schachaufgaben, eine Sprachsteuerung beziehungsweise Sprachausgabe sowie ein Online-Spielmodus gegen andere Spieler.

Die detaillierten Anforderungen und die dazugehörigen Prüf- und Akzeptanzkriterien werden im Kapitel **Anforderungsanalyse und Pflichtenheft** definiert.

1.4 Abgrenzung

Die Diplomarbeit konzentriert sich auf die Weiterentwicklung des bestehenden Schachbretts und dessen Integration in ein digitales Gesamtsystem. Der mechanische Grundaufbau des Schachbretts wird grundsätzlich beibehalten und nur dort angepasst, wo dies zur Verbesserung der Zuverlässigkeit notwendig ist.

Der Schwerpunkt liegt auf der Sensorik, der Embedded-Software des Arduino Mega 2560, der LED-Visualisierung, der Kommunikation zwischen Schachbrett und PC, der Node-RED-basierten Benutzeroberfläche sowie der Integration der Schachengine Stockfish.

Eine vollautomatische mechanische Bewegung der Schachfiguren mittels eines Robotersystems gehört nicht zum zwingenden Umfang der Diplomarbeit. Auch Sprachfunktionen und ein Online-Spielmodus sind als KANN-Ziele definiert. Diese Erweiterungen werden gegenüber den MUSS- und SOLL-Zielen bewusst nachrangig behandelt.

Dadurch soll verhindert werden, dass zusätzliche Funktionen auf Kosten der Zuverlässigkeit, Verifikation oder Dokumentation der Kernfunktionen umgesetzt werden. Vorrang hat ein stabiles und nachvollziehbar getestetes Gesamtsystem, welches sämtliche definierten MUSS- und möglichst alle SOLL-Ziele erfüllt.

1.5 Nutzen der Arbeit und fachlichen Bezug

Die Diplomarbeit verbindet mehrere Themengebiete der Elektrotechnik und Informatik innerhalb eines gemeinsamen Systems. Dazu gehören Sensorik, digitale Ein- und Ausgangsverarbeitung, Mikrocontrollertechnik, LED-Ansteuerung, serielle Kommunikation, Softwareentwicklung, Benutzeroberflächen sowie die Integration einer Schachengine.

Ein wesentlicher Nutzen der Arbeit liegt in der Verbindung eines physischen Systems mit digitaler Datenverarbeitung. Ein auf dem realen Schachbrett ausgeführter Vorgang wird sensorisch erfasst, elektronisch verarbeitet, an ein Computersystem übertragen und dort weiterverarbeitet beziehungsweise analysiert. Gleichzeitig können Informationen aus dem Computersystem wieder an das physische System zurückgegeben und über das LED-System dargestellt werden.

Dadurch entsteht eine bidirektionale Kommunikation zwischen Hardware und Software. Besonders bei einem Spiel gegen die Schach-KI wird dieses Zusammenspiel deutlich: Der Spieler führt seinen Zug physisch aus, das System erkennt und überträgt diesen Zug, die Schachengine berechnet einen Gegenzug und dieser wird anschliessend wieder an das physische Schachbrett übermittelt und für den Spieler visualisiert.

Das Projekt stellt somit ein praktisches Beispiel für die Integration von **Sensorik, Embedded Systems, Kommunikationstechnik und PC-Software zu einem gemeinsamen intelligenten Gesamtsystem** dar.

2 Projektmanagement und Planung

2.1 Projektorganisation

Die Diplomarbeit wird als Einzelarbeit durchgeführt. Sämtliche Arbeiten in den Bereichen Planung, Hardwareoptimierung, Softwareentwicklung, Systemintegration, Testdurchführung, Auswertung und Dokumentation werden durch den Diplomanden selbstständig ausgeführt.

Als technische Grundlage dienen das bereits vorhandene elektronische Schachbrett sowie die im Rahmen einer früheren Transferarbeit entwickelte Node-RED-Benutzeroberfläche. Beide Teilsysteme werden im Rahmen der Diplomarbeit weiterentwickelt und zu einem gemeinsamen Gesamtsystem zusammengeführt. Die Betreuung der Diplomarbeit erfolgt gemäss den Vorgaben der TEKO. Die vorgesehenen Besprechungstermine dienen insbesondere dazu, den aktuellen Projektfortschritt, die Zielerreichung sowie offene technische und organisatorische Punkte zu besprechen.

Da keine weiteren Projektmitglieder beteiligt sind, ist die Projektorganisation bewusst einfach gehalten. Gleichzeitig liegt dadurch die vollständige Verantwortung für Terminplanung, technische Entscheidungen, Umsetzung, Tests und Dokumentation beim Diplomanden.

2.2 Project vorgehen

Für die Entwicklung wurde ein schrittweises und iteratives Vorgehen gewählt. Das Gesamtsystem besteht aus mehreren voneinander abhängigen Komponenten. Dazu gehören die Sensorik des physischen Schachbretts, die Arduino-Software, die serielle Kommunikation, die Node-RED-Benutzeroberfläche sowie die Schachengine Stockfish.

Eine gleichzeitige Entwicklung aller Komponenten ohne Zwischenprüfungen würde die Fehlersuche deutlich erschweren. Deshalb wurde das Gesamtsystem in einzelne Funktionsbereiche unterteilt:

1. Stabilisierung und Prüfung der bestehenden Hardware
2. Weiterentwicklung der Arduino-Software und Schachlogik
3. Kommunikation zwischen Arduino und Node-RED
4. Integration und Erweiterung der digitalen Benutzeroberfläche
5. Integration der Schachengine Stockfish
6. Umsetzung des Spielmodus gegen die KI
7. Integration zusätzlicher Schachregeln und Statusfunktionen
8. Systematische Verifikation des Gesamtsystems
9. Auswertung und Dokumentation

Nach jeder grösseren Erweiterung wurde überprüft, ob die bereits bestehenden Funktionen weiterhin korrekt arbeiten. Dadurch soll verhindert werden, dass neue Funktionen bereits funktionierende Systemteile unbeabsichtigt beeinträchtigen.

Dieses Vorgehen erwies sich insbesondere bei der Integration von Stockfish und bei der Umsetzung des Spielmodus gegen die KI als sinnvoll. Die einzelnen Kommunikationsschritte konnten separat überprüft werden, bevor der vollständige Ablauf als Gesamtsystem getestet wurde.

Der vollständige Informationsfluss im Spielmodus gegen die KI kann vereinfacht folgendermassen dargestellt werden:

Physischer Spielzug → Reed-Schalter → Arduino → serielle Kommunikation → Node-RED → Stockfish → Node-RED → Arduino → LED-Anzeige → physische Ausführung des KI-Zuges

Erst nachdem die einzelnen Schnittstellen funktionsfähig waren, wurde dieser vollständige Ablauf als zusammenhängendes System getestet.

2.3 Project Phasen

2.3.1 Phase 1 – Analyse des Ausgangszustands

Zu Beginn der Diplomarbeit wurde der vorhandene Entwicklungsstand des Schachbretts überprüft. Dabei standen insbesondere die Funktion der 64 Reed-Schalter sowie die mechanische Stabilität der bestehenden Verdrahtung im Vordergrund.

Bei der Funktionsprüfung zeigte sich, dass verschiedene Verbindungen der Reed-Switch-Matrix nicht mehr zuverlässig arbeiteten. Da eine zuverlässige Sensorik die Grundlage für sämtliche weiteren Funktionen des Systems bildet, wurde die Behebung dieser Probleme zunächst höher priorisiert als die Entwicklung zusätzlicher Softwarefunktionen.

Gleichzeitig wurde der vorhandene Stand der Node-RED-Benutzeroberfläche aus der früheren Transferarbeit analysiert. Die grundlegenden Elemente der digitalen Oberfläche waren bereits vorhanden, jedoch bestand zu diesem Zeitpunkt noch keine Verbindung zum realen Arduino beziehungsweise zum physischen Schachbrett.

2.3.2 Phase 2 – Stabilisierung der Sensorik

Die fehlerhaften Verbindungen der Reed-Switch-Matrix wurden systematisch überprüft und nachgelötet. Als wesentliche mechanische Schwachstelle wurden bewegliche Dioden beziehungsweise die dadurch belasteten Lötstellen erkannt.

Nach der elektrischen Instandsetzung wurden die Dioden zusätzlich mechanisch mit Heisskleber fixiert. Dadurch werden Bewegungen der Bauteile reduziert und die Lötstellen mechanisch entlastet.

Nach Abschluss dieser Arbeiten funktionierten wieder alle 64 Sensorpositionen. Die Sensorik wurde anschliessend nicht nur einmalig geprüft, sondern über einen längeren Zeitraum regelmässig kontrolliert. Dadurch soll neben der momentanen Funktion insbesondere die Langzeitstabilität der vorgenommenen Verbesserungen beurteilt werden können.

Die langfristige Überprüfung der Sensorik wird während der weiteren Projektlaufzeit fortgesetzt und später durch definierte quantitative Tests ergänzt.

2.3.3 Phase 3 – Erweiterung der Arduino-Software und Schachlogik

Nachdem die Sensorik wieder vollständig funktionsfähig war, wurde die bestehende Arduino-Software schrittweise weiterentwickelt. Die ursprüngliche Software war hauptsächlich auf die Erkennung und Verarbeitung grundlegender Figurenbewegungen ausgelegt. Im Rahmen der Diplomarbeit wurde diese Grundlage zu einer umfassenderen Schachlogik erweitert.

Zunächst wurde die Verarbeitung beider Spieler umgesetzt. Das System erkennt, welcher Spieler am Zug ist, welche Figur aufgenommen wurde und auf welchem Feld diese anschliessend platziert wird. Auf Basis des intern gespeicherten Brettzustands werden mögliche Spielzüge berechnet. Legale Züge werden akzeptiert und der Spielzustand entsprechend aktualisiert, während unzulässige Züge abgewiesen und über das LED-System signalisiert werden.

Im weiteren Entwicklungsverlauf wurde die Zugvalidierung um die **Königssicherheit** erweitert. Dadurch wird nicht nur geprüft, ob eine Figur grundsätzlich entsprechend ihren Bewegungsregeln gezogen werden darf, sondern auch, ob der eigene König nach dem Zug weiterhin geschützt ist. Ein Zug, durch den der eigene König im Schach stehen würde, wird vom System nicht zugelassen. Dadurch werden beispielsweise auch Bewegungen gefesselter Figuren korrekt eingeschränkt.

Zusätzlich wurden die Sonderzüge **Rochade** und **En passant** implementiert. Für beide Funktionen müssen neben der normalen Figurenbewegung zusätzliche Bedingungen sowie vorherige Spielzustände berücksichtigt werden. Die implementierten Funktionen wurden anschliessend gezielt auf dem physischen Schachbrett getestet und funktionierten in den geprüften Spielsituationen korrekt.

Die Spiellogik wurde ausserdem um die Erkennung der Spielzustände **Schach, Schachmatt und Patt** erweitert. Nach einem ausgeführten Zug wird überprüft, ob ein König angegriffen wird und ob dem betroffenen Spieler noch ein legaler Zug zur Verfügung steht. Auf dieser Grundlage kann zwischen einer normalen Schachsituation, Schachmatt und Patt unterschieden werden.

Zur Funktionsprüfung wurden gezielt definierte Spielsituationen verwendet. Die Schachmatt-Erkennung wurde beispielsweise anhand einer festgelegten Zugfolge überprüft. Zusätzlich wurden Situationen getestet, in denen eine Figur den eigenen König schützt und deshalb nicht bewegt werden darf. Auch Rochade und En passant wurden praktisch auf dem physischen Schachbrett durchgeführt und erfolgreich erkannt.

Damit wurde die ursprünglich eingeschränkte Spiellogik zu einem System erweitert, das die für das Projekt relevanten regulären Spielabläufe und Sonderfälle selbstständig verarbeiten kann.

2.3.4 Phase 4 – Integration von Arduino und Node-RED

Im nächsten Schritt wurde die bereits aus der Transferarbeit vorhandene Node-RED-Benutzeroberfläche mit dem realen Arduino verbunden.

Dazu wurde eine serielle Kommunikation eingerichtet, über welche erkannte Spielzüge vom Arduino an Node-RED übertragen werden. Anschliessend wurden die bestehenden Dashboardfunktionen schrittweise mit den realen Daten des physischen Schachbretts verbunden.

Zu den integrierten Funktionen gehören unter anderem:

- digitale Darstellung des aktuellen Schachbretts,
- Anzeige des aktuellen Spielers,
- Zugzähler,
- Anzeige des zuletzt ausgeführten Zuges,

- vollständiges Zugprotokoll,
- Reset-Funktion,
- Anzeige des aktuellen Spielstatus.

Während der Integration traten verschiedene Softwarefehler auf. Diese wurden mithilfe von Debug-Ausgaben und durch die getrennte Prüfung einzelner Verarbeitungsschritte lokalisiert.

Ein Beispiel dafür war die Verarbeitung der vom Arduino übertragenen Spielzüge. Die Daten wurden über die serielle Schnittstelle korrekt übertragen, jedoch zunächst innerhalb von Node-RED fehlerhaft verarbeitet. Durch die getrennte Kontrolle der einzelnen Verarbeitungsschritte konnte die Ursache auf die entsprechende Softwarefunktion eingegrenzt und korrigiert werden.

Ein weiterer Fehler trat bei der Darstellung des Zugprotokolls auf. Die Datenverarbeitung innerhalb von Node-RED funktionierte korrekt, während die ursprüngliche Darstellung im Dashboard nicht alle Einträge wie vorgesehen anzeigte. Durch die getrennte Betrachtung von Datenverarbeitung und Visualisierung konnte festgestellt werden, dass die Ursache in der Darstellung und nicht in der eigentlichen Verarbeitung der Züge lag.

Diese Fehlerfälle werden in einem späteren Kapitel detaillierter analysiert und dokumentiert.

2.3.5 Phase 5 – Integration der Schachengine Stockfish

Nach erfolgreicher Verbindung des physischen und digitalen Systems wurde die Schachengine Stockfish in Node-RED integriert.

Zunächst wurde überprüft, ob Stockfish über das UCI-Protokoll korrekt angesprochen werden kann. Anschliessend wurde die aktuelle Spielstellung beziehungsweise die bisherige Zugfolge aus Node-RED an die Schachengine übergeben.

Stockfish kann dadurch die aktuelle Stellung analysieren und eine Bewertung sowie einen geeigneten nächsten Spielzug berechnen.

Auf Grundlage der Engine-Bewertung wurde zusätzlich eine eigene Klassifikation der Zugqualität entwickelt. Dabei wird die Bewertung einer Stellung vor und nach einem ausgeführten Spielzug miteinander verglichen. Aus der Veränderung der Bewertung wird abgeleitet, wie stark sich der ausgeführte Zug auf die Stellung des jeweiligen Spielers ausgewirkt hat.

Anschliessend wird der Spielzug einer definierten Qualitätskategorie zugeordnet. Diese Kategorien dienen der verständlichen Darstellung der Analyse auf der Node-RED-Benutzeroberfläche. Die verwendeten Grenzwerte stellen eine projektspezifische Klassifikation dar und sind keine offiziellen Stockfish-Kategorien.

Zusätzlich wurde eine Auswahl verschiedener Spielstärken für den KI-Gegner in die Benutzeroberfläche integriert.

2.3.6 Phase 6 – Umsetzung des PvAI-Spielmodus

Nach erfolgreicher Integration von Stockfish wurde der vollständige Spielmodus **Player versus AI (PvAI)** umgesetzt.

Der menschliche Spieler führt seinen Spielzug direkt auf dem physischen Schachbrett aus. Die Bewegung wird über die Reed-Schalter erkannt und vom Arduino verarbeitet. Anschliessend wird der Zug über die serielle Schnittstelle an Node-RED und von dort an Stockfish weitergegeben.

Stockfish analysiert die neue Stellung und berechnet einen Gegenzug für die schwarzen Figuren. Dieser Zug wird nicht automatisch im digitalen Brett ausgeführt. Stattdessen übermittelt Node-RED den berechneten Zug zurück an den Arduino.

Der Arduino speichert den erwarteten KI-Zug und zeigt über das LED-System an, welche Figur bewegt und auf welches Zielfeld sie gestellt werden soll. Der Benutzer führt den berechneten Zug anschliessend physisch auf dem Schachbrett aus.

Während dieser Ausführung kontrolliert der Arduino, ob tatsächlich die erwartete Figur bewegt und auf dem richtigen Zielfeld platziert wird. Eine falsche Bewegung wird nicht als KI-Zug akzeptiert. Erst nach der korrekten physischen Ausführung wird der Spielzustand aktualisiert und der menschliche Spieler ist wieder am Zug.

Damit wurde erstmals ein vollständiger bidirektionaler Ablauf zwischen dem physischen Schachbrett und der Schach-KI realisiert:

Spieler → physisches Brett → Arduino → Node-RED → Stockfish → Node-RED → Arduino → LED-Anzeige → physisches Brett

Der vollständige Ablauf wurde praktisch getestet und funktionierte in den durchgeführten Tests korrekt.

2.3.7 Phase 7 – Verifikation und Optimierung

Nach der Umsetzung der wesentlichen Systemfunktionen wird der Schwerpunkt zunehmend von der Entwicklung auf die systematische Verifikation verlagert.

Dabei sollen nicht ausschliesslich vollständige Schachpartien gespielt werden. Zusätzlich werden definierte Einzeltests durchgeführt, damit die Zuverlässigkeit der verschiedenen Teilsysteme quantitativ beurteilt werden kann.

Dazu gehören insbesondere:

- wiederholte Prüfung aller 64 Sensorfelder,
- Langzeitprüfung der Reed-Switch-Matrix,
- Prüfung legaler Spielzüge,
- Prüfung illegaler Spielzüge,
- Prüfung der Königssicherheit,
- Prüfung von Rochade und En passant,

- Prüfung von Schach, Schachmatt und Patt,
- Prüfung der seriellen Kommunikation,
- Prüfung der Synchronisation zwischen physischem und digitalem Brett,
- Funktionsprüfung der zeitnahen Verarbeitung und Rückmeldung von Spielzügen im normalen Spielbetrieb,
- Prüfung der Stockfish-Integration,
- wiederholte Prüfung von KI-Spielzügen,
- Prüfung fehlerhafter physischer Ausführungen eines KI-Zuges,
- vollständige Schachpartien,
- längere Belastungs- und Stabilitätstests.

Die Ergebnisse dieser Tests werden dokumentiert und später mit den definierten Anforderungen des Pflichtenhefts verglichen.

2.4 Terminplanung

Für die Durchführung der Diplomarbeit wird ein Soll-Terminplan verwendet, welcher die wesentlichen Entwicklungs-, Test- und Dokumentationsphasen enthält.

Neben dem geplanten Projektverlauf wird auch der tatsächliche Verlauf dokumentiert. Dadurch kann am Ende der Diplomarbeit ein **Soll-Ist-Vergleich** durchgeführt werden. Abweichungen vom ursprünglichen Zeitplan werden dabei nicht nur dargestellt, sondern auch begründet.

Insbesondere bei einem Entwicklungsprojekt können technische Probleme dazu führen, dass einzelne Arbeiten mehr Zeit benötigen als ursprünglich geplant. Im vorliegenden Projekt wurde beispielsweise die Stabilisierung der Sensorik zu Beginn höher priorisiert, da eine zuverlässige Hardware die Voraussetzung für sämtliche nachfolgenden Software- und Integrationstests darstellt.

Für die Darstellung der Terminplanung wird zusätzlich ein grafischer Terminplan beziehungsweise ein Gantt-Diagramm verwendet.

Tabelle 2-1: Übersicht der Projektphasen und Terminplanung

Phase	Arbeitspaket	Zeitraum	Status
1	Ausgangsanalyse und Bestandsaufnahme	Projektbeginn	Abgeschlossen
2	Stabilisierung der Sensorik und Verdrahtung	Frühe Projektphase	Abgeschlossen
3	Erweiterung Arduino-Software und Schachlogik	Entwicklungsphase	Abgeschlossen
4	Arduino-Node-RED-Integration	Entwicklungsphase	Abgeschlossen
5	Stockfish-Integration und Zugbewertung	Entwicklungsphase	Abgeschlossen
6	PvAI-End-to-End-Integration	Entwicklungsphase	Abgeschlossen
7	Formale Verifikation und Optimierung	Abschlussphase	Abgeschlossen
8	Auswertung und Fertigstellung der Dokumentation	Abschlussphase	Abgeschlossen

2.5 Meilensteine

Zur Kontrolle des Projektfortschritts wurden definierte Meilensteine festgelegt.

M1 – Sensorik vollständig funktionsfähig

Alle 64 Sensorfelder können erkannt werden. Die festgestellten mechanischen Schwachstellen der Verdrahtung wurden verbessert und die Sensorik wird anschliessend regelmässig auf ihre Langzeitstabilität geprüft.

M2 – Zwei-Spieler-Modus funktionsfähig

Zwei Spieler können Spielzüge direkt auf dem physischen Schachbrett ausführen. Das System erkennt die Bewegungen beider Spieler, validiert die Spielzüge und aktualisiert den internen Spielzustand.

M3 – Physisches und digitales System synchronisiert

Die auf dem physischen Schachbrett ausgeführten Spielzüge werden korrekt an Node-RED übertragen. Die digitale Brettdarstellung sowie die weiteren Spielinformationen entsprechen dem physischen Spielzustand.

M4 – Stockfish erfolgreich integriert

Stockfish kann aus Node-RED angesprochen werden, die aktuelle Spielstellung analysieren und einen geeigneten Spielzug berechnen. Zusätzlich können ausgeführte Spielzüge anhand der Engine-Bewertung analysiert werden.

M5 – PvAI-End-to-End-Funktion erreicht

Ein menschlicher Spielzug wird physisch erkannt und an Stockfish übertragen. Der berechnete KI-Gegenzug wird anschliessend an den Arduino zurückgesendet, über das LED-System dargestellt und nach der korrekten physischen Ausführung akzeptiert.

M6 – Systemfunktionen verifiziert

Die definierten Testfälle für Sensorik, Schachlogik, Kommunikation, Synchronisation, Stockfish und den PvAI-Spielmodus wurden durchgeführt und dokumentiert.

M7 – Diplomarbeit abgeschlossen

Die Testergebnisse wurden ausgewertet, die Zielerreichung beurteilt und sämtliche erforderlichen Kapitel, Anhänge und Abschlussarbeiten fertiggestellt.

2.6 Zeitaufwand

Neben der Terminplanung wird auch der tatsächliche Arbeitsaufwand dokumentiert. Dadurch kann am Ende der Diplomarbeit beurteilt werden, in welchen Projektbereichen der grösste Aufwand entstanden ist und wie stark der tatsächliche Aufwand von der ursprünglichen Planung abweicht.

Der Zeitaufwand wird nach Tätigkeitsbereichen gegliedert:

Tabelle 2-2: Erfassung des Zeitaufwands nach Tätigkeitsbereichen

Tätigkeitsbereich	Zeitaufwand [h]
Projektplanung und Analyse	20
Hardware / Sensorik	30
Arduino-Software / Schachlogik	50
Node-RED / Benutzeroberfläche	50
Stockfish-Integration / PvAI	20
Tests und Verifikation	20
Dokumentation	50
Gesamt	240

2.7 Projektkontrolle und wöchentlicher Fortschritt

Der Projektfortschritt wird während der Bearbeitungszeit regelmässig dokumentiert. Dabei werden die erledigten Arbeiten, geplante aber noch nicht abgeschlossene Aufgaben, Abweichungen vom Zeitplan sowie aufgetretene technische Schwierigkeiten festgehalten.

Die regelmässige Projektkontrolle dient dazu, Probleme frühzeitig zu erkennen und die Prioritäten des Projekts bei Bedarf anzupassen.

Ein Beispiel dafür war die Hardwarezuverlässigkeit zu Beginn der Diplomarbeit. Obwohl weitere Softwareentwicklungen vorgesehen waren, wurde zunächst die Sensorik instand gesetzt und mechanisch stabilisiert. Ohne diese Massnahme wären spätere Systemtests nur eingeschränkt aussagekräftig gewesen.

Auch im weiteren Projektverlauf werden die MUSS- und SOLL-Ziele gegenüber optionalen KANN-Zielen priorisiert. Zusätzliche Funktionen werden nur umgesetzt, wenn dadurch die Verifikation, Dokumentation und zuverlässige Fertigstellung der Kernfunktionen nicht gefährdet werden.

Die wöchentlichen Statusberichte dienen gleichzeitig als Grundlage für den späteren Soll-Ist-Vergleich und die Nachvollziehbarkeit des Projektverlaufs.

2.8 Risikomanagement

Bereits während der Planung wurden technische und organisatorische Risiken betrachtet, welche den erfolgreichen Abschluss der Diplomarbeit beeinträchtigen könnten.

Durch die frühzeitige Identifikation dieser Risiken können geeignete Gegenmassnahmen definiert werden.

Tabelle 2-3: Risikobeurteilung und Gegenmassnahmen

Risiko	Eintritt	Auswirkung	Gegenmassnahme
Ausfall einzelner Reed-Schalter oder Lötstellen	Mittel	Hoch	Mechanische Stabilisierung, regelmässige Sensorprüfung und Nachlöten bei Bedarf
Desynchronisation zwischen physischem und digitalem Brett	Mittel	Hoch	Physische Zugbestätigung, definierte Zustandsübergänge und vollständige Testpartien
Fehler in der seriellen Kommunikation	Mittel	Mittel	Definierte Nachrichtenformate, Debug-Ausgaben und separate Schnittstellentests

Risiko	Eintritt	Auswirkung	Gegenmassnahme
Fehlerhafte Verarbeitung von Stockfish-Daten	Mittel	Mittel	UCI-Kommunikation schrittweise prüfen und definierte Referenzstellungen verwenden
Zeitverlust durch technische Fehlersuche	Mittel	Hoch	MUSS-Ziele priorisieren, Fehler systematisch eingrenzen und KANN-Ziele zurückstellen
Daten- oder Programmverlust	Niedrig	Hoch	Regelmässige Sicherung von Arduino-Code, Node-RED-Flow, Messdaten und Dokumentation

3 Anforderungen und Pflichtenheft

3.1 Zweck der Anforderungsanalyse

Bevor das bestehende Schachbrett weiterentwickelt wurde, mussten die Ziele der Diplomarbeit in klar definierte technische Anforderungen überführt werden. Die genehmigte Themeneingabe unterscheidet dabei zwischen **MUSS-, SOLL- und KANN-Zielen**.

Diese Einteilung wird für die gesamte Diplomarbeit beibehalten. Die allgemein formulierten Projektziele werden jedoch in einzelne technische Anforderungen unterteilt und konkretisiert, damit diese später objektiv geprüft und bewertet werden können.

Die drei Kategorien werden wie folgt definiert:

MUSS-Anforderungen sind für den erfolgreichen Abschluss des Projekts zwingend erforderlich. Sie bilden die Kernfunktionen des Systems und müssen umgesetzt sowie verifiziert werden.

SOLL-Anforderungen erweitern die Funktionalität und Benutzerfreundlichkeit des Systems. Sie sollen umgesetzt werden, sobald die grundlegenden Kernfunktionen zuverlässig funktionieren.

KANN-Anforderungen stellen optionale Erweiterungen des Systems dar. Sie werden nur berücksichtigt, wenn die MUSS- und SOLL-Anforderungen ausreichend umgesetzt, getestet und dokumentiert wurden.

Soweit möglich, wird jeder technischen Anforderung ein **messbares Akzeptanzkriterium** sowie eine entsprechende **Prüfmethode** zugeordnet. Dadurch entsteht eine direkte Verbindung zwischen den zu Beginn des Projekts definierten Anforderungen und den später in der Diplomarbeit dokumentierten Testergebnissen.

Auf diese Weise kann am Ende der Arbeit nachvollziehbar beurteilt werden, welche Anforderungen vollständig, teilweise oder nicht erfüllt wurden.

3.2 Herleitung der Anforderungen

Die Anforderungen an das intelligente Schachbrett basieren in erster Linie auf den in der genehmigten Themeneingabe definierten Projektzielen. Diese beschreiben die gewünschten Funktionen des Systems auf übergeordneter Ebene. Für die technische Umsetzung und die spätere Beurteilung der Zielerreichung müssen diese Ziele jedoch weiter konkretisiert und mit überprüfbaren Kriterien ergänzt werden.

Als Ausgangsbasis der Diplomarbeit dienen zwei bereits bestehende Teilsysteme. Einerseits ist das physische elektronische Schachbrett aus der vorgängigen Projektarbeit vorhanden. Dieses verfügt über eine Sensorik aus 64 Reed-Schaltern, einen Arduino Mega 2560 sowie zwei seitlich am Schachbrett angeordnete LED-Matrizen. Andererseits besteht aus einer früheren Transferarbeit bereits eine grundlegende Node-RED-Benutzeroberfläche. Zu Beginn der Diplomarbeit waren diese beiden Systeme jedoch noch nicht miteinander verbunden.

Aus dieser Ausgangslage ergeben sich Anforderungen aus mehreren technischen Bereichen. Neben der zuverlässigen Funktion der physischen Sensorik müssen die Spiellogik, die Kommunikation zwischen Arduino und PC, die Synchronisation zwischen physischem und digitalem Brett sowie die Integration der Schachengine Stockfish berücksichtigt werden. Zusätzlich müssen die berechneten KI-Züge wieder an das physische System übertragen und für den Spieler eindeutig dargestellt werden.

Die Anforderungen lassen sich deshalb in folgende technische Bereiche unterteilen:

- Zuverlässigkeit der Sensorik und Verdrahtung
- Erkennung und Verarbeitung physischer Figurenbewegungen
- Umsetzung der erforderlichen Schachlogik
- Kommunikation zwischen Arduino und Node-RED
- Synchronisation zwischen physischem und digitalem Schachbrett
- Integration der Schachengine Stockfish
- Rückübertragung und Darstellung der KI-Spielzüge
- digitale Darstellung und Analyse des Spielverlaufs
- systematische Prüfung und Dokumentation des Gesamtsystems

Bei der Beurteilung wird zwischen einer **implementierten Funktion** und einer **verifizierten Funktion** unterschieden. Eine Funktion gilt nicht bereits deshalb als abschliessend nachgewiesen, weil sie während der Entwicklung mehrmals korrekt funktioniert hat. Für die formale Verifikation werden definierte Testverfahren verwendet und die entsprechenden Ergebnisse dokumentiert.

Diese Unterscheidung ist insbesondere bei der Langzeitzuverlässigkeit der Reed-Switch-Matrix, der Kommunikation zwischen den einzelnen Systemkomponenten und der Synchronisation zwischen physischem und digitalem Schachbrett von Bedeutung.

3.3 Pflichtenheft

Das Pflichtenheft bildet die zentrale Grundlage für die technische Umsetzung und die spätere Bewertung der Diplomarbeit. Die Anforderungen werden entsprechend der genehmigten Themeneingabe in **MUSS-, SOLL- und KANN-Anforderungen** gegliedert.

Neben der eigentlichen Anforderung enthält das Pflichtenheft für jede relevante Funktion ein Akzeptanzkriterium sowie eine vorgesehene Verifikationsmethode. Dadurch wird bereits vor der abschliessenden Testphase definiert, anhand welcher Kriterien die Erfüllung einer Anforderung beurteilt werden soll.

Die Ergebnisse der einzelnen Prüfungen werden nicht bereits an dieser Stelle eingetragen, sondern im späteren Verifikations- und Auswertungskapitel dokumentiert. Dadurch bleibt eine klare Trennung zwischen **Anforderung**, **Prüfmethode** und **tatsächlichem Ergebnis** bestehen.

Tabelle 3-1: Priorität – Anforderung – Akzeptanzkriterium - Verifikation

ID	Priorität	Anforderung	Akzeptanzkriterium	Verifikation
MUSS-01	MUSS	Figurenbewegungen auf allen 64 Feldern zuverlässig erkennen.	64 Felder funktionsfähig; kontrollierter Wiederholungstest wird quantitativ ausgewertet.	Sensor-Repetitionstest

MUSS-02	MUSS	Langzeitzuverlässigkeit der Reed-Schalter und Verdrahtung erhöhen.	Technische Ursache analysiert, Massnahme umgesetzt und Langzeittest dokumentiert.	Vorher/Nachher + tägliche Tests
MUSS-03	MUSS	Spielmodus für zwei Spieler auf dem physischen Brett.	Mehrere vollständige Partien ohne nicht behebbaren Synchronisationsfehler.	Komplette PvP-Partien
MUSS-04	MUSS	Schach-KI integrieren, gegen die direkt am Brett gespielt werden kann.	End-to-End PvAI-Spielablauf funktioniert.	PvAI-Testfälle
MUSS-05	MUSS	Kommunikation zwischen Schachbrett und KI in Echtzeit.	Züge werden bidirektional und im normalen Spielbetrieb ohne wahrnehmbare Verzögerung verarbeitet; keine quantitative Latenzmessung vorgesehen.	Kommunikations- und Funktionstest in vollständigen Partien
MUSS-06	MUSS	KI-Gegenzüge über LED-System darstellen.	Von Stockfish berechneter Zug wird eindeutig am Brett angezeigt.	AI-Move-LED-Test
MUSS-07	MUSS	Tests durchführen und Ergebnisse dokumentieren.	Testplan, Rohdaten, Auswertung und Zielerreichung vorhanden.	Verifikationskapitel
SOLL-01	SOLL	Digitale Schachoberfläche für aktuelle Brettposition.	Digitale Brettposition entspricht dem physischen Zustand.	Synchronisationstest
SOLL-02	SOLL	Physisches und digitales Brett synchronisieren.	Definierte Testpartien ohne dauerhafte Abweichung.	Komplette Partien
SOLL-03	SOLL	Vollständiges Zugprotokoll anzeigen.	Alle ausgeführten Züge erscheinen in korrekter Reihenfolge.	Move-Log-Test
SOLL-04	SOLL	Jeden Spielzug mit Schach-KI bewerten.	Nach jedem Zug wird eine Engine-basierte Bewertung erzeugt.	Stockfish-Test
SOLL-05	SOLL	Spielzüge in Qualitätskategorien einteilen.	Eigene, dokumentierte Centipawn-Loss-Grenzen erzeugen reproduzierbare Kategorien.	Referenzpositionen
SOLL-06	SOLL	Zusätzliche Analyseinformationen am Bildschirm darstellen.	Dashboard zeigt definierte Spiel-/Analyseinformationen.	UI-Funktionstest
KANN-01	KANN	Robotersystem/Roboterarm.	Nicht geplant.	-
KANN-02	KANN	Automatische physische Ausführung der KI-Züge.	Nicht geplant.	-
KANN-03	KANN	Automatische physische Ausführung von Schlagzügen.	Nicht geplant	-
KANN-04	KANN	Sprachsteuerung/Sprachausgabe.	Nicht geplant.	-
KANN-05	KANN	Online-Spiel gegen andere Spieler.	Nicht geplant	-

Die MUSS-Anforderungen besitzen während der gesamten Entwicklung die höchste Priorität. Die SOLL-Anforderungen werden umgesetzt, sobald die für den Betrieb notwendigen Kernfunktionen ausreichend stabil sind. Die KANN-Anforderungen werden bewusst nachrangig behandelt. Dadurch soll verhindert werden, dass optionale Erweiterungen auf Kosten der Zuverlässigkeit, der systematischen Verifikation oder der Dokumentation der Kernfunktionen umgesetzt werden.

3.4 Zusätzliche technische Anforderungen

Die im Pflichtenheft definierten Anforderungen beschreiben die übergeordneten Funktionen des Systems. Für deren vollständige technische Umsetzung sind jedoch weitere Detailanforderungen notwendig.

Diese stellen keine zusätzlichen MUSS-Ziele ausserhalb der genehmigten Themeneingabe dar, sondern konkretisieren die technischen Funktionen, die zur Erfüllung der übergeordneten Projektziele erforderlich sind.

3.4.1 Schachlogik

Damit vollständige Partien sowohl zwischen zwei Personen als auch gegen Stockfish durchgeführt werden können, muss die Arduino-Software die für das Projekt relevanten Schachregeln korrekt verarbeiten.

Das System muss zunächst zwischen weissen und schwarzen Figuren unterscheiden und die korrekte Zugreihenfolge einhalten. Nach dem Anheben einer Figur werden die möglichen Zielfelder auf Grundlage der aktuellen Brettposition bestimmt. Eine Bewegung darf nur akzeptiert werden, wenn sie entsprechend den implementierten Schachregeln zulässig ist.

Neben den grundlegenden Bewegungsregeln der einzelnen Figuren muss die Zugvalidierung auch die **Königssicherheit** berücksichtigen. Ein Spielzug darf nicht akzeptiert werden, wenn sich der eigene König dadurch im Schach befindet oder durch die Bewegung einer schützenden Figur einem Angriff ausgesetzt wird. Dadurch werden beispielsweise auch Bewegungen gefesselter Figuren korrekt eingeschränkt.

Zusätzlich müssen folgende Sonderfälle verarbeitet werden:

- Rochade
- En passant
- Schach
- Schachmatt
- Patt

Diese Funktionen wurden während der Entwicklungsphase bereits implementiert und funktional getestet. Für die abschliessende Verifikation werden sie zusätzlich anhand definierter Testsituationen überprüft und dokumentiert.

Die formale Verifikation ist dabei von den bisherigen Entwicklungstests zu unterscheiden. Die während der Entwicklung durchgeführten Tests bestätigen die grundsätzliche Funktion, während die späteren definierten Testfälle einen nachvollziehbaren Nachweis für die Dokumentation liefern.

3.4.2 Physisch-digitale Synchronisation

Eine zentrale Anforderung des Gesamtsystems ist die Übereinstimmung zwischen dem tatsächlichen Zustand des physischen Schachbretts und dem intern beziehungsweise digital gespeicherten Spielzustand.

Das physische Schachbrett bildet dabei die reale Spielebene. Die digitale Darstellung muss den akzeptierten physischen Bewegungen folgen und darf nicht selbstständig einen Spielzug übernehmen, welcher auf dem realen Brett noch nicht ausgeführt wurde.

Besonders relevant ist dieses Prinzip im Spielmodus gegen Stockfish. Nachdem der menschliche Spieler einen gültigen Zug ausgeführt hat, berechnet Stockfish einen Gegenzug. Dieser Zug wird jedoch nicht unmittelbar als ausgeführt betrachtet. Der berechnete Zug wird zunächst über Node-RED an den Arduino zurückgesendet und mithilfe des LED-Systems angezeigt.

Anschliessend muss der entsprechende KI-Zug physisch auf dem Schachbrett ausgeführt werden. Der Arduino überprüft, ob die erwartete Figur bewegt und auf dem vorgesehenen Zielfeld platziert wurde. Erst nach erfolgreicher physischer Ausführung wird der Zug akzeptiert und der interne Spielzustand aktualisiert.

Dieses Vorgehen soll verhindern, dass sich der digitale Spielzustand vom tatsächlichen Zustand des physischen Schachbretts entfernt.

3.4.3 Kommunikation

Das Gesamtsystem besteht aus mehreren miteinander verbundenen Verarbeitungsebenen. Informationen müssen deshalb zuverlässig zwischen Hardware und Software übertragen werden.

Für einen vom Spieler ausgeführten Zug verläuft der Informationsfluss grundsätzlich in folgender Richtung:

Physisches Schachbrett → Arduino → serielle Schnittstelle → Node-RED → Stockfish

Bei einem von Stockfish berechneten Gegenzug wird zusätzlich der umgekehrte Kommunikationsweg benötigt:

Stockfish → Node-RED → serielle Schnittstelle → Arduino → physisches Schachbrett

Damit handelt es sich um ein bidirektional kommunizierendes System.

Neben der korrekten Übertragung der Informationen soll die Struktur der Kommunikation auch eine möglichst eindeutige Fehlersuche ermöglichen. Bei einem fehlerhaften Ergebnis muss untersucht werden können, ob die Ursache beispielsweise bei der Sensorerfassung, der Verarbeitung im Arduino, der seriellen Übertragung, der Verarbeitung in Node-RED oder der Kommunikation mit Stockfish liegt.

Die einzelnen Schnittstellen werden deshalb während der Entwicklung und später im Rahmen der Verifikation getrennt sowie im Zusammenspiel des Gesamtsystems geprüft.

3.4.4 Benutzerfeedback und Bedienung

Der Spieler soll den aktuellen Zustand des Systems erkennen können, ohne direkt auf den Arduino-Code oder die Entwicklungsumgebung von Node-RED zugreifen zu müssen.

Die Rückmeldung erfolgt dabei auf zwei Ebenen. Das LED-System am physischen Schachbrett stellt Informationen direkt während des Spiels dar. Die Node-RED-Benutzeroberfläche dient zur detaillierteren digitalen Darstellung des Spielverlaufs und des aktuellen Systemzustands.

Abhängig vom Spielmodus werden unter anderem folgende Informationen dargestellt:

- aktueller Spieler
- aktuelle digitale Brettposition
- letzter Spielzug
- Zugzähler
- vollständiges Zugprotokoll
- aktueller Spielstatus
- Stockfish-Bewertung beziehungsweise Zugqualität
- berechneter KI-Spielzug
- ausgewählter Spielmodus
- ausgewählte KI-Spielstärke

Die Benutzeroberfläche soll diese Informationen automatisch aktualisieren, sobald sich der entsprechende Spielzustand ändert.

Die Bedienung des Systems soll dabei möglichst einfach bleiben. Für den normalen Spielbetrieb sollen keine Änderungen am Programmcode erforderlich sein. Die Auswahl der vorgesehenen Spielmodi und KI-Spielstärken erfolgt über die Benutzeroberfläche.

3.5 Abnahme- und Verifikationsstrategie

Die Erfüllung der Anforderungen wird nicht ausschliesslich danach beurteilt, ob eine Funktion während der Entwicklung augenscheinlich korrekt arbeitet. Für die abschliessende Beurteilung werden definierte und reproduzierbare Prüfverfahren eingesetzt.

Abhängig von der jeweiligen Anforderung kommen unterschiedliche Verifikationsmethoden zum Einsatz.

Für hardwarebezogene Anforderungen werden insbesondere wiederholte Sensorprüfungen und Langzeittests durchgeführt. Dadurch soll neben der grundsätzlichen Funktion auch die Wiederholbarkeit und Stabilität der Sensorik beurteilt werden.

Softwarefunktionen werden anhand definierter Eingangssituationen und erwarteter Ergebnisse überprüft. Dies betrifft beispielsweise die Erkennung legaler und illegaler Spielzüge, Königssicherheit, Rochade, En passant sowie die Erkennung von Schach, Schachmatt und Patt.

Die Kommunikation zwischen den verschiedenen Systemkomponenten wird sowohl an den einzelnen Schnittstellen als auch über den vollständigen Kommunikationsweg geprüft. Dadurch kann festgestellt werden, ob Informationen korrekt zwischen Arduino, Node-RED und Stockfish übertragen und verarbeitet werden.

Vollständige Testpartien dienen zusätzlich dazu, das Zusammenspiel der einzelnen Komponenten unter realistischen Bedingungen zu überprüfen. Dabei werden insbesondere die Synchronisation zwischen physischem und digitalem Brett, das Zugprotokoll, die Spielerwechsel und die Stabilität über längere Spielabläufe betrachtet.

Wo eine quantitative Beurteilung sinnvoll ist, werden Messwerte beziehungsweise Erfolgs- und Fehlerzahlen erfasst. Dazu gehören beispielsweise die Anzahl erfolgreicher Sensorbetätigungen und korrekt verarbeiteter Spielzüge. Die Kommunikation wird im Spielbetrieb funktional geprüft; eine quantitative Zeitmessung ist dafür nicht vorgesehen.

Der grundsätzliche Ablauf der Verifikation lässt sich folgendermassen darstellen:

Anforderung → Akzeptanzkriterium → Prüfverfahren → Ergebnis → Beurteilung

Eine Anforderung wird erst dann abschliessend als erfüllt bewertet, wenn das definierte Akzeptanzkriterium anhand der vorgesehenen Prüfung nachgewiesen wurde.

Die detaillierten Testaufbauten, Testabläufe, Rohdaten und Ergebnisse werden im späteren Kapitel **Verifikation und Validierung** dokumentiert. Dort werden die Testergebnisse anschliessend den Anforderungen aus dem Pflichtenheft gegenübergestellt und die tatsächliche Zielerreichung beurteilt.

4 Konzeptentwicklung und Variantenentscheide

4.1 Ziel der Konzeptphase

Nach der Definition der Anforderungen wurde ein technisches Konzept für die Weiterentwicklung des bestehenden elektronischen Schachbretts erarbeitet.

Die Ausgangslage unterscheidet sich dabei von einer vollständigen Neuentwicklung, da mehrere Komponenten bereits aus vorgängigen Arbeiten vorhanden waren. Das physische Schachbrett mit der Reed-Switch-Matrix, dem Arduino Mega 2560 und dem LED-System wurde bereits im Rahmen der vorherigen Projektarbeit entwickelt. Zusätzlich war aus einer früheren Transferarbeit bereits eine grundlegende Node-RED-Benutzeroberfläche vorhanden.

Ziel der Konzeptphase war deshalb nicht, das gesamte System von Grund auf neu zu entwickeln. Stattdessen sollten die vorhandenen Komponenten beurteilt, gezielt verbessert und zu einem zuverlässigen Gesamtsystem zusammengeführt werden.

Im Mittelpunkt der Konzeptentwicklung standen insbesondere folgende Fragestellungen:

- Wie kann die Zuverlässigkeit der bestehenden Sensorik verbessert werden?
- Wie soll das physische Schachbrett mit dem digitalen System kommunizieren?
- Welche Aufgaben sollen vom Arduino und welche vom PC-System übernommen werden?
- Wie soll eine Schachengine in das Gesamtsystem integriert werden?
- Wie wird ein von der Schach-KI berechneter Zug zurück an das physische Schachbrett übertragen?
- Wie kann die Synchronisation zwischen physischem und digitalem Spielzustand gewährleistet werden?
- Wie kann die Systemarchitektur so aufgebaut werden, dass einzelne Komponenten separat geprüft und Fehler möglichst eindeutig lokalisiert werden können?

Das gewählte Konzept soll die definierten Anforderungen erfüllen und gleichzeitig die bereits vorhandene Hard- und Software sinnvoll weiterverwenden.

4.2 Bestehendes System als Grundlage

Zu Beginn der Diplomarbeit waren zwei wesentliche technische Grundlagen vorhanden.

Die erste Grundlage bildete das physische elektronische Schachbrett. Dieses verfügt über 64 Reed-Schalter zur Erfassung der Schachfiguren. In den Figuren befinden sich Magnete, welche die entsprechenden Sensoren betätigen. Die Verarbeitung der Sensorsignale erfolgt mit einem Arduino Mega 2560. Zusätzlich befinden sich seitlich am Schachbrett zwei LED-Matrizen, wobei jedem Spieler eine eigene LED-Matrix zugeordnet ist.

Die zweite Grundlage bildete die im Rahmen einer früheren Transferarbeit entwickelte Node-RED-Benutzeroberfläche. Diese enthielt bereits grundlegende Funktionen zur digitalen Darstellung verschiedener Spielinformationen, war jedoch noch nicht mit dem physischen Schachbrett verbunden.

Im Rahmen der Diplomarbeit werden diese beiden bestehenden Systeme zusammengeführt und um Stockfish als zusätzliche Softwarekomponente erweitert.

Das Gesamtsystem kann dadurch in vier grundlegende Ebenen unterteilt werden:

1. Physische Ebene
Schachfiguren, Magnete, Reed-Schalter und LED-Matrizen.

2. Embedded-Ebene
Arduino Mega 2560 zur Sensorerfassung, Verarbeitung der Figurenbewegungen, Schachlogik und LED-Ansteuerung.

3. Anwendungsebene
Node-RED für Kommunikation, Verarbeitung von Spielinformationen, digitale Visualisierung und Koordination der verschiedenen Softwarekomponenten.

4. Analyse- und KI-Ebene
Stockfish zur Analyse von Schachstellungen, Bewertung von Spielzügen und Berechnung von KI-Gegenzügen.

Durch diese Aufteilung entsteht ein modular aufgebautes System, bei dem jede Ebene eine klar definierte Aufgabe übernimmt.

4.3 Gesamtsystemkonzept

Das gewählte Gesamtkonzept basiert auf einer bidirektionalen Kommunikation zwischen dem physischen Schachbrett und dem Computersystem.

Führt ein menschlicher Spieler einen Zug aus, wird die physische Bewegung zunächst durch die Reed-Switch-Matrix erkannt. Der Arduino verarbeitet die Zustandsänderungen der Sensoren, bestimmt den ausgeführten Spielzug und überprüft dessen Zulässigkeit mithilfe der implementierten Schachlogik.

Nach der Erkennung eines gültigen Spielzuges wird dieser über die serielle Schnittstelle an Node-RED übertragen. Node-RED verarbeitet die empfangenen Informationen, aktualisiert die digitale Benutzeroberfläche und übermittelt die für die Analyse erforderlichen Informationen an Stockfish.

Abhängig vom ausgewählten Spielmodus übernimmt Stockfish unterschiedliche Aufgaben. Im Modus **Player versus Player (PvP)** dient Stockfish hauptsächlich zur Analyse und Bewertung der ausgeführten Spielzüge. Anschliessend führt der nächste Spieler seinen Zug direkt auf dem physischen Schachbrett aus.

Im Modus **Player versus AI (PvAI)** berechnet Stockfish zusätzlich den Gegenzug für den KI-Spieler. Dieser wird über Node-RED zurück an den Arduino übertragen. Der Arduino zeigt den erforderlichen Spielzug über das LED-System an.

Der KI-Spielzug wird jedoch erst dann als abgeschlossen betrachtet, wenn die entsprechende Figurenbewegung tatsächlich auf dem physischen Schachbrett ausgeführt und vom Arduino korrekt erkannt wurde.

Dadurch entsteht ein geschlossener Informationskreislauf zwischen der physischen und der digitalen Ebene des Systems.

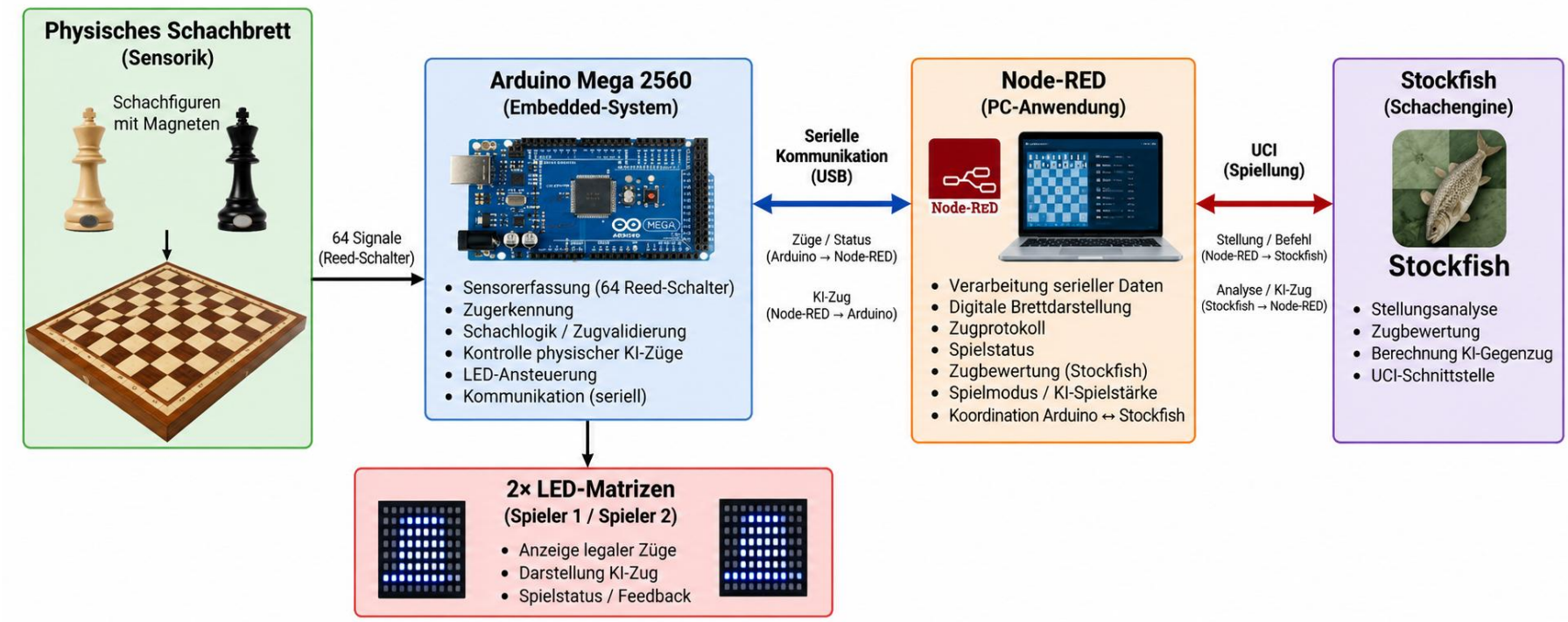


Abbildung 4-1: Systemarchitektur und Informationsfluss des intelligenten Schachbretts (eigene Darstellung)

Die Abbildung zeigt gleichzeitig die Trennung der verschiedenen Systemebenen und deren Schnittstellen. Besonders relevant ist die bidirektionale Verbindung zwischen Arduino und Node-RED. Dadurch können nicht nur physische Spielzüge an das Computersystem übertragen, sondern auch von Stockfish berechnete Spielzüge wieder an das physische System zurückgegeben werden.

4.4 Konzept zur Verbesserung der Sensorzuverlässigkeit

Die bestehende Reed-Switch-Matrix hatte bereits gezeigt, dass das magnetische Erkennungsprinzip grundsätzlich für das Projekt geeignet ist. Das Hauptproblem lag daher nicht im gewählten Sensorprinzip selbst, sondern in der mechanischen Zuverlässigkeit des bestehenden Aufbaus.

Für die weitere Vorgehensweise wurden drei grundsätzliche Varianten betrachtet.

Variante 1 – Vollständige Neuentwicklung der Sensorik

Eine Möglichkeit wäre gewesen, die bestehende Sensorik vollständig zu entfernen und durch ein neues Erkennungssystem zu ersetzen.

Dadurch hätte die mechanische Konstruktion grundsätzlich neu aufgebaut werden können. Gleichzeitig wäre jedoch ein grosser Teil des bereits funktionsfähigen Systems verworfen worden. Zusätzlich wären erheblicher Entwicklungs- und Montageaufwand sowie neue mögliche Fehlerquellen entstanden.

Diese Variante wurde deshalb nicht weiterverfolgt.

Variante 2 – Vollständige Neuverdrahtung

Eine weitere Möglichkeit bestand darin, das bestehende Reed-Switch-Prinzip beizubehalten, jedoch die gesamte Verdrahtung neu aufzubauen.

Dadurch hätten bestehende Schwachstellen der alten Verbindungen beseitigt werden können. Gleichzeitig wäre auch bei dieser Variante ein erheblicher Arbeitsaufwand entstanden. Zudem hätte eine vollständige Neuverdrahtung wiederum neue Montage- und Lötfehler verursachen können.

Variante 3 – Gezielte Verbesserung des bestehenden Aufbaus

Als gewählte Lösung wurde die bestehende Reed-Switch-Matrix beibehalten und die tatsächlich auftretenden Schwachstellen gezielt untersucht und verbessert.

Bei der Fehlersuche zeigte sich, dass insbesondere Bewegungen der Dioden beziehungsweise die dadurch entstehende mechanische Belastung der Lötstellen zu unzuverlässigen Verbindungen führen konnten.

Die betroffenen Verbindungen wurden deshalb nachgelötet. Zusätzlich wurden die Dioden mit Heisskleber mechanisch fixiert. Dadurch sollen Bewegungen der Bauteile reduziert und die Lötstellen entlastet werden.

Diese Variante wurde gewählt, da sie direkt an der beobachteten Fehlerursache ansetzt und gleichzeitig die bereits funktionierenden Komponenten des bestehenden Systems weiterverwendet.

Die Wirksamkeit der Massnahme wird nicht allein aufgrund der unmittelbar wiederhergestellten Funktion beurteilt, sondern durch wiederholte Sensorprüfungen und Langzeittests überprüft.

4.5 Konzept für die Kommunikation zwischen Arduino und PC

Für die Verbindung des physischen Schachbretts mit dem Computersystem wird eine Kommunikationsschnittstelle benötigt, welche Informationen zuverlässig in beide Richtungen übertragen kann.

Grundsätzlich wären sowohl drahtlose als auch kabelgebundene Kommunikationslösungen möglich gewesen. Für den vorliegenden Prototyp wurde eine serielle Kommunikation über USB gewählt.

Für diese Entscheidung waren insbesondere folgende Punkte ausschlaggebend:

- der Arduino Mega 2560 verfügt bereits über eine geeignete serielle Schnittstelle,
- es wird keine zusätzliche Kommunikationshardware benötigt,
- die Verbindung ist einfach einzurichten,
- übertragene Daten können während der Entwicklung direkt kontrolliert werden,
- Node-RED kann serielle Daten empfangen und senden,
- das Schachbrett wird als stationärer Prototyp eingesetzt,
- die Kommunikation muss bidirektional funktionieren.

Für die Datenübertragung werden definierte textbasierte Nachrichten verwendet. Dadurch können die übertragenen Informationen während der Entwicklung direkt gelesen und mithilfe von Debug-Ausgaben überprüft werden.

So kann beispielsweise ein erkannter Spielzug vom Arduino an Node-RED übertragen werden. In umgekehrter Richtung kann Node-RED einen von Stockfish berechneten KI-Spielzug an den Arduino senden.

Die serielle Kommunikation bildet somit eine zentrale Schnittstelle zwischen der physischen und der digitalen Ebene des Gesamtsystems.

4.6 Konzept der digitalen Benutzeroberfläche

Die PC-seitige Anwendung übernimmt innerhalb des Gesamtsystems drei wesentliche Aufgaben:

1. Kommunikation mit dem Arduino,
2. Verarbeitung und Visualisierung der Spielinformationen,
3. Kommunikation mit Stockfish.

Für diese Aufgaben wurde Node-RED weiterverwendet. Ein wesentlicher Grund dafür ist, dass bereits im Rahmen der früheren Transferarbeit eine grundlegende Benutzeroberfläche entwickelt worden war.

Anstatt eine vollständig neue Anwendung zu entwickeln, konnte diese Grundlage gezielt erweitert und mit dem physischen Schachbrett verbunden werden.

Ein weiterer Vorteil liegt im modularen Aufbau von Node-RED. Serielle Eingänge, Datenverarbeitung, Zugprotokoll, Brettdarstellung, Stockfish-Kommunikation und Dashboardausgaben können in getrennten Nodes beziehungsweise Funktionen verarbeitet werden.

Diese Struktur unterstützt insbesondere die Fehlersuche. Durch Debug-Ausgaben können Daten an verschiedenen Stellen des Informationsflusses überprüft werden. Dadurch lässt sich beispielsweise unterscheiden, ob ein Fehler bei der Datenübertragung, der Verarbeitung oder lediglich bei der Darstellung im Dashboard entsteht.

Node-RED bildet damit die zentrale Softwareplattform zwischen dem physischen Schachbrett, der Benutzeroberfläche und Stockfish.

4.7 Auswahl der Schachengine

Für das Projekt wird eine Schachengine für zwei wesentliche Funktionen benötigt:

- Analyse und Bewertung ausgeführter Spielzüge,
- Berechnung der Spielzüge des KI-Gegners.

Für diese Aufgaben wurde Stockfish gewählt.

Ein wesentlicher Vorteil für die Integration in das Gesamtsystem ist die Unterstützung des **Universal Chess Interface (UCI)**. Dadurch kann die Engine über definierte Befehle angesprochen werden und Informationen über die aktuelle Spielstellung verarbeiten.

Stockfish kann dadurch als eigenständige Softwarekomponente eingesetzt werden. Die Entwicklung einer eigenen Schach-KI ist nicht Bestandteil der Diplomarbeit.

Der technische Schwerpunkt liegt stattdessen auf der Integration einer bestehenden Schachengine in das physische System. Dazu gehören insbesondere die korrekte Übertragung der Spielstellung, die Verarbeitung der Engine-Ausgaben, die Bewertung ausgeführter Spielzüge sowie die Rückübertragung berechneter KI-Züge an das physische Schachbrett.

Dadurch kann eine leistungsfähige Schachengine verwendet werden, während sich die Entwicklungsarbeit auf die für das Projekt relevanten Schnittstellen und Systemfunktionen konzentriert.

4.8 Konzept zur Ausführung der KI-Spielzüge

Eine wichtige Konzeptentscheidung betrifft die Frage, wie ein von Stockfish berechneter Spielzug auf dem physischen Schachbrett umgesetzt werden soll.

Dafür wurden drei Varianten betrachtet.

Variante 1 – Ausschliessliche Anzeige auf dem Bildschirm

Die technisch einfachste Möglichkeit wäre, den berechneten KI-Spielzug lediglich auf der digitalen Benutzeroberfläche anzuzeigen.

Diese Variante würde jedoch nur eine eingeschränkte Verbindung zwischen der Schach-KI und dem physischen Schachbrett herstellen.

Variante 2 – Vollautomatische mechanische Ausführung

Eine weitere Möglichkeit wäre die automatische Bewegung der Schachfiguren mithilfe eines Roboters beziehungsweise einer mechanischen Bewegungseinrichtung.

Dadurch könnte ein hoher Automatisierungsgrad erreicht werden. Gleichzeitig würde dies den mechanischen und steuerungstechnischen Aufwand des Projekts deutlich erhöhen.

Eine solche Erweiterung ist deshalb als KANN-Ziel vorgesehen und gehört nicht zum erforderlichen Kernumfang des Systems.

Variante 3 – LED-geführte physische Ausführung

Bei der gewählten Variante wird der von Stockfish berechnete Spielzug über Node-RED an den Arduino zurückgesendet.

Der Arduino zeigt anschliessend mithilfe des LED-Systems an, welche Bewegung ausgeführt werden soll. Der Benutzer bewegt die entsprechende KI-Figur physisch auf dem Schachbrett.

Der Arduino kontrolliert dabei, ob tatsächlich der erwartete Spielzug ausgeführt wurde. Erst nach erfolgreicher Erkennung wird der KI-Zug akzeptiert und der Spielablauf fortgesetzt.

Diese Variante wurde gewählt, da sie eine echte bidirektionale Interaktion zwischen Schach-KI und physischem Schachbrett ermöglicht, ohne die hohe mechanische Komplexität eines Robotersystems zu verursachen.

Gleichzeitig verbessert dieses Vorgehen die Synchronisation zwischen physischem und digitalem Spielzustand.

4.9 Konzept zur Synchronisation des Spielzustands

Die Synchronisation zwischen dem physischen und dem digitalen Brettzustand stellt einen zentralen Bestandteil des Systemkonzepts dar.

Eine mögliche Vorgehensweise wäre, den digitalen Spielzustand unabhängig vom physischen Schachbrett weiterzuführen. Dies könnte jedoch dazu führen, dass die digitale Stellung bereits verändert wird, obwohl die entsprechende Bewegung auf dem physischen Brett noch nicht oder falsch ausgeführt wurde.

Im gewählten Konzept bildet deshalb der akzeptierte physische Brettzustand die entscheidende Grundlage für den weiteren Spielverlauf.

Bei einem menschlichen Spielzug wird die physische Bewegung zunächst vom Arduino erkannt und validiert. Erst danach wird der Spielzug an das digitale System weitergegeben.

Bei einem KI-Spielzug wird der berechnete Zug zunächst an das physische System zurückgesendet. Erst nachdem die entsprechende Bewegung tatsächlich ausgeführt und korrekt erkannt wurde, gilt der Spielzug als abgeschlossen.

Daraus ergibt sich ein wichtiges Grundprinzip des Systems:

Die Berechnung oder Anzeige eines Spielzuges bedeutet noch nicht, dass dieser Spielzug ausgeführt wurde. Ein Zug gilt erst nach der korrekten physischen Ausführung und Erkennung als abgeschlossen.

Dieses Prinzip reduziert das Risiko einer dauerhaften Abweichung zwischen physischem und digitalem Spielzustand.

4.10 Bewertung der Konzeptvarianten

Die betrachteten Lösungsvarianten wurden hinsichtlich ihres Aufwands und ihres Nutzens für das Projekt miteinander verglichen.

Tabelle 4-1: Vergleich und Bewertung der betrachteten Konzeptvarianten

Bereich	Variante	Aufwand	Nutzen für das Projekt	Entscheid
Sensorik	Vollständige Neuentwicklung	Hoch	Potenziell hoch	Nicht gewählt
Sensorik	Vollständige Neuverdrahtung	Hoch	Mittel bis hoch	Nicht gewählt
Sensorik	Gezielte Reparatur und mechanische Stabilisierung	Niedrig bis mittel	Hoch	Gewählt
PC-Kommunikation	Drahtlose Kommunikation	Mittel	Mittel	Nicht gewählt
PC-Kommunikation	Serielle USB-Kommunikation	Niedrig	Hoch	Gewählt
Benutzeroberfläche	Neue Anwendung entwickeln	Hoch	Mittel	Nicht gewählt
Benutzeroberfläche	Bestehendes Node-RED-System erweitern	Mittel	Hoch	Gewählt
KI-Spielzug	Nur am Bildschirm anzeigen	Niedrig	Niedrig bis mittel	Nicht gewählt
KI-Spielzug	Automatisch mit Robotersystem ausführen	Sehr hoch	Hoch	KANN-Ziel
KI-Spielzug	LED-geführte physische Ausführung	Mittel	Hoch	Gewählt
Schach-KI	Eigene Schachengine entwickeln	Sehr hoch	Gering für das Projektziel	Nicht gewählt
Schach-KI	Stockfish integrieren	Mittel	Sehr hoch	Gewählt

Die Bewertung zeigt, dass bei der Konzeptauswahl nicht der maximale technische Funktionsumfang im Vordergrund stand. Entscheidend war vielmehr, mit einem angemessenen Entwicklungsaufwand eine zuverlässige und überprüfbare Lösung für die definierten Projektziele zu erreichen.

4.11 Gewähltes Gesamtkonzept

Auf Grundlage der Anforderungen und der Bewertung der verschiedenen Lösungsansätze wurde entschieden, das bestehende physische Schachbrett gezielt weiterzuentwickeln und nicht vollständig neu aufzubauen.

Der Arduino Mega 2560 bleibt die zentrale Steuerung der physischen Ebene. Er verarbeitet die Reed-Switch-Matrix, erkennt und validiert physische Figurenbewegungen, verarbeitet die relevante Schachlogik und steuert das LED-System.

Node-RED bildet die zentrale Verbindung zwischen der physischen Hardware, der digitalen Benutzeroberfläche und Stockfish. Es empfängt Spielzüge vom Arduino, verarbeitet und visualisiert die Spielinformationen, kommuniziert mit Stockfish und übermittelt berechnete KI-Spielzüge wieder zurück an den Arduino.

Stockfish stellt die für die Schachanalyse und den KI-Spielmodus benötigte Rechenfunktionalität zur Verfügung.

Die Systemarchitektur kann dadurch vereinfacht folgendermassen zusammengefasst werden:

Schachfiguren und Magnete → Reed-Switch-Matrix → Arduino Mega 2560 ↔ Node-RED ↔ Stockfish

Zusätzlich steuert der Arduino die beiden physischen LED-Matrizen, während Node-RED die digitale Benutzeroberfläche bereitstellt.

Das gewählte Gesamtkonzept verbindet damit die bereits vorhandenen Hardware- und Softwaregrundlagen mit den im Rahmen der Diplomarbeit neu entwickelten Kommunikations-, Analyse- und KI-Funktionen.

Entscheidend ist dabei, dass nicht mehrere voneinander unabhängige Funktionen entstehen, sondern ein **zusammenhängendes bidirektionales Gesamtsystem**, in dem physische und digitale Prozesse miteinander verknüpft sind.

4.12 Zusammenfassung

Im Rahmen der Konzeptentwicklung wurde festgelegt, wie die in Kapitel 3 definierten Anforderungen technisch umgesetzt werden sollen.

Anstelle einer vollständigen Neuentwicklung wurde eine gezielte Weiterentwicklung des bestehenden Systems gewählt. Bereits vorhandene Komponenten werden dort weiterverwendet, wo sie ihre Aufgabe grundsätzlich erfüllen. Festgestellte Schwachstellen werden gezielt verbessert und fehlende Funktionen ergänzt.

Die gewählte Architektur unterteilt das Gesamtsystem in die physische Ebene, die Embedded-Ebene, die PC-seitige Anwendung und die Schachengine. Zwischen diesen Ebenen bestehen klar definierte Schnittstellen.

Das zentrale Merkmal des Konzepts ist die bidirektionale Verbindung zwischen dem physischen Schachbrett und dem digitalen System. Physisch ausgeführte Spielzüge werden an das Computersystem übertragen, während von Stockfish berechnete KI-Spielzüge wieder an das Schachbrett zurückgegeben werden.

Dadurch entsteht die technische Grundlage für ein intelligentes Schachbrett, bei dem Sensorik, Embedded-Software, digitale Benutzeroberfläche und Schach-KI als gemeinsames Gesamtsystem zusammenarbeiten.

5 Systemarchitektur

Die Systemarchitektur verbindet das physische Schachbrett, den Arduino Mega 2560, die LED-Anzeigen, eine Node-RED-Anwendung und die Schachengine Stockfish. Bei der Konzeption stand die Interaktion mit dem physischen Brett im Mittelpunkt: Ein Zug gilt erst dann als physisch abgeschlossen, wenn die Sensormatrix die entsprechende Veränderung auf dem Brett erkannt hat. Die Komponenten übernehmen klar abgegrenzte Aufgaben. Dadurch konnte die bestehende Schachbretthardware beibehalten und um digitale Funktionen ergänzt werden.

5.1 Hardwarearchitektur

Die physische Eingabe erfolgt über 64 Reed-Schalter unter den 64 Feldern des 8 × 8-Schachbretts. Magnete in den Schachfiguren betätigen die Schalter, sodass das System die Belegung der einzelnen Felder erkennen kann. Die Schalter erfassen jedoch nicht, welche Figur auf einem Feld steht. Die Identität der Figuren und ihre zulässigen Bewegungen werden deshalb anhand der aktuellen Spielstellung durch die Software bestimmt.

Die Sensoren sind als Zeilen-Spalten-Matrix verdrahtet. An jedem Reed-Schalter ist eine Diode angeschlossen, um Ghosting zu verhindern. Der Arduino Mega 2560 fragt die Matrix über acht Zeilen- und acht Spaltenanschlüsse ab. Bei der bestehenden Hardware sind die Spalten mit den digitalen Pins D22–D29 und die Zeilen mit D30–D37 verbunden. Die WS2812B-LEDs werden über den digitalen Pin D6 angesteuert.

Seitlich am physischen Schachbrett befinden sich zwei WS2812B-LED-Matrizen, jeweils eine pro Spieler. Sie ermöglichen eine visuelle Führung, ohne dass unter jedem einzelnen Spielfeld LEDs angebracht werden müssen. Der Arduino steuert die Leuchtmuster zur Anzeige von Zügen, möglichen Zielfeldern und zur Anleitung bei der physischen Ausführung eines KI-Zuges.

Tabelle 5-1: Wichtigste Hardwarekomponenten und ihre Aufgaben

Komponente	Funktion	Anschluss / Anordnung
64 Reed-Schalter	Erkennen der Feldbelegung	8×8-Sensormatrix
64 Dioden	Verhindern von Ghosting	Eine Diode pro Reed-Schalter
Arduino Mega 2560	Sensoren abfragen und lokale Spiellogik steuern	Zeilen D30–D37; Spalten D22–D29
Zwei WS2812B-LED-Matrizen	Visuelle Rückmeldung für beide Spieler	Seitlich montiert; LED-Datenleitung an D6
PC / Hostsystem	Node-RED und Stockfish ausführen	Serielle USB-Verbindung zum Arduino

Durch diesen Aufbau verbleiben die elektrische Eingabe, die eingebettete Steuerung und die visuelle Rückmeldung im physischen Schachbrett. Die rechenintensiveren Funktionen der Schachengine können hingegen auf dem angeschlossenen Computer ausgeführt werden.

5.2 Softwarearchitektur

Die Software ist auf den Arduino und den angeschlossenen Computer aufgeteilt. Der Arduino wertet die Sensormatrix fortlaufend aus, verwaltet die interne Schachstellung, erkennt physische Züge, prüft deren Zulässigkeit und erzeugt unmittelbar eine LED-Rückmeldung. Die Spiellogik berücksichtigt auch Sonderregeln wie Rochade, En passant und Bauernumwandlung sowie die Sicherheit des Königs, Schach, Schachmatt und Patt.

Node-RED bildet die Verbindung zwischen dem eingebetteten Schachbrettsystem und den computerbasierten Funktionen. Die Anwendung empfängt über die serielle Schnittstelle Informationen vom Arduino, aktualisiert das digitale Schachbrett und den Zugverlauf und koordiniert die Kommunikation mit Stockfish. Über das Dashboard kann die laufende Partie digital verfolgt werden, während die Züge weiterhin auf dem physischen Brett ausgeführt werden.

Stockfish übernimmt die Schachanalyse und die Rolle des Computergegners. Die Engine wird über das Universal Chess Interface (UCI) angesprochen. Node-RED übermittelt die jeweilige Spielstellung an Stockfish und empfängt die Analyse oder den ausgewählten Antwortzug. Die Verarbeitung durch die Schachengine bleibt damit vom Arduino getrennt, dessen Hauptaufgabe die zuverlässige Interaktion mit dem physischen Brett ist.

Die klare Aufgabenteilung erleichtert zudem die Fehlersuche: Probleme mit Sensoren oder LEDs können auf der Arduino-Ebene untersucht werden, während Fehler im Dashboard oder bei der Schachengine unabhängig davon am Computer analysiert werden können.

5.3 Datenfluss und Schnittstellen

Die Kommunikation zwischen Arduino und Node-RED erfolgt über eine serielle USB-Verbindung. Der Arduino sendet Informationen zu erkannten physischen Zügen und relevanten Spielzuständen. Node-RED verarbeitet diese Nachrichten, aktualisiert die digitale Darstellung und kommuniziert bei Bedarf mit Stockfish. Die genauen Formate der seriellen Nachrichten werden durch die implementierte Software festgelegt und hier nicht durch ungeprüfte Beispiele ersetzt.

Bei einer Partie gegen die KI führt zunächst der menschliche Spieler einen Zug auf dem physischen Schachbrett aus. Der Arduino erkennt und validiert diesen Zug und übermittelt ihn an Node-RED. Node-RED aktualisiert die Spielstellung und fordert von Stockfish einen Antwortzug an. Sobald Stockfish einen Zug zurückliefert, leitet Node-RED diesen an den Arduino weiter. Dieser zeigt die erforderliche Bewegung mithilfe der LEDs an.

Das Schachbrett bewegt die Figuren nicht selbstständig. Der menschliche Spieler muss den durch die LEDs angezeigten KI-Zug physisch ausführen. Der Arduino vergleicht die anschliessenden Sensoränderungen mit dem erwarteten Zug. Eine falsche physische Bewegung wird nicht als abgeschlossener KI-Zug akzeptiert. Diese Überprüfung verhindert, dass das digitale Spiel unabhängig vom physischen Brett fortgesetzt wird.

5.4 Kapitelzusammenfassung

Die Aufgabenteilung zwischen Arduino, Node-RED und Stockfish bildet die Grundlage für die nachfolgend beschriebene Umsetzung.

6 Hardwareentwicklung und Zuverlässigkeit

Die Zuverlässigkeit der physischen Hardware ist für den Betrieb des intelligenten Schachbretts entscheidend. Selbst eine fehlerfreie Schachsoftware und Kommunikation mit Stockfish können unzuverlässige Sensorsignale nicht ausgleichen. Dieses Kapitel beschreibt den Zustand der vorhandenen Hardware, die erkannten Schwachstellen sowie die durchgeführten Reparaturen und Prüfungen.

6.1 Ausgangszustand des Sensorsystems

Das physische Schachbrett stammt aus einem früheren Projekt. Unter den 64 Feldern befinden sich Reed-Schalter, die durch Magnete in den Figuren betätigt werden. Ihre Signale werden über eine Matrix vom Arduino erfasst.

Obwohl ursprünglich alle 64 Positionen funktionierten, nahm die Zuverlässigkeit mit der Zeit ab. Zu Beginn der Diplomarbeit reagierte ungefähr die Hälfte der Sensorpositionen nicht mehr zuverlässig. Deshalb musste zunächst die bestehende Hardware instand gesetzt werden.

6.1.1 Analyse der bestehenden Hardware

Das Sensorsystem wurde untersucht, um zwischen defekten Reed-Schaltern, Fehlern in der Matrixverdrahtung und mangelhaften elektrischen Verbindungen zu unterscheiden. Besonderes Augenmerk galt den manuell gelöteten Anschlüssen und den Dioden, da mechanische Belastungen dort Kontaktprobleme verursachen können.

6.2 Ghosting und Funktion der Dioden

Die Reed-Schalter sind als 8 × 8-Matrix verschaltet. Dadurch werden statt 64 einzelner Eingänge lediglich acht Zeilen- und acht Spaltenleitungen benötigt. Bei mehreren gleichzeitig betätigten Schaltern können jedoch unerwünschte Strompfade entstehen.

6.2.1 Erklärung des Ghostings

Ghosting entsteht, wenn Strom über mehrere geschlossene Schalter fliesst und dadurch ein zusätzlicher, tatsächlich nicht betätigter Schalter als aktiv erkannt wird. Auf einem Schachbrett mit zahlreichen gleichzeitig aufgestellten Figuren ist dieses Problem besonders relevant. Falsche Belegungsinformationen würden die Zugerkennung beeinträchtigen.

6.2.2 Einsatz der Dioden

Zu jedem Reed-Schalter ist eine Diode in Serie geschaltet. Sie verhindert unerwünschte Rückstrompfade in der Matrix, sodass mehrere belegte Felder gleichzeitig zuverlässig erkannt werden können. Die Dioden lösen das elektrische Ghosting-Problem, schützen ihre eigenen Anschlüsse jedoch nicht vor mechanischer Belastung.

6.3 Mechanische Fehlerursachen und Verbesserungen

6.3.1 Festgestellte Schwachstelle

Die von Hand aufgebaute Matrix weist weniger mechanisch geschützte Bauteile auf als eine professionell gefertigte Leiterplatte. Bewegungen der Dioden und Leitungen können Kräfte auf die Lötstellen übertragen. Dadurch können zeitweise oder dauerhafte Unterbrechungen entstehen.

6.3.2 Reparatur der elektrischen Verbindungen

Die vorhandenen Verbindungen wurden kontrolliert und unzuverlässige Lötstellen nachgearbeitet, um die elektrische Durchgängigkeit wiederherzustellen. Ziel war es, sämtliche 64 Sensorpositionen zu reparieren, ohne die gesamte Matrix neu aufbauen zu müssen.

6.3.3 Mechanische Stabilisierung mit Heisskleber

Die Dioden wurden mit Heisskleber fixiert, um Bewegungen und die mechanische Beanspruchung ihrer elektrischen Anschlüsse zu reduzieren. Der Klebstoff verbessert die elektrische Leitfähigkeit nicht; er dient ausschliesslich der mechanischen Stabilisierung der bestehenden Verbindungen.

6.3.4 Bewertung der gewählten Lösung

Ein vollständiger Neuaufbau der Matrix oder die Herstellung einer Leiterplatte könnte eine bessere mechanische Integration ermöglichen. Diese Varianten würden jedoch deutlich mehr Zeit und Aufwand erfordern. Für den vorhandenen Prototyp wurde deshalb die gezielte Reparatur mit zusätzlicher Fixierung gewählt.

6.4 Überprüfung der Hardwareverbesserung

Nach den ersten elektrischen Reparaturen und der mechanischen Stabilisierung wurde die Sensormatrix mit Funktionsprüfungen, einem zehntägigen Wiederholungstest und mechanischen Belastungsprüfungen untersucht. Diese Tests sollten sowohl die normale Funktion als auch die Empfindlichkeit gegenüber Bewegungen beurteilen.

6.4.1 Erste Funktionsprüfung

Alle 64 Felder wurden einzeln mit einer magnetbestückten Schachfigur aktiviert und die Reaktion am Arduino überprüft. Nach der ersten Reparatur reagierten sämtliche Sensorpositionen wieder korrekt. Damit war die Grundlage für die anschliessenden Zuverlässigkeitsprüfungen geschaffen.

6.4.2 Zehntägiger Sensorbetätigungstest

Über zehn Tage wurde jede der 64 Sensorpositionen täglich zweimal betätigt. Daraus ergeben sich insgesamt 64 × 2 × 10 = 1280 Bestätigungen. Alle 1280 Aktivierungen wurden während dieser Prüfung korrekt erkannt.

Tabelle 6-1: Zehntägiger Sensorbetätigungstest

Prüfparameter	Wert
Sensorpositionen	64
Betätigungen je Position und Tag	2
Betätigungen pro Tag	128
Prüfdauer	10 Tage
Betätigungen insgesamt	1280

Der Betätigungstest überprüfte vor allem die elektrischen Reaktionen bei vergleichsweise ruhiger Lage des Schachbretts. Daraus allein liess sich nicht ableiten, ob die Verbindungen auch bei Transport oder stärkeren Bewegungen stabil bleiben würden.

6.4.3 Mechanischer Belastungstest und weitere Reparaturen

Nach dem zehntägigen Test wurde das gesamte Schachbrett gezielt bewegt, geschüttelt und umgedreht. Diese Belastungen sollten Situationen nachbilden, in denen die manuell aufgebauten elektrischen Verbindungen mechanisch beansprucht werden.

Nach dem ersten mechanischen Belastungstest funktionierten 11 Reed-Schalter-Positionen nicht mehr korrekt. Die betroffenen Stellen und ihre Verbindungen wurden erneut überprüft und repariert. Anschliessend wurde die Funktion aller 64 Felder wieder kontrolliert.

Die Überprüfung endete nicht mit der erneuten Funktionskontrolle. Es folgte ein abschliessender mechanischer Belastungstest, bei dem das Schachbrett nochmals bewegt, geschüttelt und umgedreht wurde. Danach funktionierten weiterhin alle 64 Sensorpositionen.

6.4.4 Auswertung der Testergebnisse

Tabelle 6-2: Übersicht über die Sensorprüfungen und Reparaturschritte

Prüfschritt	Beobachtung / Ergebnis
Erste Funktionsprüfung	Alle 64 Sensorpositionen funktionsfähig
Zehntägiger Betätigungstest	128 Betätigungen pro Tag; insgesamt 1280
Erster mechanischer Belastungstest	Danach 11 fehlerhafte Sensorpositionen festgestellt
Zusätzliche Reparaturen	Betroffene Positionen repariert
Erneute Funktionsprüfung	Alle 64 Positionen funktionsfähig
Abschliessender mechanischer Belastungstest	Bestanden; keine weiteren Sensorausfälle festgestellt

6.4.5 Grenzen der Überprüfung

Die mechanischen Prüfungen wurden manuell durchgeführt. Bewegung, Schütteln und Umdrehen wurden nicht hinsichtlich Kraft, Beschleunigung, Schwingungsfrequenz oder Dauer messtechnisch erfasst. Deshalb sind die Belastungen nicht exakt reproduzierbar und nicht mit genormten Prüfverfahren gleichzusetzen.

Aus dem Ausfall einer Sensorposition lässt sich nicht unmittelbar auf ein einzelnes defektes Bauteil schliessen. Als Ursache kommen der Reed-Schalter, Lötstellen, Diodenanschlüsse oder Leitungsverbindungen infrage. Die Fehlersuche muss deshalb die gesamte betroffene Signalstrecke berücksichtigen.

Der erfolgreiche abschliessende Belastungstest bestätigt, dass die reparierte Baugruppe die manuell aufgebrachten Belastungen dieser Prüfung überstanden hat. Er garantiert jedoch nicht, dass bei jeder zukünftigen Belastung oder über eine unbegrenzte Betriebsdauer keine weiteren Fehler auftreten können.

6.5 Kapitelzusammenfassung

Die anfänglich unzuverlässige Sensormatrix konnte durch die Überprüfung und Nacharbeit der Verbindungen wieder funktionsfähig gemacht werden.

Nach 1280 erfolgreichen Sensorbetätigungen zeigten sich im ersten mechanischen Belastungstest elf fehlerhafte Positionen, die erneut repariert wurden.

Der abschliessende Belastungstest verlief ohne weitere Sensorausfälle. Die Grenzen der manuellen Prüfungen sind in Abschnitt 6.4.5 erläutert.

7 Embedded-Software auf dem Arduino

Der Arduino Mega 2560 verarbeitet die physischen Bewegungen auf dem Schachbrett und steuert die LED-Rückmeldungen. Er bildet die Verbindung zwischen der Reed-Schalter-Matrix, der internen Schachlogik und der Kommunikation mit Node-RED.

Die Software muss mehr leisten als die reine Erkennung belegter Felder. Sie muss Sensoränderungen als Schachzüge interpretieren, die Zulässigkeit prüfen und das Ergebnis mit der gespeicherten Spielstellung abgleichen.

Dies ist besonders wichtig, weil Reed-Schalter keine Figurenarten erkennen. Sie zeigen lediglich an, ob sich ein Magnet über einem Feld befindet. Die Software muss daher zusätzlich wissen, welche Figur sich laut internem Spielzustand auf diesem Feld befindet.

Die Arduino-Software wurde in der Arduino IDE entwickelt. Für die WS2812B-LEDs wird die Bibliothek Adafruit NeoPixel verwendet. Die wesentlichen Aufgaben umfassen die Sensorabfrage, die Verwaltung des Spielzustands, die Zugprüfung und die visuelle Rückmeldung.

7.1 Sensorabfrage und Zustandsmodell

Das Schachbrett besitzt 64 Reed-Schalter in einer 8 × 8-Matrix. Die Spalten sind mit D22–D29 und die Zeilen mit D30–D37 verbunden. Mit dieser Verschaltung kann der Arduino sämtliche Felder über 16 digitale Anschlüsse abfragen.

7.1.1 Abfrage der Sensormatrix

Der Arduino liest die Sensormatrix fortlaufend aus und ermittelt, welche Felder gerade belegt sind. Bei jeder Abfrage werden die elektrischen Zustände der Zeilen und Spalten ausgewertet und den entsprechenden Schachfeldern zugeordnet.

Die erkannte Belegung wird in einem 8 × 8-Array gespeichert. Jeder Eintrag steht für ein Feld und kennzeichnet, ob sich dort eine Figur befindet. Durch den Vergleich der aktuellen mit der vorherigen Messung lassen sich Veränderungen erkennen.

Wird beispielsweise eine Figur von e2 angehoben, wechselt der zugehörige Reed-Schalter von belegt zu frei. Wird sie anschliessend auf e4 abgestellt, wechselt das Zielfeld von frei zu belegt. Aus dieser Folge kann die Software einen möglichen Zug ableiten.

Die Sensormatrix liefert somit Informationen über die physische Bewegung. Ob diese Bewegung einen gültigen Schachzug darstellt, entscheidet die interne Schachlogik.

7.1.2 Interne Brettdarstellung

Neben den Sensorsignalen führt der Arduino eine interne Darstellung der Schachstellung. Diese enthält Informationen, die sich aus der reinen Feldbelegung nicht ermitteln lassen, beispielsweise Figurenarten, Farben und das Zugrecht.

Zu Beginn einer normalen Partie wird die interne Brettdarstellung mit der üblichen Schachanfangsstellung initialisiert. Nach jedem akzeptierten Zug wird die gespeicherte Stellung entsprechend aktualisiert.

Die Trennung zwischen physischer Belegung und logischer Brettdarstellung ist entscheidend. Ein Reed-Schalter kann eine Figur erkennen, aber nicht feststellen, ob es sich um einen Bauern, Turm oder König handelt. Diese Information muss aus dem gespeicherten Zustand stammen.

Die interne Stellung darf nicht bei jeder Sensoränderung sofort angepasst werden. Das Anheben oder Verschieben einer Figur kann vorübergehend einen Zustand erzeugen, der noch keinen abgeschlossenen und gültigen Zug darstellt.

7.1.3 Interpretation physischer Bewegungen

Ein normaler Schachzug besteht aus dem Anheben einer Figur vom Startfeld und dem Abstellen auf dem Zielfeld. Der Arduino muss beide Vorgänge als zusammengehörende Bewegung erkennen.

Beim Anheben einer Figur bestimmt die Software das Ausgangsfeld und berechnet anhand der internen Stellung die erlaubten Zielfelder. Die LED-Anzeige unterstützt den Spieler bei der Auswahl des nächsten Schritts.

Ist der Zug gültig, aktualisiert die Software die interne Schachstellung und kann den abgeschlossenen Zug an Node-RED melden. Bei einem unerlaubten Zug wird die Stellung nicht übernommen; stattdessen erhält der Spieler eine visuelle Fehlermeldung.

Schlagzüge und Sonderzüge erfordern zusätzliche Behandlung, da mehrere Belegungsänderungen auftreten können. Bei einem Schlagzug wird beispielsweise eine gegnerische Figur entfernt, bevor oder während der eigenen Figur auf das Zielfeld gesetzt wird.

7.2 Zuglogik und erlaubte Züge

Die Zuglogik ist ein zentraler Bestandteil der Arduino-Software. Sie prüft, ob eine physisch erkannte Bewegung den Schachregeln entspricht. Dazu werden die gespeicherte Stellung, die aktive Spielerfarbe und die Bewegungsmöglichkeiten der Figuren berücksichtigt.

7.2.1 Bewegungsregeln der Schachfiguren

Die grundlegenden Bewegungsregeln wurden für alle sechs Figurenarten umgesetzt. Bauern ziehen normalerweise ein Feld vorwärts und unter bestimmten Bedingungen zwei Felder vom Ausgangsfeld. Sie schlagen diagonal und besitzen besondere Regeln.

Türme ziehen horizontal oder vertikal über freie Felder; Läufer bewegen sich diagonal. Die Dame kombiniert beide Bewegungsarten. Springer können andere Figuren überspringen, während der König normalerweise ein Feld in jede Richtung zieht.

Bei Figuren mit längeren Zugwegen muss die Software kontrollieren, ob andere Figuren den Weg blockieren. Ein Turm darf beispielsweise nicht durch eine belegte Zwischenposition ziehen. Zusätzlich dürfen eigene Figuren nicht geschlagen werden.

7.2.2 Berechnung erlaubter Züge

Hebt ein Spieler eine Figur an, ermittelt der Arduino deren Position und berechnet die möglichen Zielfelder. Dabei werden die Bewegungsregeln und die aktuelle Stellung berücksichtigt.

Ein geometrisch möglicher Zug kann trotzdem regelwidrig sein, etwa wenn dadurch der eigene König im Schach stehen würde. Deshalb reicht die Prüfung des Bewegungsmusters allein nicht aus.

Die Zugprüfung ermittelt die ausgewählte Figur und den aktiven Spieler, berechnet Kandidatenzüge und verwirft alle Bewegungen, die gegen die Schachregeln verstossen. Nur die verbleibenden Zielfelder werden als erlaubt angezeigt.

Die LEDs zeigen daher nicht bloss geometrisch erreichbare Felder, sondern die nach der aktuellen Stellung zulässigen Zugmöglichkeiten.

7.2.3 Zugverwaltung und Zugannahme

Die Software verwaltet, welcher Spieler am Zug ist. Zu Beginn der Partie beginnt Weiss. Ein Wechsel des Zugrechts erfolgt erst nach einem vollständig akzeptierten Zug.

Nach Abschluss eines gültigen Zuges wird die interne Stellung angepasst und das Zugrecht an den nächsten Spieler übergeben. Die aktualisierte Stellung bildet die Grundlage für die Prüfung des nächsten Zuges.

Diese Unterscheidung ist bei einem physischen System besonders wichtig. Zwischen dem Anheben und dem Abstellen einer Figur können mehrere Sensorsignale auftreten, ohne dass der Spieler bereits einen gültigen Zug abgeschlossen hat. Die Software muss diese Zwischenzustände korrekt behandeln.

7.3 Schach, Schachmatt und Sonderzüge

Neben den normalen Bewegungsregeln wurde die Arduino-Software um wichtige Sonderregeln und Endbedingungen erweitert. Dadurch können vollständige Schachpartien mit den üblichen Regeln durchgeführt werden.

7.3.1 Erkennung von Schach

Ein König steht im Schach, wenn er von einer gegnerischen Figur angegriffen wird. Zur Erkennung untersucht die Software, ob ein Gegner das Feld des Königs erreichen könnte.

Diese Prüfung ist sowohl nach einem abgeschlossenen Zug als auch bei der Bewertung eines vorgeschlagenen Zuges notwendig. Ein Spieler darf keinen Zug ausführen, durch den der eigene König angegriffen bleibt oder angegriffen wird.

Steht ein Spieler bereits im Schach, werden alle Züge ausgeschlossen, die diese Bedrohung nicht beseitigen. Als gültig gelten nur Bewegungen, die den König schützen, den Angreifer schlagen oder den Angriff anderweitig verhindern.

7.3.2 Schachmatt und Patt

Schachmatt liegt vor, wenn der König angegriffen wird und kein gültiger Zug zur Abwehr mehr vorhanden ist. Die Software erkennt diesen Zustand durch die Kombination aus Schachprüfung und Ermittlung sämtlicher legaler Züge.

Patt liegt vor, wenn der Spieler am Zug keinen legalen Zug mehr ausführen kann, sein König aber nicht im Schach steht. Im Gegensatz zum Schachmatt endet die Partie dabei unentschieden.

7.3.3 Rochade

Die Rochade ist ein Sonderzug, an dem König und Turm beteiligt sind. Sie darf nur unter bestimmten Bedingungen erfolgen, unter anderem wenn die beteiligten Figuren noch nicht gezogen haben und die betreffenden Felder frei und sicher sind.

Die Software speichert deshalb, ob der jeweilige König und Turm bereits bewegt wurden. Zusätzlich kontrolliert sie die erforderlichen Zwischenfelder und berücksichtigt, dass bei einer Rochade zwei Figuren ihre Position ändern.

7.3.4 En passant

En passant ist ein besonderer Bauernschlagzug, der unmittelbar nach einem gegnerischen Doppelschritt möglich sein kann. Die Software muss dafür den vorherigen Zug berücksichtigen und prüfen, ob die erforderliche Stellung vorliegt.

Anders als bei einem gewöhnlichen Schlagzug steht der geschlagene Bauer nicht auf dem Zielfeld des schlagenden Bauern. Deshalb müssen bei der Auswertung der Sensoränderungen und beim Aktualisieren der Brettstellung mehrere Felder berücksichtigt werden.

7.3.5 Bauernumwandlung

Erreicht ein Bauer die gegenüberliegende Grundreihe, muss er umgewandelt werden. Bei der Bauernumwandlung wird der Bauer durch eine andere zulässige Figur ersetzt.

Dies ist für Reed-Schalter anspruchsvoll, da sie lediglich die Feldbelegung und nicht die neu gewählte Figurenart erkennen. Die logische Brettdarstellung muss die Umwandlung deshalb unabhängig vom unveränderten Belegungssignal erfassen.

7.3.6 Bedeutung der erweiterten Schachlogik

Die Umsetzung von Schach, Schachmatt, Patt und Sonderzügen erweitert die Zugprüfung über einfache Bewegungsmuster hinaus. Sie ermöglicht eine regelkonforme Durchführung vollständiger Partien.

Die zusätzlichen Regeln erhöhen die Komplexität des Zusammenspiels von physischer Belegung und logischer Stellung. Insbesondere bei Zügen mit mehreren beteiligten Figuren muss die Software sicherstellen, dass die tatsächliche Brettbelegung mit dem gespeicherten Zustand übereinstimmt.

7.4 LED-Visualisierung

Das LED-System gibt den Spielern direkt am physischen Schachbrett Rückmeldungen. Es unterstützt die Auswahl erlaubter Züge und zeigt besondere Situationen an, ohne dass ständig auf die digitale Oberfläche geschaut werden muss.

Die beiden LED-Matrizen befinden sich seitlich des Schachbretts und stellen die relevanten Feldpositionen visuell dar. Die Anzeige wird vom Arduino gesteuert.

7.4.1 Darstellung des Schachbretts

Jede LED-Position entspricht einem Schachfeld. Die Software ordnet die internen Brettkoordinaten den entsprechenden LED-Positionen zu, damit die Hinweise zum physischen Brett passen.

Die LED-Darstellung verwendet dasselbe Koordinatensystem wie die Sensorabfrage und die interne Schachlogik. Eine konsistente Zuordnung ist erforderlich, damit Start- und Zielfelder korrekt angezeigt werden.

7.4.2 Anzeige erlaubter Züge

Wird eine Figur angehoben, berechnet der Arduino die erlaubten Zielfelder und markiert diese über die LEDs. Die blaue Anzeige dient als Orientierung für den nächsten Zug.

Gerade für weniger erfahrene Spieler erleichtert diese Funktion die Bedienung des physischen Schachbretts. Gleichzeitig lässt sich dadurch kontrollieren, ob die Software die zulässigen Bewegungen korrekt berechnet.

7.4.3 Rückmeldung bei unerlaubten Zügen

Bei einem unerlaubten Zug zeigt das System eine rote Rückmeldung an. Damit wird dem Spieler signalisiert, dass die ausgeführte Bewegung nicht übernommen wurde und korrigiert werden muss.

Die intern gespeicherte Stellung bleibt unverändert, bis die physische Situation wieder korrekt ist und ein gültiger Zug abgeschlossen wurde. So verhindert die Software, dass fehlerhafte Bewegungen die Partie verändern.

7.4.4 Anzeige von KI-Zügen

Im Spieler-gegen-KI-Modus übernimmt die LED-Anzeige eine zusätzliche Aufgabe. Nach der Berechnung durch Stockfish werden die für den KI-Zug vorgesehenen Felder violett markiert.

Der Spieler führt den Zug der KI anschliessend von Hand aus. Der Arduino überprüft anhand der Reed-Schalter, ob die richtige Figur vom vorgesehenen Startfeld zum vorgesehenen Zielfeld bewegt wurde.

Dadurch wird verhindert, dass die digitale Partie unabhängig vom physischen Schachbrett fortschreitet.

7.4.5 Bewertung des LED-Systems

Die LED-Visualisierung überträgt die Entscheidungen der Software unmittelbar auf das physische Brett. Sie verbindet die digitale Zugberechnung mit einer verständlichen Rückmeldung für den Spieler.

Die visuellen Hinweise erleichtern ausserdem die Fehlersuche während der Entwicklung. Anhand der angezeigten Felder lässt sich prüfen, ob Sensorerkennung, Koordinatenzuordnung und Zuglogik zusammenpassen.

7.5 Kapitelzusammenfassung

Die Arduino-Software verbindet die Sensorbelegung mit einer internen Schachstellung und steuert Zugprüfung, Sonderregeln und LED-Rückmeldungen. Die Kommunikation mit Node-RED wird im nächsten Kapitel erläutert.

Die Schachlogik prüft die Züge nach den Schachregeln und berücksichtigt auch Sonderzüge und Spielendbedingungen. Die LED-Matrizen zeigen erlaubte Bewegungen, Fehlversuche und KI-Züge an. Dadurch entsteht eine direkte Verbindung zwischen dem physischen Spiel und der digitalen Verarbeitung.

8 Kommunikation zwischen Arduino und Node-RED

Die Kommunikation zwischen Arduino Mega 2560 und Node-RED ist ein wesentlicher Bestandteil des intelligenten Schachbretts. Sie verbindet die lokale Erkennung und Zugprüfung mit der digitalen Benutzeroberfläche und der Schachengine.

Vom Arduino erkannte physische Züge müssen an Node-RED übertragen werden, damit dort die digitale Brettdarstellung, die Zugliste und die Analyse aktualisiert werden können. Umgekehrt muss Node-RED berechnete KI-Züge an den Arduino senden können.

Besonders wichtig ist, dass die physische und die digitale Schachstellung übereinstimmen. Deshalb muss das Kommunikationsprotokoll zwischen angeforderten und tatsächlich ausgeführten Bewegungen unterscheiden.

8.1 Serielles Kommunikationsprotokoll

Der Arduino Mega 2560 kommuniziert über eine USB-Verbindung mit Node-RED. Dabei wird eine serielle Schnittstelle verwendet, die auf dem Computer als entsprechender Anschluss verfügbar ist.

8.1.1 Kommunikationseinstellungen

Die serielle Schnittstelle arbeitet mit einer Baudrate von 115200. Arduino und Node-RED müssen dieselben Übertragungsparameter verwenden, damit Nachrichten korrekt empfangen werden.

Die Kommunikation erfolgt über zeilenweise übertragene Textnachrichten. Dieses Format lässt sich sowohl auf dem Arduino als auch in Node-RED einfach erzeugen, lesen und während der Entwicklung kontrollieren.

8.1.2 Aufbau der Nachrichten

Das Protokoll verwendet eindeutige Textpräfixe, um verschiedene Nachrichtentypen voneinander zu unterscheiden. Dadurch kann Node-RED erkennen, ob eine Meldung einen Zug, das Anheben einer Figur oder eine Bestätigung betrifft.

Tabelle 8-1: Beispiele für serielle Nachrichten zwischen Arduino und Node-RED

Nachricht	Richtung	Zweck
LIFT: e2	Arduino → Node-RED	Meldet das Anheben einer Figur
PLACE: e4	Arduino → Node-RED	Meldet das Abstellen einer Figur
MOVE: e2 -> e4	Arduino → Node-RED	Meldet einen abgeschlossenen Schachzug
AI_MOVE:d7d5	Node-RED → Arduino	Übermittelt einen berechneten KI-Zug
AI_MOVE_RECEIVED: ...	Arduino → Node-RED	Bestätigt den Empfang eines KI-Zugbefehls
AI_MOVE_COMPLETE	Arduino → Node-RED	Meldet den physischen Abschluss eines KI-Zugs
AI_ERROR: WRONG PIECE LIFTED	Arduino → Node-RED	Meldet eine falsche physische Bewegung
CHECK: WHITE	Arduino → Node-RED	Meldet eine Schachsituation
STALEMATE	Arduino → Node-RED	Meldet ein Patt

8.1.3 Darstellung von Schachzügen

Schachzüge werden mit Start- und Zielfeld dargestellt. Beispielsweise bezeichnet MOVE:e2e4 eine Bewegung von e2 nach e4. Die kompakte Schreibweise eignet sich für die Übertragung und Weiterverarbeitung.

Das Präfix AI_MOVE: kennzeichnet einen von Node-RED übermittelten KI-Zug. Die nachfolgenden Koordinaten bestimmen, welche Bewegung auf dem physischen Brett auszuführen ist.

8.1.4 Vorteile des gewählten Protokolls

Das textbasierte serielle Protokoll ist einfach auf dem Arduino umzusetzen und erleichtert die Fehlersuche. Nachrichten können im seriellen Monitor oder in den Debug-Ausgaben von Node-RED direkt überprüft werden.

Die Verwendung von Textnachrichten garantiert jedoch nicht automatisch, dass jede Meldung korrekt verarbeitet wird. Deshalb müssen Nachrichtenformate geprüft und fehlerhafte oder unerwartete Befehle behandelt werden.

8.2 Bidirektionale Kommunikation

Der Arduino sendet physische Ereignisse und Spielinformationen an Node-RED. In Gegenrichtung erhält er unter anderem Befehle für die von Stockfish berechneten KI-Züge.

8.2.1 Kommunikation vom Arduino zu Node-RED

Während einer Partie überwacht der Arduino fortlaufend die Reed-Schalter. Wird eine Figur angehoben und anschliessend abgestellt, versucht die Software, daraus einen Zug abzuleiten.

Die Software prüft die Zulässigkeit der Bewegung. Nach der Annahme sendet der Arduino die Zugdaten an Node-RED, wo die digitale Spielstellung und die weiteren Anzeigen aktualisiert werden.

8.2.2 Kommunikation von Node-RED zum Arduino

Nach einem gültigen menschlichen Zug im Spieler-gegen-KI-Modus übermittelt Node-RED die aktuelle Stellung an Stockfish. Die Schachengine berechnet daraufhin eine Antwort für Schwarz.

Der Arduino empfängt den KI-Befehl und kontrolliert, ob er zur erwarteten Stellung passt. Anschliessend zeigt er die vorgesehene Bewegung an und wartet auf deren physische Ausführung.

8.2.3 Kommunikationsablauf im PvAI-Modus

Der vollständige Ablauf beginnt mit einem menschlichen Zug auf dem Brett. Nach der Validierung werden die Daten an Node-RED und weiter an Stockfish übertragen, wo der nächste KI-Zug berechnet wird.

Der berechnete Zug gelangt als AI_MOVE-Befehl zurück zum Arduino. Die LEDs markieren die erforderliche Bewegung, die anschliessend von Hand auf dem Brett ausgeführt wird.

Stockfish bewegt keine Figuren direkt. Die Engine bestimmt ausschliesslich den Schachzug; die physische Umsetzung erfolgt durch den Spieler und wird vom Arduino kontrolliert.

8.2.4 Synchronisation des physischen und digitalen Bretts

Eine wesentliche Herausforderung besteht darin, die reale Brettbelegung und den digitalen Spielstand synchron zu halten. Wenn die Software einen KI-Zug bereits als abgeschlossen betrachtet, bevor die Figur tatsächlich bewegt wurde, können unterschiedliche Stellungen entstehen.

Um dies zu verhindern, unterscheidet der Arduino zwischen einem empfangenen KI-Zug und einem physisch bestätigten KI-Zug. Erst nach korrekter Ausführung wird die entsprechende Abschlussmeldung übermittelt.

Dieser Mechanismus ist besonders wichtig, wenn der Spieler eine falsche Figur anhebt, ein falsches Zielfeld wählt oder die Bewegung unterbricht. Solche Situationen dürfen die digitale Stellung nicht unbemerkt verändern.

8.3 Fehlerbehandlung

Zuverlässige Kommunikation erfordert mehr als das reine Senden von Nachrichten. Das System muss eingehende Befehle auf ihre Struktur und Plausibilität prüfen und bei Abweichungen eine passende Rückmeldung liefern.

8.3.1 Prüfung eingehender KI-Befehle

Ein gültiger KI-Befehl beginnt mit AI_MOVE: und enthält die Koordinaten von Start- und Zielfeld. Fehlen Angaben oder ist das Format ungültig, wird der Befehl zurückgewiesen.

In der umgesetzten Spieler-gegen-KI-Konfiguration spielt der Mensch mit Weiss und die Engine mit Schwarz. Die Prüfung eingehender Befehle berücksichtigt deshalb auch die erwartete Spielerfarbe und den aktuellen Spielzustand.

Zu den Diagnosemeldungen gehören beispielsweise AI_MOVE_ERROR: INVALID LENGTH und weitere Hinweise auf ungültige KI-Befehle. Sie erleichtern das Erkennen von Format- und Zustandsfehlern.

8.3.2 Erkennung fehlerhafter physischer Bewegungen

Nach Empfang eines gültigen KI-Befehls zeigt der Arduino die vorgesehene Bewegung über die LEDs an. Danach überwacht er die Reed-Schalter, bis die physische Ausführung bestätigt werden kann.

Ein falsches Anheben kann beispielsweise mit AI_ERROR: WRONG PIECE LIFTED gemeldet werden. Auch eine Bewegung auf ein unerwartetes Zielfeld wird als Fehler behandelt, statt den Zug zu übernehmen.

8.3.3 Behandlung unerlaubter Züge

Unerlaubte Züge können sowohl in normalen Partien als auch im Spieler-gegen-KI-Modus vorkommen. Die Software muss sie erkennen und dem Spieler verständlich anzeigen.

Ein unerlaubter Zug darf weder die akzeptierte interne Stellung verändern noch das Zugrecht wechseln. Die betroffene Bewegung muss zuerst korrigiert werden.

8.3.4 Fehlermeldungen und Diagnose

Während der Entwicklung diente die serielle Schnittstelle auch als Diagnosewerkzeug. Die ausgegebenen Meldungen halfen dabei, einzelne Schritte des Ablaufs nachzuvollziehen und Fehler zwischen Sensorik, Zuglogik und Node-RED einzugrenzen.

Tabelle 8-2: Ausgewählte Fehlerbedingungen und ihre Behandlung im Arduino-Programm

Fehlerbedingung	Arduino-Rückmeldung	Auswirkung auf das Spiel
Ungültige Länge des KI-Befehls	AI_MOVE_ERROR: INVALID LENGTH	Befehl zurückgewiesen
Ungültige Koordinaten	AI_MOVE_ERROR: INVALID SQUARE	Befehl zurückgewiesen
KI-Befehl bei falschem Zugrecht	AI_MOVE_ERROR: NOT BLACK TURN	Befehl zurückgewiesen
Unerlaubter KI-Zug	AI_MOVE_ERROR: MOVE NOT LEGAL ON ARDUINO BOARD	Befehl zurückgewiesen
Falsche Figur angehoben	AI_ERROR: WRONG PIECE LIFTED	KI-Zug nicht abgeschlossen
Falsches Zielfeld	AI_ERROR: WRONG DESTINATION	KI-Zug nicht abgeschlossen
Unerlaubter Spielerzug	ILLEGAL MOVE	Zug zurückgewiesen

8.3.5 Grenzen des Kommunikationssystems

Obwohl das Protokoll die benötigten Funktionen unterstützt, besitzt es Grenzen. Die Übertragung erfolgt über eine direkte USB-Verbindung und verwendet ein vergleichsweise einfaches textbasiertes Nachrichtenformat.

Wird die USB-Verbindung unterbrochen, kann die Kommunikation zwischen Arduino und Node-RED ausfallen. Für eine spätere Weiterentwicklung wären zusätzliche Verfahren zur Wiederverbindung und Synchronisierung sinnvoll.

Für diese Diplomarbeit stellt der gewählte Ansatz einen praktikablen Kompromiss zwischen einfacher Umsetzung, Transparenz und benötigtem Funktionsumfang dar.

8.4 Kapitelzusammenfassung

Das serielle Protokoll ermöglicht den Austausch erkannter Züge und KI-Anweisungen. Entscheidend ist, dass physische Ausführung und digitaler Spielzustand übereinstimmen.

Der Arduino überprüft eingehende Befehle und zeigt erforderliche Bewegungen über die LEDs an. KI-Züge werden erst nach ihrer physischen Ausführung bestätigt, wodurch die Synchronisierung unterstützt wird.

Das Protokoll liefert während der Entwicklung ausserdem wichtige Diagnoseinformationen. Zusammen mit den Prüfmechanismen bildet es die Grundlage für den zuverlässigen Austausch zwischen Hardware und PC-Anwendung.

9 Node-RED und digitale Benutzeroberfläche

Node-RED bildet die zentrale Softwareschnittstelle zwischen dem Arduino Mega 2560, der digitalen Benutzeroberfläche und Stockfish. Die Anwendung verarbeitet serielle Daten und stellt die Informationen für den Spieler bereit.

Die Anwendung baut auf einer früheren Transferarbeit auf und wurde während der Diplomarbeit erweitert. Dabei wurden insbesondere die Spielmodi, die Darstellung der Zugliste und die Integration der Schachengine weiterentwickelt.

9.1 Datenverarbeitung

9.1.1 Aufbau der Node-RED-Anwendung

Node-RED verwendet miteinander verbundene Nodes zum Empfangen serieller Nachrichten, zur Verarbeitung der Spieldaten und zur Aktualisierung der Benutzeroberfläche. Diese modulare Struktur erleichtert die Aufteilung in einzelne Funktionen.

Die wichtigsten Verarbeitungspfade betreffen eingehende Arduino-Nachrichten, die aktuelle Schachstellung, die Zugliste, die Analyse durch Stockfish und die Rückübertragung von KI-Zügen.

9.1.2 Verarbeitung serieller Nachrichten

Der Arduino übermittelt zeilenweise Textnachrichten über die serielle USB-Schnittstelle. Node-RED empfängt diese Daten und wertet die enthaltenen Präfixe und Koordinaten aus.

Während der Entwicklung führte ein Fehler in der Auswertungslogik dazu, dass eingehende Informationen teilweise falsch interpretiert wurden. Durch die Kontrolle der Daten vor und nach der Verarbeitung konnte die Ursache eingegrenzt werden.

9.1.3 Verwaltung der Spielinformationen

Nach einem akzeptierten Zug aktualisiert Node-RED die Informationen, die für die digitale Brettdarstellung und die weiteren Anzeigen benötigt werden. Der Zug wird dabei dem aktuellen Spielzustand zugeordnet.

Im Spieler-gegen-KI-Modus unterscheidet die Anwendung zwischen einem von Stockfish vorgeschlagenen Zug und dessen tatsächlicher Ausführung auf dem Brett. Diese Trennung verhindert eine verfrühte Aktualisierung.

9.1.4 Verteilung der Informationen

Ein abgeschlossener Zug kann gleichzeitig die digitale Brettdarstellung, die Zugliste, die Zugbewertung und die weitere Kommunikation beeinflussen. Deshalb müssen diese Funktionen auf denselben akzeptierten Spielzustand zugreifen.

9.1.5 Verarbeitung der Stockfish-Ergebnisse

Node-RED empfängt und verarbeitet die Ergebnisse der Stockfish-Engine. Je nach Anwendungsfall werden daraus die Bewertung einer Stellung, die Qualität eines Zuges oder die nächste Bewegung der KI abgeleitet.

9.2 Dashboard

9.2.1 Aufbau des Dashboards

Das Dashboard bietet eine digitale Übersicht über die Schachpartie und wichtige Einstellungen. Es ergänzt das physische Schachbrett um Informationen, die über die LEDs allein nicht sinnvoll dargestellt werden können.

Tabelle 9-1: Übersicht über die Funktionen des digitalen Dashboards

Dashboard-Element	Funktion
Digitales Schachbrett	Zeigt die aktuelle Schachstellung
Spieler am Zug	Zeigt das aktuelle Zugrecht
Letzter Zug	Zeigt den zuletzt abgeschlossenen Zug
Zugzähler	Zeigt den Spielfortschritt
Zugliste	Speichert abgeschlossene Züge in Reihenfolge
Zugbewertung	Zeigt die mit Stockfish ermittelte Qualitätskategorie
Spielmodus	Wählt Spieler gegen Spieler oder Spieler gegen KI
KI-Schwierigkeit	Passt die Spielstärke des Computergegners an
Spielstatus	Zeigt Informationen zur aktuellen Spielsituation

9.2.2 Digitale Schachbrettdarstellung

Das digitale Schachbrett visualisiert die akzeptierte Stellung der Figuren. Es dient als Kontrolle des aktuellen Spielzustands und ergänzt die physische Darstellung.

9.2.3 Zugverlauf

Die Zugliste speichert die Reihenfolge der akzeptierten Schachzüge und ermöglicht es, den bisherigen Spielverlauf nachzuvollziehen. Sie wird nach abgeschlossenen Zügen aktualisiert.

9.2.4 Stockfish-Zugbewertung

Nach der Analyse eines Zuges zeigt Node-RED die berechnete Qualitätskategorie im Dashboard an. Dadurch erhält der Spieler eine verständliche Rückmeldung zu einzelnen Entscheidungen.

9.2.5 Auswahl des Spielmodus

Das Dashboard unterstützt sowohl Spieler gegen Spieler als auch Spieler gegen KI. Im ersten Modus führen beide Personen ihre Züge physisch aus; im zweiten Modus werden die Züge des Computers zusätzlich über Stockfish berechnet.

9.2.6 Auswahl des KI-Schwierigkeitsgrads

Die Spielstärke des KI-Gegners kann über Node-RED eingestellt werden. Diese Konfiguration ist von der Sensorabfrage und der Zugvalidierung auf dem Arduino getrennt.

9.2.7 Bedeutung des Dashboards

Dashboard und LED-Matrizen erfüllen unterschiedliche, sich ergänzende Aufgaben. Die LEDs unterstützen die direkte Bedienung am Brett; das Dashboard zeigt den digitalen Spielstand, den Verlauf und zusätzliche Analyseinformationen.

9.3 Dokumentierte Entwicklungsprobleme

Bei der Integration der Node-RED-Anwendung traten verschiedene Probleme auf. Dank der modularen Flow-Struktur konnten Zwischenergebnisse kontrolliert und Fehler bestimmten Verarbeitungsschritten zugeordnet werden.

9.3.1 Fehlerhafte Interpretation serieller Daten

Eingehende serielle Nachrichten wurden zunächst in einem Teil des Node-RED-Flows falsch interpretiert. Die Kontrolle der empfangenen Daten vor und nach dem Parsen zeigte, dass die Ursache in der Weiterverarbeitung lag.

Dieses Problem verdeutlichte, dass eine erfolgreiche Übertragung noch keine korrekte Interpretation garantiert. Durch die Prüfung der Zwischennachrichten konnte der Fehler behoben werden, ohne unnötig andere Systemteile zu verändern.

9.3.2 Problem bei der Zuglistenanzeige

Die Zugliste wurde zunächst mit einer ui_table-Komponente angezeigt. Obwohl die zugrunde liegenden Daten vorhanden waren, stellte die Tabelle nicht alle erwarteten Informationen korrekt dar.

Deshalb wurde die Anzeige von ui_table auf eine Umsetzung mit ui_template umgestellt. Dadurch liessen sich Aufbau und Darstellung gezielter steuern; die Zugliste wurde anschliessend korrekt angezeigt.

9.3.3 Falscher Feldname bei der Zugbewertung

Ein weiterer Fehler betraf die Anzeige der Zugqualität. Die Bewertungsfunktion speicherte die Kategorie unter dem Eigenschaftsnamen rating, während die Dashboard-Anzeige einen anderen Feldnamen erwartete.

Die Eigenschaftsnamen wurden zwischen Berechnung und Anzeige vereinheitlicht. Eine grundlegende Änderung der Bewertung war nicht nötig. Das Beispiel zeigt die Bedeutung einheitlicher Datenstrukturen zwischen den Nodes.

9.3.4 Bewertung der Fehlersuche

Die dokumentierten Fälle zeigen eine systematische Fehlersuche: Soll- und Ist-Daten werden an aufeinanderfolgenden Stellen eines Flows verglichen, die Abweichung lokalisiert und anschliessend gezielt korrigiert.

Tabelle 9-2: Übersicht über dokumentierte Node-RED-Probleme und ihre Lösungen

Problem	Ermittelte Ursache	Umgesetzte Lösung
Fehlerhafte Interpretation serieller Daten	Parsing-Logik interpretierte empfangene Nachrichten falsch	Nachrichtenverarbeitung korrigiert
Unvollständige Zuglistenanzeige	ui_table stellte die erwarteten Daten nicht korrekt dar	Anzeige durch ui_template ersetzt
Fehlende Zugqualitätskategorie	Unterschiedliche Eigenschaftsnamen: rating und quality	Feldnamen vereinheitlicht

Die Beispiele betreffen die Verarbeitung der Kommunikation, die Darstellung der Benutzeroberfläche und die Konsistenz von Datenstrukturen. Die Möglichkeit, Zwischenwerte in Node-RED einzusehen, erwies sich bei der Fehlersuche als besonders hilfreich.

Kapitelzusammenfassung

Node-RED verarbeitet die seriellen Meldungen, führt Spielinformationen zusammen und stellt sie im Dashboard dar. Die beschriebenen Fehler wurden bei der Integration behoben.

Während der Entwicklung wurden Probleme bei der Interpretation serieller Nachrichten, bei der Anzeige der Zugliste und bei uneinheitlichen Feldnamen der Zugbewertung erkannt und behoben. Diese Erfahrungen verdeutlichen den Nutzen einer modularen und überprüfbaren Datenverarbeitung.

10 Stockfish-Integration und Spielanalyse

Die Integration der Schachengine Stockfish gehört zu den wichtigsten Softwareerweiterungen des intelligenten Schachbretts. Während der Arduino die physischen Bewegungen erkennt und die Schachregeln prüft, übernimmt Stockfish die Analyse von Stellungen und die Berechnung von KI-Zügen.

Stockfish läuft lokal auf dem Computer und kommuniziert über das Universal Chess Interface (UCI). Für die Integration müssen die korrekte Stellung übertragen, die Berechnung gestartet und die Antworten der Engine richtig interpretiert werden.

10.1 UCI-Anbindung

Stockfish ist eine Schachengine und keine vollständige Benutzeroberfläche. Das UCI-Protokoll bietet ein standardisiertes textbasiertes Verfahren, mit dem eine externe Anwendung die Engine initialisieren, konfigurieren und Berechnungen anfordern kann.

10.1.1 Softwarearchitektur

Die drei Hauptkomponenten besitzen klar getrennte Aufgaben: Der Arduino steuert das physische Brett, Node-RED verarbeitet Spielzustand, Dashboard und Kommunikation, und Stockfish führt die Schachberechnungen aus.

Nach einem akzeptierten Zug gelangen die Informationen vom physischen Brett zu Node-RED. Dort wird die aktuelle Stellung an Stockfish übergeben. Das Analyseergebnis wird anschliessend für die Zugbewertung oder zur Berechnung eines KI-Zuges verwendet.

10.1.2 Initialisierung der Engine

Eine UCI-Sitzung beginnt mit dem Befehl uci. Stockfish meldet seine Identität und verfügbaren Optionen und bestätigt den Abschluss mit uciok. Danach können unter anderem isready und weitere Befehle zur Konfiguration verwendet werden.

10.1.3 Übertragung einer Schachstellung

Vor einer Analyse muss die Engine eine Stellung erhalten, die dem akzeptierten Spielzustand entspricht. UCI ermöglicht es, die Stellung aus der normalen Anfangsstellung und einer anschliessenden Zugfolge oder aus einer anderen definierten Position aufzubauen.

Eine konsistente Stellung ist entscheidend. Unterscheiden sich die Zugliste oder der aktive Spieler zwischen Node-RED, Stockfish und Arduino, könnte die Engine eine falsche Stellung bewerten oder einen unpassenden Zug berechnen.

10.1.4 Start der Analyse und Auslesen der Ergebnisse

Der UCI-Befehl go startet die Berechnung. Beispielsweise fordert go depth 12 eine Suche bis zu einer bestimmten Tiefe an. Währenddessen kann Stockfish mit info score Bewertungsinformationen und mit bestmove einen ausgewählten Zug zurückgeben.

Node-RED muss zwischen Bewertungsdaten und dem ausgewählten Zug unterscheiden. Ein Centipawn-Wert wird für die Zugqualität verwendet; die bestmove-Antwort ist dagegen für die Steuerung des KI-Gegners relevant.

10.2 Zugbewertung

Ein wesentliches Ziel der digitalen Anwendung ist eine verständliche Rückmeldung zur Qualität der Spielerentscheidungen. Stockfish bewertet Schachstellungen numerisch; diese Werte werden in der Anwendung in übersichtliche Kategorien umgewandelt.

10.2.1 Centipawn-Bewertungen

Ein Centipawn entspricht einem Hundertstel Bauernwert. Wird die Bewertung aus Sicht von Weiss angegeben, begünstigt ein positiver Wert Weiss und ein negativer Wert Schwarz. Ein Wert von +150 entspricht damit ungefähr einem Vorteil von 1,5 Bauernwerten für Weiss.

10.2.2 Centipawn-Verlust

Der Centipawn-Verlust vergleicht die Bewertung der stärksten verfügbaren Fortsetzung mit der Bewertung des tatsächlich gespielten Zuges. Ein geringer Verlust bedeutet, dass der Zug nahe an der besten berechneten Möglichkeit liegt.

Die vereinfachte Berechnung lautet CPL = max(0, E_best − E_played). Beide Bewertungen müssen dabei aus Sicht des Spielers angegeben werden, dessen Zug untersucht wird. Diese Perspektive ist für eine korrekte Berechnung entscheidend.

Wird beispielsweise die beste Fortsetzung für Weiss mit +120 Centipawns bewertet, der gespielte Zug aber nur mit +40, beträgt der Centipawn-Verlust 80. Für einen sinnvollen Vergleich sollten die Bewertungen unter vergleichbaren Analysebedingungen entstehen.

10.2.3 Projektspezifische Klassifizierung

Damit die numerischen Ergebnisse verständlich bleiben, unterscheidet das Projekt fünf Kategorien der Zugqualität. Die Schwellenwerte wurden für dieses intelligente Schachbrett festgelegt und stellen keine allgemeingültige offizielle Stockfish-Klassifizierung dar.

Tabelle 10-1: Projektspezifische Kategorien der Zugqualität

Centipawn-Verlust	Kategorie	Interpretation
0–10	Bester Zug	Sehr nahe an der stärksten verfügbaren Fortsetzung
11–30	Ausgezeichnet	Sehr geringer Bewertungsverlust
31–70	Gut	Mässiger Bewertungsverlust
71–150	Fehler	Deutliche Verschlechterung der Stellung
Über 150	Schwerer Fehler	Erhebliche Verschlechterung der Stellung

10.2.4 Beispiel und Grenzen

Wird die beste Fortsetzung beispielsweise mit +90 und der gespielte Zug mit +35 Centipawns bewertet, ergibt sich ein Verlust von 55. Dieser wird gemäss den projektspezifischen Grenzwerten als Good eingestuft.

Die Bewertung hängt von der Suchtiefe und Berechnungsdauer ab. Eine tiefere Analyse kann eine stärkere Fortsetzung finden und den Wert verändern. Mattstellungen und andere besondere Situationen erfordern bei der Interpretation zusätzliche Vorsicht.

10.3 Spieler-gegen-KI-Modus

Im Spieler-gegen-KI-Modus führt der Mensch die weissen Figuren, während Stockfish Schwarz übernimmt. Das System bewegt die Figuren nicht mechanisch. Der Spieler führt die von der Engine berechneten Züge von Hand aus.

10.3.1 Spielablauf

Der menschliche Spieler hebt eine weisse Figur an und stellt sie auf einem erlaubten Zielfeld ab. Der Arduino erkennt und prüft den Zug, aktualisiert seine interne Stellung und übermittelt die Informationen an Node-RED.

Nachdem Stockfish einen schwarzen Zug berechnet hat, liest Node-RED die Koordinaten aus und sendet einen AI_MOVE:-Befehl an den Arduino. Dieser überprüft den Befehl anhand der erwarteten Stellung.

10.3.2 Führung und Bestätigung des KI-Zugs

Die Engine-Antwort bestmove g8f6 kann beispielsweise in AI_MOVE:g8f6 umgewandelt werden. Der Arduino markiert die vorgesehene Bewegung violett. Danach führt der Spieler den KI-Zug physisch aus.

Der Empfang des KI-Befehls bedeutet noch nicht, dass der Zug abgeschlossen ist. Der Arduino überwacht weiterhin die Reed-Schalter und prüft, ob die richtige Figur vom vorgesehenen Start- zum Zielfeld bewegt wurde.

Diese Bestätigung verhindert, dass die digitale Partie der physischen Stellung vorausläuft. Das ist besonders wichtig, weil Stockfish seine Berechnung unabhängig von der tatsächlichen Bewegung der Figuren durchführt.

10.3.3 Konsistenz der Schachregeln

Arduino und Stockfish müssen mit übereinstimmenden Schachstellungen arbeiten. Sonderzüge wie Rochade, en passant und Bauernumwandlung erfordern besondere Aufmerksamkeit, weil dabei mehrere Feldbelegungen oder Figureninformationen betroffen sein können.

10.4 KI-Schwierigkeitsgrad

Im Node-RED-Dashboard kann die Spielstärke des Computergegners eingestellt werden. Diese Konfiguration erfolgt auf der PC-Seite; der Arduino bleibt für Sensorerkennung und Zugprüfung zuständig.

10.4.1 Konfiguration des Schwierigkeitsgrads

Stockfish unterstützt Engine-Optionen über UCI. Je nach Version und Konfiguration kann die Spielstärke beispielsweise mit UCI_LimitStrength und UCI_Elo begrenzt werden. Die tatsächlich verfügbaren Optionen müssen zur verwendeten Version passen.

10.4.2 Konfiguierte Schwierigkeitsstufen

Im Node-RED-Dashboard stehen fünf Schwierigkeitsstufen zur Auswahl. Diese reichen von «Anfänger» bis «Experte» und sind den Stockfish-Elo-Zielwerten 800, 1200, 1600, 2000 und 2400 zugeordnet. Die Auswahl wird über die UCI-Konfiguration an Stockfish übermittelt. Die angegebenen Elo-Werte dienen als Zielwerte und stellen keine unabhängig gemessene Spielstärke dar.

Tabelle 10-2: Vorgeschlagene KI-Schwierigkeitsstufen und zugehörige Elo-Zielwerte

Schwierigkeit	Stockfish-Elo-Zielwert	Beschreibung
Anfänger	800	Für Spieler, die grundlegende Regeln und Strategien lernen
Einfach	1200	Gut bewältigbare Herausforderung für Gelegenheitsspieler
Mittel	1600	Für Spieler mit etwas Schacherfahrung
Schwer	2000	Stärkerer Gegner, der ein gutes Stellungsverständnis erfordert
Experte	2400	Anspruchsvoller Gegner für erfahrene Spieler

Die fünf Schwierigkeitskategorien sollen die Auswahl vereinfachen. Höhere Elo-Zielwerte führen im Allgemeinen zu einem stärkeren Gegner; sie garantieren jedoch keine exakt reproduzierbare Spielstärke in jeder einzelnen Partie.

10.4.3 Trennung von Schwierigkeit und Analyse

Die Spielstärke des KI-Gegners wird von der Einstellung für die Bewertung menschlicher Züge getrennt werden. Eine absichtlich geschwächte Engine kann bei der Analyse weniger zuverlässige Vergleichswerte liefern als eine stärkere Konfiguration.

10.4.4 Grenzen

Auch bei reduzierter Schwierigkeit kann Stockfish gelegentlich sehr gute Züge finden. Umgekehrt kann eine starke Einstellung einen einfachen Zug wählen, wenn die Stellung dies verlangt. Die Schwierigkeitsstufe beschreibt daher keine feste Qualität jedes einzelnen Zuges.

Kapitelzusammenfassung

Stockfish liefert über UCI Zugvorschläge und Bewertungen. Im PvAI-Modus werden die vorgeschlagenen Züge am physischen Brett ausgeführt und überprüft.

11 Verifikation und Validierung

Die Verifikation umfasst Sensorik, Schachlogik, LED-Anzeigen, Kommunikation und Stockfish-Integration. Dieses Kapitel dokumentiert die Einzelprüfungen und Tests vollständiger Partien.

Bei der Verifikation wird überprüft, ob die entwickelten Funktionen die festgelegten technischen Anforderungen erfüllen. Die Validierung untersucht hingegen, ob das Gesamtsystem seinen vorgesehenen Zweck erfüllt und in der Praxis sinnvoll eingesetzt werden kann.

Zunächst wurden Hardware und Software einzeln überprüft, danach ihr Zusammenspiel in vollständigen Partien.

Ein Schwerpunkt war die Sensormatrix, da jede physische Zugerkennung von ihr abhängt.

11.1 Teststrategie

Die Teststrategie wurde so aufgebaut, dass Fehler möglichst früh erkannt und den entsprechenden Komponenten zugeordnet werden konnten. Anstatt das gesamte System erst nach Abschluss der Entwicklung zu überprüfen, wurden die einzelnen Funktionen bereits während ihrer Implementierung getestet.

Dieses Vorgehen war notwendig, da ein fehlerhaft erkannter Schachzug unterschiedliche Ursachen haben kann. Beispielsweise kann ein Reed-Sensor nicht korrekt schalten, eine elektrische Verbindung unterbrochen sein oder die Software eine Sensoränderung falsch interpretieren. Durch die getrennte Überprüfung der Hardware und Software konnten solche Fehler gezielter untersucht werden.

11.1.1 Aufbau der Teststrategie

Die Überprüfung des intelligenten Schachbretts wurde in vier Bereiche unterteilt.

Tabelle 11-1: Übersicht der Testbereiche

Testbereich	Überprüfte Komponenten	Ziel
Hardware	Reed-Sensoren, Dioden, Verdrahtung und LED-Matrizen	Zuverlässige elektrische und mechanische Funktion
Arduino-Software	Sensorabfrage, Zugerkennung, Schachregeln und LED-Steuerung	Korrekte Verarbeitung der physischen Spielzüge
Kommunikation und Softwareintegration	Arduino, Node-RED und Stockfish	Korrekte Datenübertragung und Verarbeitung
Gesamtsystem	Vollständiges Schachbrett im PvP- und PvAI-Modus	Überprüfung des Zusammenspiels aller Komponenten

Die einzelnen Testbereiche bauen aufeinander auf. Bevor die Zugerkennung zuverlässig überprüft werden konnte, musste sichergestellt werden, dass die Reed-Sensoren korrekt funktionieren. Ebenso war für den Spielmodus gegen Stockfish eine funktionierende Kommunikation zwischen Arduino und Node-RED erforderlich. Diese schrittweise Vorgehensweise erleichterte die Fehlersuche und verhinderte, dass Probleme einzelner Komponenten mit Fehlern des Gesamtsystems verwechselt wurden.

11.1.2 Durchführung der Hardwaretests

Die Reed-Sensor-Matrix stellte einen besonders wichtigen Teil der Hardwareprüfung dar. Das Schachbrett besitzt insgesamt 64 Reed-Sensoren, die den jeweiligen Feldern zugeordnet sind. Durch die Magnete in den Schachfiguren wird erkannt, ob ein Feld besetzt ist.

Da die Software aus den Änderungen der Feldbelegung das Anheben und Absetzen der Figuren ableitet, können fehlerhafte Sensorsignale unmittelbar zu einer falschen Zugerkennung führen. Aus diesem Grund wurden die Sensoren nicht nur einmalig überprüft, sondern über einen längeren Zeitraum wiederholt betätigt. Zusätzlich wurden mechanische Belastungstests durchgeführt. Damit sollte festgestellt werden, ob die elektrischen Verbindungen auch dann zuverlässig funktionieren, wenn das Schachbrett bewegt oder erschüttert wird.

11.1.3 Durchführung der Softwaretests

Die Softwaretests wurden parallel zur Entwicklung durchgeführt. Zunächst wurden die grundlegenden Funktionen überprüft. Dazu gehörten die Erkennung angehobener und abgesetzter Figuren, die Berechnung legaler Zugmöglichkeiten und die Ansteuerung der LEDs. Anschliessend wurden die verschiedenen Schachfiguren und deren Bewegungsregeln getestet.

Im weiteren Verlauf kamen die besonderen Schachregeln wie Rochade, En passant und Bauernumwandlung sowie die Erkennung von Schach, Schachmatt und Patt hinzu. Nach der Integration von Node-RED und Stockfish wurden die Datenübertragung, die digitale Benutzeroberfläche, die Zugbewertung und der Spielmodus gegen die künstliche Intelligenz überprüft. Die Funktionsfähigkeit wurde sowohl anhand gezielt ausgeführter Spielzüge als auch durch vollständige Schachpartien kontrolliert.

11.2 Durchgeführte Tests und Ergebnisse

In diesem Abschnitt werden die tatsächlich durchgeführten Prüfungen und deren Ergebnisse beschrieben. Die Hardware wurde durch wiederholte Sensorbetätigungen und mechanische Belastungstests überprüft. Die Software wurde anhand der implementierten Schachfunktionen sowie durch praktische Spielpartien getestet.

11.2.1 Hardwaretests

Nach der Fertigstellung der Reed-Sensor-Matrix wurden sämtliche 64 Sensorpositionen einzeln überprüft. Während der Entwicklung waren zunächst mehrere unzuverlässige elektrische Verbindungen festgestellt worden. Diese wurden durch Nachlöten und zusätzliche mechanische Fixierung der betroffenen Verbindungen verbessert. Nach den Korrekturen funktionierten alle 64 Sensorpositionen.

Um die Zuverlässigkeit über einen längeren Zeitraum zu überprüfen, wurde anschliessend ein zehntägiger Funktionstest durchgeführt. Dabei wurde jeder der 64 Sensoren zweimal pro Tag betätigt. Die Anzahl der Sensorbetätigungen ergibt sich aus folgender Berechnung: $64 \times 2 \times 10 = 1280$. Somit wurden während der zehn Tage insgesamt 1280 Sensorbetätigungen durchgeführt.

Nach Abschluss dieser Testperiode wurde das gesamte Schachbrett einem mechanischen Belastungstest unterzogen. Dazu wurde das Schachbrett bewegt, geschüttelt und umgedreht. Mit diesem Test sollte überprüft werden, ob die elektrischen Verbindungen auch bei mechanischer Beanspruchung zuverlässig bleiben. Bei diesem ersten Belastungstest funktionierten 11 Sensorpositionen anschliessend nicht mehr korrekt.

Die betroffenen Stellen wurden untersucht und die fehlerhaften Verbindungen repariert. Danach wurden erneut sämtliche 64 Sensorpositionen überprüft. Alle Sensoren funktionierten wieder ordnungsgemäss. Um die Wirksamkeit der Reparaturen zu kontrollieren, wurde ein zweiter mechanischer Belastungstest durchgeführt. Bei diesem abschliessenden Test traten keine weiteren Sensorausfälle auf.

Tabelle 11-2: Ergebnisse der Hardwaretests

Prüfung	Durchführung	Ergebnis
Erste Funktionskontrolle	Überprüfung aller 64 Sensorpositionen	Alle 64 funktionierten
Zehntägiger Funktionstest	Täglich zwei Betätigungen pro Sensor	1280 Betätigungen durchgeführt
Erster mechanischer Belastungstest	Schachbrett bewegen, schütteln und umdrehen	11 Sensorpositionen ausgefallen
Reparatur und Nachkontrolle	Verbindungen reparieren und alle Sensoren überprüfen	Alle 64 funktionierten
Zweiter mechanischer Belastungstest	Erneute mechanische Beanspruchung	Bestanden, keine weiteren Ausfälle

Die Ergebnisse zeigen, dass eine einmalige Funktionskontrolle nicht ausreichte, um sämtliche Schwachstellen der Konstruktion zu erkennen. Obwohl die Sensoren nach den ersten Reparaturen korrekt funktionierten, konnten durch die mechanische Belastung weitere Schwachstellen aufgedeckt werden. Erst nach der erneuten Reparatur und dem erfolgreichen abschliessenden Belastungstest konnte die Konstruktion unter den angewendeten Testbedingungen als ausreichend stabil beurteilt werden.

Die Ergebnisse beziehen sich auf die durchgeführten manuellen Belastungstests. Eine standardisierte Vibrationsprüfung oder eine Langzeituntersuchung über mehrere Monate wurde nicht durchgeführt.

11.2.2 Softwaretests

Nach der Überprüfung der Hardware wurden die Funktionen der Arduino-Software sowie die Integration mit Node-RED und Stockfish getestet. Ziel war es, sicherzustellen, dass die physischen Spielzüge korrekt erkannt, die Schachregeln eingehalten und die entsprechenden Informationen zuverlässig verarbeitet werden.

Die Software wurde während der Entwicklung schrittweise erweitert und jeweils nach der Implementierung neuer Funktionen überprüft. Neben gezielten Tests einzelner Schachregeln wurden auch vollständige Schachpartien im Spieler-gegen-Spieler-Modus (PvP) und im Spieler-gegen-KI-Modus (PvAI) durchgeführt. Dabei konnten die implementierten Funktionen erfolgreich überprüft werden.

11.2.2.1 Überprüfung der Zugerkennung

Zunächst wurde getestet, ob der Arduino das Anheben und Absetzen von Schachfiguren korrekt erkennt. Dazu wurden Figuren von verschiedenen Feldern angehoben und auf andere Felder gesetzt. Die Software musste dabei das Ausgangsfeld und das Zielfeld korrekt bestimmen.

Zusätzlich wurde überprüft, ob die LED-Matrizen nach dem Anheben einer Figur die entsprechenden legalen Zugmöglichkeiten anzeigen. Bei einem gültigen Spielzug musste der Arduino die interne Brettstellung aktualisieren und den Zug akzeptieren. Bei einem ungültigen Spielzug durfte die Spielstellung hingegen nicht verändert werden. Diese Funktionen wurden erfolgreich getestet.

11.2.2.2 Überprüfung der Schachregeln

Ein weiterer Schwerpunkt lag auf der korrekten Umsetzung der Schachregeln. Dazu wurden die Bewegungsmöglichkeiten sämtlicher sechs Figurenarten getestet. Neben den normalen Bewegungen

wurden auch Situationen überprüft, in denen ein Zug aufgrund einer blockierten Zugbahn, einer eigenen Figur auf dem Zielfeld oder einer Gefährdung des eigenen Königs nicht erlaubt ist.

Anschliessend wurden die besonderen Schachregeln gezielt getestet. Dazu gehörten die Rochade, En passant und die Bauernumwandlung. Ausserdem wurde überprüft, ob die Software Schachstellungen sowie die Spielendbedingungen Schachmatt und Patt korrekt erkennt. Alle genannten Funktionen wurden erfolgreich getestet.

Tabelle 11-3: Ergebnisse der Überprüfung der Schachregeln

Testbereich	Durchgeführter Test	Ergebnis
Bauern	Normale Züge, Doppelschritt und Schlagen	Bestanden
Türme	Horizontale und vertikale Bewegungen	Bestanden
Springer	L-förmige Bewegungen und Überspringen von Figuren	Bestanden
Läufer	Diagonale Bewegungen	Bestanden
Dame	Horizontale, vertikale und diagonale Bewegungen	Bestanden
König	Normale Königszüge und Vermeidung unerlaubter Züge	Bestanden
Ungültige Spielzüge	Versuch nicht erlaubter Bewegungen	Bestanden
Rochade	Durchführung einer erlaubten Rochade	Bestanden
En passant	Durchführung des besonderen Bauernschlags	Bestanden
Bauernumwandlung	Umwandlung eines Bauern	Bestanden
Schach	Erkennung einer Schachstellung	Bestanden
Schachmatt	Erkennung einer Schachmattstellung	Bestanden
Patt	Erkennung einer Pattstellung	Bestanden

Die erfolgreichen Tests bestätigten, dass die implementierten Bewegungsregeln und Sonderfunktionen im getesteten Umfang korrekt verarbeitet wurden. Die Prüfung dieser Funktionen war besonders wichtig, da die Software nicht nur geometrisch mögliche Bewegungen erkennen, sondern auch die tatsächlichen Schachregeln berücksichtigen muss.

11.2.2.3 Überprüfung der LED-Anzeigen

Die beiden seitlich montierten WS2812B-LED-Matrizen wurden ebenfalls auf ihre korrekte Funktion überprüft. Dabei wurde getestet, ob nach dem Anheben einer Figur die möglichen legalen Zielfelder angezeigt werden. Zusätzlich wurde überprüft, ob die Software bei einem ungültigen Spielzug eine entsprechende rote LED-Rückmeldung ausgibt.

Im PvAI-Modus musste die LED-Anzeige den von Stockfish berechneten Spielzug darstellen. Die verschiedenen LED-Funktionen wurden erfolgreich getestet. Dadurch konnte bestätigt werden, dass die LED-Matrizen die vorgesehenen Informationen während des Spiels anzeigen.

11.2.2.4 Überprüfung der Kommunikation und Zugbewertung

Nach der Integration von Node-RED wurde überprüft, ob die vom Arduino erkannten Spielzüge korrekt übertragen werden. Die empfangenen Informationen mussten im digitalen Schachbrett dargestellt und in der Zugliste gespeichert werden. Zusätzlich wurde die Integration von Stockfish getestet.

Dabei wurde überprüft, ob die Schachengine die aktuelle Spielstellung auswertet und Node-RED die erhaltenen Ergebnisse verarbeitet. Die daraus berechneten Zugbewertungen mussten anschliessend den festgelegten Kategorien Best Move, Excellent, Good, Mistake oder Terrible zugeordnet werden.

Die Datenübertragung, die Aktualisierung der digitalen Anzeige und die Verarbeitung der Zugbewertungen wurden erfolgreich getestet. Während der Entwicklung festgestellte Probleme bei der Verarbeitung und Darstellung einzelner Daten wurden korrigiert. Diese Fehler und ihre Behebung sind in Kapitel 9 beschrieben.

11.2.2.5 Überprüfung des Spieler-gegen-KI-Modus

Der Spieler-gegen-KI-Modus wurde durch praktische Schachpartien getestet. Dabei spielte der Benutzer mit den weissen Figuren gegen Stockfish mit den schwarzen Figuren. Nach einem gültigen weissen Spielzug musste Node-RED die aktualisierte Spielstellung an Stockfish übermitteln.

Stockfish berechnete anschliessend einen Gegenzug, der über Node-RED an den Arduino übertragen wurde. Der Arduino musste den entsprechenden Zug auf den LED-Matrizen anzeigen. Anschliessend wurde die schwarze Figur manuell auf dem physischen Schachbrett bewegt.

Die Software musste erkennen, ob die richtige Figur angehoben und auf das richtige Zielfeld gesetzt wurde. Erst nach der korrekten physischen Ausführung durfte der KI-Spielzug als abgeschlossen gelten. Zusätzlich wurde überprüft, ob die Software auf das Anheben einer falschen Figur oder die Auswahl eines falschen Zielfeldes entsprechend reagiert.

Die Funktionen des PvAI-Modus wurden erfolgreich getestet. Durch das Spielen vollständiger Partien konnte zudem überprüft werden, ob die verschiedenen Komponenten über mehrere aufeinanderfolgende Spielzüge zusammenarbeiten.

11.2.2.6 Zusammenfassung der Softwaretests

Die durchgeführten Softwaretests zeigten, dass die implementierten Funktionen im getesteten Umfang korrekt arbeiteten.

Tabelle 11-4: Zusammenfassung der erfolgreich durchgeführten Softwaretests

Softwarebereich	Überprüfung	Ergebnis
Zugerkennung	Anheben und Absetzen von Figuren	Bestanden
Legale und illegale Züge	Prüfung erlaubter und unerlaubter Bewegungen	Bestanden
Schachregeln	Alle Figuren, Sonderzüge und Spielendbedingungen	Bestanden
LED-Anzeigen	Legale Züge, ungültige Züge und KI-Züge	Bestanden
Arduino–Node-RED	Übertragung und Verarbeitung von Spielzügen	Bestanden
Digitales Dashboard	Brettstellung, Zugliste und Spielinformationen	Bestanden
Stockfish-Zugbewertung	Verarbeitung und Anzeige der Zugqualität	Bestanden

Softwarebereich	Überprüfung	Ergebnis
PvP-Modus	Durchführung vollständiger Schachpartien	Bestanden
PvAI-Modus	Durchführung vollständiger Partien gegen Stockfish	Bestanden
Physische KI-Zugbestätigung	Erkennung der korrekten Ausführung eines KI-Zuges	Bestanden

Die erfolgreichen Tests bestätigen die Funktionsfähigkeit der Software im überprüften Umfang. Da nicht für sämtliche Funktionen eine genaue Anzahl von Testwiederholungen dokumentiert wurde, wird auf die Angabe einer prozentualen Erfolgsquote verzichtet.

11.2.3 Überprüfung des Gesamtsystems

Nach der erfolgreichen Überprüfung der einzelnen Hardware- und Softwarefunktionen wurde das vollständige Schachbrett im praktischen Spielbetrieb getestet. Dabei mussten die Reed-Sensoren, der Arduino, die LED-Matrizen, Node-RED und Stockfish zusammenarbeiten.

Im PvP-Modus wurde überprüft, ob zwei Spieler vollständige Schachpartien durchführen können und ob die physischen Spielzüge korrekt erkannt und digital verarbeitet werden. Im PvAI-Modus wurde zusätzlich die Kommunikation mit Stockfish sowie die manuelle Ausführung der berechneten KI-Züge getestet.

Die vollständigen Spielpartien konnten erfolgreich durchgeführt werden. Dadurch wurde bestätigt, dass die einzelnen Komponenten nicht nur unabhängig voneinander funktionieren, sondern auch im Gesamtsystem zusammenarbeiten. Besonders wichtig war dabei die Übereinstimmung zwischen der physischen Brettstellung und dem digitalen Spielzustand.

Die erfolgreiche Durchführung der Partien zeigte, dass diese Synchronisation während der getesteten Spielabläufe funktionierte. Eine statistische Langzeituntersuchung über eine grosse Anzahl von Partien wurde nicht durchgeführt. Die Ergebnisse belegen daher die praktische Funktionsfähigkeit im getesteten Umfang, jedoch keine absolute Fehlerfreiheit unter sämtlichen möglichen Betriebsbedingungen.

11.3 Gesamtbewertung der Testergebnisse

Die Tests bestätigten die Hauptfunktionen des Prototyps unter den untersuchten Bedingungen.

Die 1280 Sensorbetätigungen waren erfolgreich; im ersten mechanischen Belastungstest fielen jedoch elf Positionen aus.

Nach deren Reparatur blieben alle 64 Positionen auch beim abschliessenden Belastungstest funktionsfähig.

Die Softwareprüfungen und vollständigen PvP- und PvAI-Partien bestätigten das Zusammenspiel der implementierten Funktionen.

Auch die physische Ausführung und Kontrolle der Stockfish-Züge funktionierte in den getesteten Partien.

Für die Softwaretests liegen keine statistischen Langzeitmessungen vor; aus den erfolgreichen Prüfungen folgt deshalb keine Garantie für fehlerfreien Dauerbetrieb.

12 Problemlösung und Entwicklungsverlauf

Während der Entwicklung des intelligenten Schachbretts traten verschiedene technische Herausforderungen auf. Sie betrafen die mechanische Ausführung der Sensor-Matrix, die Verarbeitung serieller Nachrichten, die Darstellung der Zugliste sowie die Abstimmung zwischen physischem Brett und digitaler Spiellogik. Die Fehler und Herausforderungen wurden schrittweise eingegrenzt, die betroffenen Funktionen angepasst und anschliessend erneut geprüft. Dieses Kapitel konzentriert sich auf die Lösungswege und die daraus gewonnenen Erkenntnisse; die ausführlichen Testresultate sind in Kapitel 11 dokumentiert.

12.1 Mechanische Instabilität der Reed-Matrix

Bei den ersten Hardwaretests verursachten mechanisch instabile Lötstellen und Verbindungen unzuverlässige Sensorwerte. Die betroffenen Kontakte wurden nachgelötet und die Dioden sowie die Verdrahtung mit Heisskleber fixiert. Obwohl anschliessend alle 64 Sensorpositionen funktionierten, zeigte der erste mechanische Belastungstest 11 fehlerhafte Positionen. Nach weiteren Reparaturen wurde der abschliessende Belastungstest ohne zusätzliche Ausfälle bestanden. Die Konstruktion und die Prüfungen werden bereits in den Kapiteln 6 und 11 ausführlich behandelt.

12.2 Serial-Parser in Node-RED

12.2.1 Problembeschreibung

Der Arduino Mega 2560 übermittelt erkannte Schachzüge und weitere Ereignisse über eine serielle USB-Verbindung mit 115200 Baud an Node-RED. Die empfangenen Zeichenfolgen konnten zunächst nicht in jedem Verarbeitungsschritt in der benötigten Form weiterverwendet werden. Das reine Empfangen einer Nachricht genügt nicht: Die nachfolgenden Funktionen müssen erkennen, um welchen Nachrichtentyp es sich handelt und welche Nutzdaten enthalten sind.

Eine Nachricht wie «MOVE:e2e4» enthält beispielsweise den Kennzeichner «MOVE:» sowie das Ausgangs- und Zielfeld des Zuges. Wird die gesamte Zeichenfolge unverändert an eine Funktion übergeben, die nur die Zugnotation erwartet, kann die Weiterverarbeitung scheitern.

12.2.2 Fehlersuche und Ursache

Mit Debug-Nodes wurden die seriellen Eingangsdaten und die Ausgaben der nachfolgenden Node-RED-Funktionen verglichen. Dadurch liess sich die eigentliche Übertragung von der Interpretation der Nachrichten unterscheiden. Die Herausforderung bestand darin, verschiedene Meldungstypen wie Zug-, Anhebe- und Absetzereignisse gezielt zu filtern, statt sie gleich zu behandeln.

12.2.3 Umsetzung und Ergebnis

Die Verarbeitung wurde so angepasst, dass eingehende Nachrichten anhand ihrer Kennzeichner unterschieden werden. Bei vollständigen Spielzügen wird die eigentliche Zugnotation aus der Meldung extrahiert und im erwarteten Format an die betreffenden Funktionen übergeben. Beispielsweise wird aus «MOVE:e2e4» der Zug «e2e4» gewonnen. Durch die schrittweise Prüfung der Nachrichten in Node-RED konnten Formatfehler lokalisiert und behoben werden.

Nach der Anpassung wurden die übertragenen Spielzüge im digitalen Schachbrett verarbeitet und für weitere Funktionen bereitgestellt. Die Fehlersuche zeigte, wie wichtig ein einheitliches Nachrichtenformat und eine klar getrennte Verarbeitung unterschiedlicher Ereignisse sind.

12.3 Move-Log-Darstellung

12.3.1 Problembeschreibung

Die Zugliste sollte den bisherigen Spielverlauf übersichtlich darstellen. Ursprünglich wurde dafür eine Tabellenkomponente des Node-RED-Dashboards verwendet. Die Darstellung entsprach jedoch nicht dem gewünschten Aufbau einer Schach-Zugliste, bei der die Züge von Weiss und Schwarz paarweise und mit fortlaufender Zugnummer erscheinen sollen.

12.3.2 Ursachenanalyse und Anpassung

Die Ursache lag nicht in der Erkennung oder Speicherung der Spielzüge, sondern in den begrenzten Gestaltungsmöglichkeiten der gewählten Darstellung. Deshalb wurde die bisherige Tabellenlösung durch eine «ui_template»-Komponente ersetzt. Damit konnte die Darstellung mit HTML gezielt auf den Aufbau einer Schachpartie abgestimmt werden, beispielsweise mit einer Zugnummer und getrennten Spalten für Weiss und Schwarz.

12.3.3 Ergebnis

Die angepasste Zugliste stellte die Spielzüge während der Softwaretests erfolgreich in der vorgesehenen Reihenfolge dar. Dieses Beispiel verdeutlicht, dass die korrekte Datenverarbeitung und eine zweckmässige Visualisierung zwei unterschiedliche Anforderungen sind. Für eine spezialisierte Benutzeroberfläche kann eine individuell gestaltete Vorlage geeigneter sein als eine allgemeine Tabellenkomponente.

12.4 Synchronisation zwischen physischem und digitalem Schachbrett

12.4.1 Technische Herausforderung

Eine zentrale technische Herausforderung war die Übereinstimmung zwischen der tatsächlichen Figurenstellung und dem internen Spielzustand. Die Reed-Sensoren melden lediglich, ob ein Feld durch eine magnetische Figur besetzt ist. Sie können weder die Figurenart noch deren Farbe bestimmen. Die Software muss deshalb die physisch gemessene Feldbelegung von der logischen Schachstellung unterscheiden.

12.4.2 Analyse

Beim Anheben einer Figur wird das Ausgangsfeld zunächst als frei erkannt, obwohl noch kein Spielzug abgeschlossen ist. Würde jede Änderung der Sensorbelegung unmittelbar als fertiger Zug übernommen, könnte der interne Spielzustand von der tatsächlichen Stellung abweichen. Dasselbe Prinzip ist bei unerlaubten Zielfeldern oder bei der manuellen Ausführung eines Stockfish-Zuges relevant. Die Software muss daher vorübergehende Sensorzustände von einem vollständig ausgeführten und gültigen Zug unterscheiden.

12.4.3 Umgesetzte Lösung

Die Arduino-Software verarbeitet eine Figurenbewegung in mehreren Schritten. Nach dem Anheben werden das Ausgangsfeld und die erlaubten Zielfelder anhand der internen Schachstellung bestimmt. Die LED-Matrizen zeigen die möglichen Zielfelder an. Erst wenn die Figur auf einem zulässigen Feld abgesetzt wurde, wird der Zug übernommen und die interne Stellung aktualisiert. Bei einem unerlaubten Zug erfolgt eine visuelle Fehlerrückmeldung, ohne den Zug als gültig zu übernehmen.

Im Spieler-gegen-KI-Modus kommt eine weitere Bestätigung hinzu: Ein von Stockfish berechneter Zug gilt erst dann als ausgeführt, wenn die entsprechende Figur auf dem physischen Brett korrekt bewegt

wurde. Auf diese Weise bleibt die digitale Spiellogik an die tatsächlich ausgeführten Bewegungen gekoppelt.

12.4.4 Bewertung

Die Trennung zwischen Sensorbelegung und logischer Schachstellung war eine wesentliche Architekturentscheidung. Sie ermöglicht es, das Anheben einer Figur von einem abgeschlossenen Spielzug zu unterscheiden. Die Funktionsweise wurde im Rahmen der Softwaretests und vollständigen Schachpartien erfolgreich überprüft (Kapitel 11). Der Abschnitt beschreibt damit eine gelöste technische Herausforderung und keinen zusätzlich behaupteten, separat aufgetretenen Softwarefehler.

12.5 PvAI-Turn-Filter

12.5.1 Problembeschreibung

Der Spieler-gegen-KI-Modus erforderte eine zusätzliche Steuerung der Zugreihenfolge. Der Mensch spielt mit Weiss und Stockfish mit Schwarz. Anders als bei einer rein digitalen Schachanwendung bewegt Stockfish die Figuren nicht selbst: Der Benutzer muss den berechneten schwarzen Zug von Hand ausführen. Diese Bewegung darf nicht fälschlicherweise als neuer menschlicher Zug behandelt werden.

12.5.2 Ursache und Untersuchung

Die Sensor-Matrix erkennt physische Bewegungen unabhängig davon, ob sie vom menschlichen Spieler frei gewählt oder zur Ausführung eines KI-Zuges vorgenommen werden. Node-RED muss jedoch unterscheiden, ob ein gültiger weisser Zug eine neue Stockfish-Berechnung auslösen darf oder ob noch die physische Ausführung des vorherigen schwarzen Zuges aussteht. Ohne eine eindeutige Filterung wäre die logische Zugreihenfolge nicht zuverlässig gewährleistet.

12.5.3 Umgesetzte Lösung

Der Ablauf wurde so aufgebaut, dass nur ein abgeschlossener und gültiger weisser Zug die nächste Stockfish-Berechnung auslöst. Nach der Berechnung übermittelt Node-RED den schwarzen Zug an den Arduino. Dieser zeigt den auszuführenden Zug über die LED-Matrizen an und kontrolliert das Anheben der richtigen Figur sowie das Absetzen auf dem vorgesehenen Zielfeld. Erst nach erfolgreicher physischer Ausführung wird die Fertigstellung bestätigt und Weiss ist wieder am Zug.

Die logische Reihenfolge lautet damit:

gültiger weisser Zug → Stockfish-Berechnung → Übermittlung des schwarzen Zuges → LED-Anzeige → manuelle Ausführung → Bestätigung durch den Arduino → nächster weisser Zug.

12.5.4 Ergebnis

Die Zugfilterung und die Bestätigung der physischen KI-Züge wurden in vollständigen Partien erfolgreich getestet. Damit konnte verhindert werden, dass die manuelle Ausführung eines schwarzen Zuges als zusätzlicher menschlicher Spielzug verarbeitet wird. Diese Lösung verbindet die digitale Spielsteuerung mit der Besonderheit des physischen Schachbretts.

Kapitelzusammenfassung

Die beschriebenen Beispiele zeigen unterschiedliche Aspekte der Problemlösung: die mechanische Stabilisierung der Sensorverdrahtung, das strukturierte Auswerten serieller Nachrichten, die

zweckmässige Darstellung der Zugliste sowie die Synchronisation von Sensorzustand und Schachlogik. Im PvAI-Modus war darüber hinaus eine klare Zugsteuerung mit physischer Bestätigung erforderlich. Die vorgenommenen Anpassungen wurden im praktischen Spielbetrieb überprüft und trugen zur Funktionsfähigkeit des Gesamtsystems bei.

13 Ergebnisse und Pflichtenhefterfüllung

Dieses Kapitel gleicht die in Kapitel 11 dokumentierten Ergebnisse mit dem Pflichtenheft aus Kapitel 3 ab.

Die Bewertung erfolgt getrennt nach Muss-, Soll- und Kann-Anforderungen.

13.1 Quantitative Ergebnisse

Die Hardware lieferte quantitative Messwerte. Die Software- und Systemfunktionen wurden mit Funktionstests und vollständigen Schachpartien überprüft.

13.1.1 Zuverlässigkeit der Reed-Schalter

Tabelle 13-1: Zusammenfassung der quantitativen Hardwareergebnisse

Kennzahl	Ergebnis
Anzahl Reed-Schalter-Positionen	64
Testdauer	10 Tagen
Betätigungen je Sensor und Tag	2
Sensorbetätigungen insgesamt	1,280
Erfolgreiche Sensorbetätigungen	1,280
Erkennungsrate	100%
Fehlerhafte Positionen nach erstem Belastungstest	11
Funktionsfähige Positionen nach Abschluss der Reparaturen	64 von 64
Abschliessender mechanischer Belastungstest	Bestanden

Alle 1’280 Sensorbetätigungen wurden während der zehntägigen Prüfung korrekt erkannt. Dies entspricht einer Erkennungsrate von 100%. Beim ersten mechanischen Belastungstest wurden elf fehlerhafte Positionen festgestellt und anschliessend repariert. Danach funktionierten alle 64 Positionen und der abschliessende Belastungstest wurde bestanden.

Die Ergebnisse bestätigen die Wirksamkeit der mechanischen Verbesserungen und die Zuverlässigkeit der Reed-Matrix unter den geprüften Bedingungen.

13.1.2 Software- und Systemergebnisse

Die Software wurde mit einzelnen Funktionstests und vollständigen Partien in den Modi Spieler gegen Spieler (PvP) und Spieler gegen KI (PvAI) überprüft. Folgende Funktionen wurden erfolgreich getestet:

- Erkennung physischer Züge und Zurückweisung unerlaubter Züge.
- Bewegungsregeln aller sechs Figurentypen.
- Sonderregeln wie Rochade, En passant und Bauernumwandlung.
- Erkennung von Schach, Schachmatt und Patt.
- LED-Anzeigen für erlaubte Züge, Fehler und Stockfish-Gegenzüge.
- Kommunikation zwischen Arduino, Node-RED und Stockfish.
- Darstellung des digitalen Bretts, der Zugliste und der Zugbewertungen.
- Physische Ausführung und Bestätigung von KI-Zügen.
- Vollständige Partien in beiden Spielmodi.

13.1.3 Gesamtergebnisse

Der Prototyp verbindet physische Zugerkennung, Regelprüfung, LED-Führung, Dashboard und Stockfish.

Die Funktionen wurden in vollständigen PvP- und PvAI-Partien überprüft.

Die KI-Züge werden dabei manuell ausgeführt und vom Arduino kontrolliert.

13.2 Erfüllung der Anforderungen

Die Anforderungen aus Kapitel 3 bilden die Bewertungsgrundlage. Das Pflichtenheft umfasst sieben Muss-, sechs Soll- und fünf Kann-Anforderungen.

13.2.1 Muss-Anforderungen

Die Muss-Anforderungen beschreiben die wesentlichen Funktionen des intelligenten Schachbretts.

Tabelle 13-2: Bewertung der Muss-Anforderungen

ID	Anforderung	Projektergebnis	Bewertung
MUSS-01	Zuverlässige Erkennung von Figurenbewegungen auf allen 64 Feldern	Nach den Reparaturen funktionierten alle 64 Felder; alle 1’280 Betätigungen waren erfolgreich.	Erfüllt
MUSS-02	Zuverlässigkeit der Reed-Schalter und Verdrahtung verbessern	Verbindungen wurden repariert und mechanisch gesichert; abschliessender Belastungstest bestanden.	Erfüllt
MUSS-03	Physischer Spielmodus für zwei Spieler	Vollständige PvP-Partien erfolgreich gespielt.	Erfüllt
MUSS-04	Schach-KI für das physische Brett integrieren	Stockfish integriert; vollständige PvAI-Partien erfolgreich gespielt.	Erfüllt
MUSS-05	Echtzeitkommunikation zwischen Schachbrett und KI	Bidirektionale Übertragung von Spielzügen zwischen Arduino, Node-RED und Stockfish in vollständigen PvAI-Partien geprüft; im normalen Spielbetrieb keine wahrnehmbare Verzögerung. Keine quantitative Latenzmessung durchgeführt.	Erfüllt
MUSS-06	KI-Züge mit dem LED-System anzeigen	Stockfish-Züge wurden über LEDs angezeigt und korrekt am Brett ausgeführt.	Erfüllt

ID	Anforderung	Projektergebnis	Bewertung
MUSS-07	Tests durchführen und dokumentieren	Hardware-, Software- und Gesamtsystemtests durchgeführt und dokumentiert.	Erfüllt

Alle sieben Muss-Anforderungen wurden erfüllt. Die Hardwaretests bestätigten die Funktion der 64 Sensorfelder. Vollständige PvP- und PvAI-Partien belegten die erfolgreiche Umsetzung beider Spielmodi.

Die bidirektionale Kommunikation funktionierte in den durchgeführten Spieltests ohne wahrnehmbare Verzögerungen; eine quantitative Latenzmessung war in der Themeneingabe nicht gefordert und wurde nicht durchgeführt. Die durchgeführten und dokumentierten Tests belegen die Funktionsfähigkeit des Prototyps. Die Erfüllung aller Muss-Anforderungen ist das wichtigste Projektergebnis.

13.2.2 Soll-Anforderungen

Die Soll-Anforderungen betreffen zusätzliche Softwarefunktionen, digitale Visualisierung und Stockfish-Analyse.

Tabelle 13-3: Bewertung der Soll-Anforderungen

ID	Anforderung	Projektergebnis	Bewertung
SOLL-01	Aktuelle Schachstellung digital anzeigen	Digitales Schachbrett in Node-RED umgesetzt und erfolgreich getestet.	Erfüllt
SOLL-02	Physisches und digitales Brett synchronisieren	Beide Darstellungen blieben während der Testpartien synchron.	Erfüllt
SOLL-03	Vollständige Zugliste anzeigen	Zugliste umgesetzt und in korrekter Reihenfolge angezeigt.	Erfüllt
SOLL-04	Schachzüge mit einer KI bewerten	Stockfish zur Analyse der Spielerzüge integriert.	Erfüllt
SOLL-05	Züge in Qualitätskategorien einteilen	Züge anhand der projektspezifischen Bewertungskategorien eingeteilt.	Erfüllt
SOLL-06	Zusätzliche Spiel- und Analyseinformationen anzeigen	Node-RED-Dashboard zeigte die vorgesehenen Spiel- und Analyseinformationen.	Erfüllt

Alle sechs Soll-Anforderungen wurden erfolgreich umgesetzt und getestet. Das Node-RED-Dashboard erweitert das physische Brett um die digitale Stellung, die Zugliste und die Zugbewertung.

Die Zugliste zeigt die bisherigen Züge, während Stockfish die Spielentscheidungen bewertet. Die Synchronisation der physischen und digitalen Spielzustände war besonders wichtig: Während der geprüften Partien entsprachen sich beide Darstellungen.

Die erfolgreiche Umsetzung dieser Funktionen zeigt, dass das Projekt über die reine Erkennung und Prüfung von Schachzügen hinausgeht.

13.2.3 Kann-Anforderungen

Die Kann-Anforderungen beschreiben mögliche Erweiterungen, die für den Projektabschluss nicht erforderlich waren.

Tabelle 13-4: Stand der Kann-Anforderungen

ID	Kann-Anforderung	Projektergebnis
KANN-01	Integration eines Robotersystems oder Roboterarms	Nicht umgesetzt
KANN-02	Automatische physische Ausführung der KI-Züge	Nicht umgesetzt; KI-Züge werden manuell ausgeführt
KANN-03	Zusätzliche automatische Ausführung von Schlagzügen	Keine robotische Schlagzugausführung umgesetzt
KANN-04	Sprachsteuerung oder Sprachausgabe	Nicht umgesetzt
KANN-05	Online-Spiel gegen entfernte Spieler	Nicht umgesetzt

Diese Erweiterungen wurden zugunsten der Kernfunktionen nicht umgesetzt. Eine Robotermechanik hätte zusätzliche Antriebe, Elektronik, Konstruktion und Programmierung benötigt.

Stattdessen nutzt das System LED-Führung und die Bestätigung manuell ausgeführter KI-Züge. Dadurch bleibt der mechanische Aufbau einfacher. Die Kann-Anforderungen bieten Möglichkeiten für spätere Erweiterungen.

13.2.4 Gesamtbewertung der Anforderungserfüllung

Von sieben Muss- und sechs Soll-Anforderungen wurden alle erfüllt (13 von 13); die fünf Kann-Anforderungen blieben optional.

Tabelle 13-5: Gesamterfüllung der Anforderungen

Anforderungskategorie	Anforderungen insgesamt	Erfüllt
MUSS	7	7
SOLL	6	6
KANN	5	0

Die fünf Kann-Anforderungen wurden nicht umgesetzt; sie waren für den Projektabschluss nicht verbindlich.

Die zugrunde liegenden Funktionsnachweise sind in Kapitel 11 beschrieben.

14 Diskussion

Die Entwicklung und Prüfung des intelligenten Schachbretts zeigt, dass sich physische Spielfiguren mit digitaler Schachlogik und künstlicher Intelligenz verbinden lassen. Wie Kapitel 13 belegt, wurden alle Muss- und Soll-Anforderungen erfüllt. Dennoch bestehen technische Grenzen und bewusste Kompromisse zwischen Funktionalität, Aufwand und Zuverlässigkeit. Dieses Kapitel beurteilt diese Aspekte sowie die praktische Nutzbarkeit des Prototyps.

14.1 Technische Grenzen

14.1.1 Reed-Matrix und mechanischer Aufbau

Die Reed-Matrix eignet sich zur Erkennung besetzter Felder. Während des zehntägigen Tests wurden alle 1’280 Betätigungen korrekt erkannt. Der manuell verdrahtete Aufbau erwies sich jedoch zunächst als empfindlich gegenüber mechanischer Belastung. Nach der Reparatur der Schwachstellen wurde der abschliessende Belastungstest bestanden. Im Vergleich zu einer professionell gefertigten Leiterplatte bleibt die Zuverlässigkeit stärker von einzelnen Lötstellen und deren mechanischer Sicherung abhängig.

Reed-Schalter erkennen nur die Anwesenheit eines Magneten, nicht aber Figurentyp oder Farbe. Der Arduino führt deshalb einen internen Spielzustand und leitet aus Änderungen der Feldbelegung die ausgeführten Züge ab. Das funktioniert bei regulärem Spiel, setzt aber voraus, dass das physische Brett mit dem gespeicherten Zustand übereinstimmt. Werden mehrere Figuren ausserhalb des vorgesehenen Ablaufs umgestellt, kann das System ihre Identität nicht selbstständig rekonstruieren.

14.1.2 Manuelle Ausführung der KI-Züge

Stockfish berechnet die Gegenzüge automatisch. Eine Mechanik zum selbstständigen Bewegen der Figuren ist jedoch nicht vorhanden. Der Benutzer führt den angezeigten Zug von Hand aus; die LED-Matrizen zeigen die erforderliche Bewegung an und der Arduino prüft deren korrekte Ausführung. Dieses Verfahren funktionierte in vollständigen PvAI-Partien.

Der Verzicht auf Motoren und Robotermechanik reduziert den Aufbau-, Steuerungs- und Wartungsaufwand. Gleichzeitig muss der Benutzer sowohl die eigenen als auch die KI-Züge physisch ausführen. Der Prototyp ist deshalb weniger autonom als ein Schachbrett mit automatischer Figurenbewegung. Für die Ziele dieser Diplomarbeit war dies ein zweckmässiger Kompromiss.

14.1.3 Abhängigkeit von externer Software

Der Arduino übernimmt die Erfassung der Sensoren, die Schachlogik und die LED-Steuerung. Die digitale Benutzeroberfläche und die Stockfish-Anbindung laufen dagegen über Node-RED auf einem externen Computer. Die Verbindung erfolgt über USB und eine serielle Schnittstelle. Diese Aufteilung trennt die hardwarenahe Verarbeitung von der Visualisierung und der rechenintensiven Schachanalyse und erleichterte die Entwicklung und Fehlersuche.

Für den vollständigen Funktionsumfang muss der externe Computer mit Node-RED und Stockfish verfügbar sein. Ohne diese Software stehen die digitale Analyse und der KI-Spielmodus nicht wie vorgesehen zur Verfügung. Eine spätere Version könnte die benötigten Programme auf einem eigenen Einplatinencomputer ausführen und damit unabhängiger von einem separaten PC werden.

14.1.4 LED-Anzeigen und Bedienung

Die beiden seitlich angeordneten WS2812B-LED-Matrizen zeigen erlaubte Züge, ungültige Züge und Stockfish-Gegenzüge an. Dadurch erhält der Benutzer unmittelbar am Brett eine Rückmeldung. Die

Anzahl der darstellbaren Informationen ist jedoch begrenzt: Eine detaillierte Zugliste, Bewertungen und weitere Analysedaten werden deshalb im Node-RED-Dashboard angezeigt.

Die Trennung zwischen direkter LED-Führung und ausführlicher Bildschirminformation hat sich im Test bewährt. Sie setzt allerdings voraus, dass der Benutzer die Bedeutung der verschiedenen LED-Farben kennt.

14.2 Machbarkeit und Akzeptanz

14.2.1 Technische Machbarkeit

Das Projekt belegt die technische Machbarkeit eines intelligenten Schachbretts mit Arduino Mega 2560, Reed-Schaltern, WS2812B-LED-Matrizen, Node-RED und Stockfish. Die Komponenten wurden erfolgreich in einem funktionsfähigen Prototyp verbunden. Die modulare Architektur erlaubte es, einzelne Funktionen getrennt zu entwickeln und zu prüfen, bevor sie in das Gesamtsystem integriert wurden.

Die bestandenen Hardwareprüfungen und vollständigen Schachpartien bestätigen die Eignung des gewählten Ansatzes für die Projektziele. Verfügbare Standardkomponenten und etablierte Softwarewerkzeuge ermöglichten die Umsetzung ohne spezielle Industriehardware. Entscheidend waren dabei eine stabile Verdrahtung und eine zuverlässige Abstimmung der Softwarekomponenten.

14.2.2 Praktische Nutzbarkeit

Der Nutzen des Systems liegt in der Verbindung des Spielgefühls eines physischen Schachbretts mit Funktionen einer digitalen Anwendung. Die Figuren werden von Hand bewegt, während die LEDs erlaubte Züge anzeigen und die Software ungültige Züge zurückweist. Gerade beim Erlernen des Schachspiels kann diese direkte Rückmeldung hilfreich sein.

Das Node-RED-Dashboard ergänzt das Brett um die aktuelle Stellung, die Zugliste und Stockfish-Bewertungen. Dadurch lassen sich Spielentscheidungen nachvollziehen und analysieren. Im PvAI-Modus ist ausserdem eine Partie gegen einen Computergegner möglich, ohne dass ein zweiter Mensch benötigt wird. Die manuelle Ausführung der KI-Züge bedeutet gegenüber einer rein digitalen Anwendung allerdings zusätzlichen Bedienaufwand.

14.2.3 Benutzerakzeptanz

Die Nutzbarkeit wurde hauptsächlich während der Entwicklung und der praktischen Systemtests beurteilt. Vollständige PvP- und PvAI-Partien zeigten, dass die Funktionen den vorgesehenen Spielablauf unterstützen. Die LED-Führung und das Dashboard stellten die benötigten Informationen bereit.

Eine formelle Benutzerstudie mit mehreren unabhängigen Testpersonen wurde nicht durchgeführt. Deshalb lassen sich keine statistisch abgesicherten Aussagen zur Benutzerzufriedenheit, Erlernbarkeit oder Bevorzugung gegenüber kommerziellen elektronischen Schachbrettern treffen. Eine spätere Untersuchung könnte prüfen, wie intuitiv neue Benutzer die LED-Anzeigen verstehen und ob die digitalen Hilfen das Spielerlebnis verbessern.

14.2.4 Gesamtbeurteilung

Die praktische Umsetzung bestätigt die Machbarkeit des gewählten Konzepts im getesteten Umfang.

Die wichtigsten Einschränkungen betreffen die mechanische Sensormatrix, fehlende Figurenidentifikation, manuelle KI-Züge und den benötigten PC.

15 Empfehlungen und Ausblick

Das intelligente Schachbrett wurde erfolgreich entwickelt und getestet. Sämtliche MUSS- und SOLL-Anforderungen konnten erfüllt werden. Die in Kapitel 14 diskutierten technischen Grenzen zeigen jedoch verschiedene Möglichkeiten auf, die Zuverlässigkeit, Unabhängigkeit und Bedienbarkeit des Systems weiter zu verbessern.

Die folgenden Empfehlungen konzentrieren sich auf technisch sinnvolle Verbesserungen des bestehenden Prototyps. Zusätzliche Funktionen wie Online-Partien und die automatische Bewegung der Figuren werden als mögliche Erweiterungen betrachtet und sind für die Erfüllung der Projektziele nicht erforderlich.

15.1 Empfehlungen für technische Verbesserungen

15.1.1 Leiterplatte für die Sensormatrix

Eine der wichtigsten Verbesserungen wäre der Ersatz der handverdrahteten Reed-Switch-Matrix durch eine professionell gefertigte Leiterplatte (PCB). Obwohl die bestehende Matrix den abschliessenden mechanischen Belastungstest bestand und während des zehntägigen Tests eine Erkennungsrate von 100 % erreichte, mussten während der Entwicklung mehrere mechanisch anfällige Verbindungen repariert werden.

Eine Leiterplatte würde gleichmässigere elektrische Verbindungen ermöglichen, den Verdrahtungsaufwand reduzieren und die Montage vereinfachen. Ausserdem liesse sich die Konstruktion bei einem späteren Nachbau einfacher reproduzieren. Das bisherige Erkennungsprinzip mit Reed-Schaltern könnte beibehalten werden, sodass an der Arduino-Software voraussichtlich nur geringe oder keine Änderungen erforderlich wären.

Diese Verbesserung besitzt eine hohe Priorität, da sie direkt die grösste während der Entwicklung aufgetretene Hardware-Schwachstelle betrifft.

15.1.2 Verbesserte Figurenerkennung und Synchronisation

Das aktuelle System erkennt, ob sich eine Figur auf einem Feld befindet, kann aber weder den Figurentyp noch die Farbe unmittelbar über den Sensor bestimmen. Stattdessen führt die Software eine interne Brettposition und interpretiert die Änderungen der Feldbelegung als Schachzüge.

Dieses Verfahren funktioniert im regulären Spielbetrieb. Werden jedoch mehrere Figuren ausserhalb des vorgesehenen Zugablaufs umgestellt, kann das System die tatsächliche Brettposition nicht selbstständig vollständig rekonstruieren.

Eine zukünftige Version könnte deshalb Technologien wie RFID oder individuell unterscheidbare magnetische Sensoren untersuchen. Damit wäre eine direkte Identifikation einzelner Figuren möglich. Allerdings wären zusätzliche Hardware und eine komplexere Software erforderlich.

Eine weniger aufwendige Alternative wäre eine Wiederherstellungsfunktion, mit der Abweichungen zwischen physischem und digitalem Schachbrett korrigiert werden können, ohne die Partie neu starten zu müssen. Dadurch könnte die Bedienbarkeit verbessert werden, ohne das bestehende Sensorsystem zu ersetzen.

15.1.3 Eigenständiger Betrieb

Der Prototyp benötigt gegenwärtig einen externen Computer, auf dem Node-RED und Stockfish ausgeführt werden. Diese Architektur hat sich während der Entwicklung bewährt, schränkt jedoch die Mobilität des Systems ein.

Eine mögliche Weiterentwicklung wäre der Einbau eines Raspberry Pi oder eines vergleichbaren Einplatinencomputers direkt in das Schachbrett. Dieser könnte Node-RED, Stockfish und die digitale Benutzeroberfläche lokal betreiben. Der Arduino würde weiterhin die Reed-Schalter, die Schachlogik und die LED-Ansteuerung übernehmen.

Dadurch wäre kein separater PC mehr erforderlich. Der Zugriff auf die digitale Oberfläche könnte beispielsweise über einen Touchscreen oder einen Webbrowser auf dem Smartphone erfolgen. Vor einer Umsetzung müssten jedoch Rechenleistung, Stromversorgung und verfügbarer Einbauraum geprüft werden.

15.2 Mögliche funktionale Erweiterungen

15.2.1 Online-Spielmodus

Ein Online-Spielmodus würde es zwei Personen an unterschiedlichen Standorten ermöglichen, mit physischen Schachbrettern gegeneinander zu spielen. Die vorhandene Architektur stellt dafür bereits wichtige Grundlagen bereit: die Erfassung physischer Züge, die digitale Darstellung der Brettposition und die Kommunikation zwischen den Softwarekomponenten.

Für die Umsetzung wären zusätzliche Funktionen zur Verbindungsherstellung, sicheren Zugübertragung und Behandlung von Verbindungsunterbrüchen erforderlich. Ausserdem müsste sichergestellt werden, dass die physischen und digitalen Brettpositionen auf beiden Seiten synchron bleiben.

Der Online-Spielmodus war bereits als KANN-Anforderung im ursprünglichen Pflichtenheft vorgesehen und stellt deshalb eine naheliegende spätere Erweiterung dar.

15.2.2 Automatische Bewegung der Schachfiguren

Eine aufwendigere Weiterentwicklung wäre die automatische Ausführung der von Stockfish berechneten Züge auf dem physischen Brett. Gegenwärtig zeigen die LED-Matrizen den KI-Zug an, der anschliessend von der spielenden Person manuell ausgeführt wird.

Ein Roboterarm oder ein Bewegungsmechanismus unterhalb des Schachbretts könnte diese Aufgabe grundsätzlich übernehmen und das Spiel gegen Stockfish stärker automatisieren.

Eine solche Erweiterung würde jedoch erhebliche zusätzliche Anforderungen an die Mechanik, Motorsteuerung, Positioniergenauigkeit und Behandlung geschlagener Figuren stellen. Sie ist daher eher als langfristige Erweiterung denn als unmittelbare Verbesserung zu betrachten.

15.2.3 Zusätzliche Trainingsfunktionen

Die bestehende Stockfish-Integration ermöglicht bereits die Bewertung und Kategorisierung von Spielzügen. Darauf aufbauend könnte das System zu einer umfassenderen Schach-Trainingsplattform erweitert werden.

Beispielsweise könnte die Software Fehler aus einer abgeschlossenen Partie identifizieren und kritische Stellungen erneut auf dem physischen Brett darstellen. Die LED-Matrizen könnten anschliessend alternative, von Stockfish vorgeschlagene Züge anzeigen.

Eine weitere Möglichkeit wären Schachaufgaben, bei denen die spielende Person aus einer vorgegebenen Stellung den besten Zug finden muss. Solche Funktionen könnten den Lernwert erhöhen, ohne grössere mechanische Änderungen vorauszusetzen.

15.3 Empfohlene Entwicklungsprioritäten

Die möglichen Verbesserungen unterscheiden sich hinsichtlich ihres erwarteten Nutzens und des notwendigen Entwicklungsaufwands. Tabelle 15-1 zeigt eine qualitative Priorisierung für eine mögliche Weiterentwicklung.

Tabelle 15-1: Empfohlene Prioritäten für die Weiterentwicklung. Die Aufwandsschätzungen sind qualitative Einschätzungen und keine gemessenen Entwicklungszeiten

Priorität	Verbesserung	Erwarteter Nutzen	Geschätzter Aufwand
1	Leiterplatte für Sensormatrix	Höhere mechanische Zuverlässigkeit und einfachere Montage	Mittel
2	Synchronisations- und Wiederherstellungsfunktionen	Einfachere Korrektur unerwarteter Brettzustände	Mittel
3	Eigenständiger Betrieb mit Einplatinencomputer	Grössere Mobilität und Unabhängigkeit	Mittel bis hoch
4	Zusätzliche Schach-Trainingsfunktionen	Höherer Lern- und Trainingswert	Mittel
5	Online-Spielmodus	Partien gegen Personen an anderen Standorten	Hoch
6	Automatische Figurenbewegung	Höhere Autonomie im KI-Spiel	Sehr hoch

Vorrang haben eine robustere Sensormatrix und eine bessere Wiederherstellung der Synchronisation.

Eigenständiger Betrieb und Trainingsfunktionen versprechen zusätzlichen Nutzen; Online-Partien und automatische Figurenbewegungen wären aufwendiger.

15.4 Ausblick

Der modulare Prototyp bietet eine Grundlage für spätere Verbesserungen.

Zunächst sollten die mechanische Konstruktion und die Bedienung verbessert werden.

Danach können weitere Funktionen entsprechend Aufwand und Nutzen ergänzt werden.

16 Reflexion und gewonnene Erkenntnisse

Die Entwicklung des intelligenten Schachbretts bot die Möglichkeit, Kenntnisse aus der Elektrotechnik, Programmierung und Systemintegration in einem praxisorientierten Projekt anzuwenden. Obwohl der fertige Prototyp sämtliche MUSS- und SOLL-Anforderungen erfüllt, verlief die Entwicklung nicht immer problemlos. Mehrere technische Schwierigkeiten machten Anpassungen an der ursprünglichen Konstruktion erforderlich, und einzelne Arbeiten beanspruchten mehr Zeit als erwartet.

Besonders lehrreich war es, die Ursachen dieser Probleme systematisch zu untersuchen und geeignete Lösungen zu entwickeln, anstatt Bauteile oder Programmteile ohne klare Fehleranalyse auszutauschen.

16.1 Gewonnene technische Erkenntnisse

Eine der wichtigsten Erfahrungen war der Aufbau und die Verbesserung der Reed-Switch-Matrix. Zu Beginn stand vor allem die zuverlässige Erfassung der Schachfiguren durch den Arduino im Vordergrund. Im Verlauf des Projekts zeigte sich jedoch, dass die mechanische Stabilität der elektrischen Verbindungen ebenso entscheidend ist wie die korrekte Schaltungsfunktion.

Beim ersten mechanischen Belastungstest wurden elf fehlerhafte Sensorpositionen festgestellt. Dadurch wurde deutlich, dass eine Schaltung im normalen Betrieb funktionieren kann und dennoch Schwachstellen aufweist, die erst unter mechanischer Beanspruchung sichtbar werden. Nach dem Nachlöten und Sichern der betroffenen Verbindungen bestand die Matrix den abschliessenden Belastungstest. Auch die 1280 Betätigungen während des zehntägigen Funktionstests wurden vollständig korrekt erkannt.

Diese Erfahrung hat mir gezeigt, dass mechanische Belastbarkeit bereits bei der elektrischen Konstruktion berücksichtigt werden muss. Weitere wichtige Erkenntnisse ergaben sich aus der Integration von Arduino, Node-RED und Stockfish. Die einzelnen Komponenten übernahmen unterschiedliche Aufgaben und mussten zuverlässig miteinander kommunizieren. Dabei konnte ich praktische Erfahrungen mit serieller Kommunikation, Datenverarbeitung, der Einbindung einer Schachengine sowie der Synchronisation physischer und digitaler Zustände sammeln.

Besonders deutlich wurde, dass die Entwicklung einzelner Funktionen nur einen Teil der Arbeit ausmacht. Die zuverlässige Zusammenarbeit aller Komponenten kann wesentlich anspruchsvoller sein als ihre isolierte Umsetzung.

16.2 Projektplanung und Zeitmanagement

Rückblickend benötigten einige Arbeitsschritte mehr Zeit als ursprünglich angenommen. Insbesondere der Aufbau der Sensormatrix, die Suche nach unzuverlässigen Verbindungen und die Integration der verschiedenen Softwarekomponenten führten zu zusätzlichem Aufwand. Daraus habe ich gelernt, für Tests, Fehlersuche und unvorhergesehene Schwierigkeiten ausreichend Zeit einzuplanen.

Eine frühere Beurteilung der mechanischen Risiken hätte möglicherweise geholfen, Schwachstellen im Sensoraufbau schneller zu erkennen. Gleichzeitig war die schrittweise Entwicklung des Projekts hilfreich: Hardware, Arduino-Programm, Node-RED-Oberfläche und Stockfish-Anbindung konnten zunächst einzeln geprüft und anschliessend zusammengeführt werden. Dadurch liessen sich Fehlerquellen besser eingrenzen.

Für zukünftige Projekte würde ich die Zeitplanung deshalb nicht nur an der eigentlichen Implementierung ausrichten, sondern ausdrücklich Reserven für Reparaturen, Integration und wiederholte Verifikation vorsehen.

16.3 Fehlersuche und Problemlösung

Das Projekt hat den Wert einer systematischen Fehlersuche verdeutlicht. Wenn eine Funktion nicht wie erwartet arbeitete, musste zunächst festgestellt werden, ob die Ursache in der Hardware, im Arduino-Programm, in der seriellen Übertragung oder in der Verarbeitung durch Node-RED lag.

Durch die getrennte Prüfung einzelner Komponenten konnten mögliche Ursachen gezielt eingegrenzt werden. Beispielsweise half die direkte Kontrolle der Reed-Switch-Signale dabei, elektrische Kontaktprobleme von Fehlern bei der Interpretation der Sensorzustände zu unterscheiden. Ebenso ermöglichte die Untersuchung serieller Nachrichten zwischen Arduino und Node-RED die Kontrolle der Datenübertragung und -verarbeitung.

Bei der Stockfish-Integration musste zudem klar zwischen dem berechneten KI-Zug und dessen physischer Ausführung auf dem Brett unterschieden werden. Die Erfahrungen haben mir gezeigt, dass eine nachvollziehbare Ursachenanalyse wirksamer ist, als mehrere Änderungen gleichzeitig vorzunehmen. Ein strukturiertes Vorgehen erleichtert ausserdem die Überprüfung, ob eine Korrektur das ursprüngliche Problem tatsächlich behoben hat.

16.4 Was ich anders machen würde

Würde ich das intelligente Schachbrett nochmals entwickeln, würde ich dem mechanischen Aufbau der Sensormatrix von Beginn an mehr Aufmerksamkeit schenken. Anstelle der vollständig handverlöteten Verdrahtung würde ich frühzeitig eine Leiterplattenlösung prüfen. Damit könnten anfällige Verbindungen reduziert sowie Montage und Fehlersuche vereinfacht werden.

Ausserdem würde ich die Kommunikationsschnittstelle zwischen Arduino und Node-RED möglichst früh verbindlich festlegen. Eine einheitliche Struktur der seriellen Nachrichten könnte die spätere Integration erleichtern und nachträgliche Anpassungen verringern.

Auch einen umfassenderen Testplan würde ich bereits in einer frühen Projektphase erstellen. Obwohl das fertige System erfolgreich geprüft wurde, erleichtern früh definierte Testabläufe und Dokumentationsanforderungen die kontinuierliche Erfassung der Ergebnisse.

Schliesslich würde ich mehr Zeit ausschliesslich für die Systemintegration reservieren. Einzelne Komponenten können für sich genommen einwandfrei funktionieren, während bei ihrem Zusammenspiel zusätzliche Schwierigkeiten auftreten.

16.5 Persönliche Reflexion

Am meisten Freude bereitete mir der Moment, als aus einzelnen elektronischen Bauteilen und Softwarefunktionen ein zusammenhängendes, funktionierendes System wurde. Am Ende entstand ein intelligentes Schachbrett, das physische Züge erkennt, Informationen anzeigt und Partien gegen Stockfish ermöglicht.

Das Projekt verband praktische Elektroarbeiten mit Softwareentwicklung und stellte damit einen direkten Bezug zu unterschiedlichen Inhalten des Bildungsgangs Elektrotechnik HF her. Gleichzeitig hat es mir gezeigt, wie wichtig Ausdauer bei technischen Problemen ist. Die Schwierigkeiten mit der Sensormatrix und der Integration waren zeitweise frustrierend, haben aber wesentlich zu meinem Verständnis des Gesamtsystems beigetragen.

Eine zentrale Erkenntnis ist für mich, dass ein erfolgreiches technisches Projekt nicht daran zu erkennen ist, dass keine Probleme auftreten. Entscheidend ist vielmehr, Schwierigkeiten zu erkennen, ihre Ursachen zu verstehen und zuverlässige Lösungen umzusetzen.

Durch die Diplomarbeit habe ich mehr Sicherheit bei der Planung und Umsetzung von Systemen gewonnen, die Hardware und Software verbinden. Zudem hat sie mein Interesse an technischen Lösungen gestärkt, die über einzelne Elektroinstallationen hinausgehen und Programmierung, Automation und Systemintegration einbeziehen.

Kapitelzusammenfassung

Besonders prägend waren die mechanische Zuverlässigkeit der Sensorverbindungen und die Integration von Hard- und Software.

Für künftige Projekte würde ich die mechanischen Risiken früher prüfen, Schnittstellen früh festlegen und mehr Zeit für Tests einplanen.

Die Arbeit stärkte meine Erfahrung in der selbstständigen technischen Projektarbeit.

17 Schlusswort und Danksagung

17.1 Schlusswort

Das entwickelte Schachbrett erkennt physische Züge, prüft die Schachregeln und verbindet LED-Führung und digitale Anzeige mit Stockfish. Die Hauptziele wurden in den dokumentierten Tests erreicht.

Die grösste persönliche Erkenntnis war, wie wichtig systematische Fehlersuche und sorgfältige Integration für ein funktionierendes Gesamtsystem sind.

17.2 Danksagung

Ich danke den Lehrpersonen und Betreuungspersonen der TEKO für die fachlichen Grundlagen und die Begleitung während der Ausbildung zum dipl. Techniker HF Elektrotechnik. Ebenso danke ich allen Personen, die mich bei der Umsetzung dieser Diplomarbeit durch Gespräche, praktische Unterstützung oder Rückmeldungen unterstützt haben.

18 Literatur- und Quellenverzeichnis

Für die Entwicklung des intelligenten Schachbretts wurden technische Dokumentationen und Softwarewerkzeuge aus den Bereichen Mikrocontrollertechnik, LED-Ansteuerung, Datenverarbeitung und Schachprogrammierung herangezogen. Die nachfolgend aufgeführten offiziellen Dokumentationen sind für die eingesetzten Komponenten und Programme relevant. Die tatsächlich konsultierten Quellen, verwendeten Versionen und Abrufdaten sind vor der definitiven Abgabe zu überprüfen und gegebenenfalls zu ergänzen.

Quelle	Bezug zum Projekt
Arduino: Arduino Mega 2560 Rev3 – Technische Dokumentation. https://docs.arduino.cc/hardware/mega-2560/	Pinbelegung und technische Eigenschaften des Mikrocontrollers
Arduino: Arduino IDE – Dokumentation. https://docs.arduino.cc/software/ide/	Entwicklung und Übertragung des Arduino-Programms
Adafruit: Adafruit NeoPixel Library. https://github.com/adafruit/Adafruit_NeoPixel	Ansteuerung der WS2812B-LED-Matrizen
Node-RED: Offizielle Dokumentation. https://nodered.org/docs/	Datenverarbeitung, serielle Kommunikation und Benutzeroberfläche
Stockfish: Offizielle Projektseite. https://stockfishchess.org/	Schachengine, KI-Gegner und Zugbewertung
Stockfish: GitHub-Repository. https://github.com/official-stockfish/Stockfish	Technische Informationen zur Schachengine
Universal Chess Interface (UCI): Protokollbeschreibung. https://www.wbec-ridderkerk.nl/html/UCIProtocol.html	Kommunikation mit der Schachengine

Die Herstellerdatenblätter der konkret eingesetzten Reed-Schalter und WS2812B-LEDs sind zusätzlich zu ergänzen, sobald Hersteller und Bauteiltypen eindeutig feststehen. Diese Zusammenstellung ist noch kein abschliessend verifiziertes Verzeichnis sämtlicher tatsächlich verwendeter Quellen.

19 Hilfsmittel und Tools

Während der Planung, Entwicklung, Umsetzung und Dokumentation der Diplomarbeit kamen verschiedene technische und digitale Hilfsmittel zum Einsatz.

Hilfsmittel	Verwendung
Arduino Mega 2560	Erfassung der Reed-Schalter, Verarbeitung der Schachlogik und LED-Ansteuerung
Arduino IDE	Programmierung, Kompilierung, Übertragung und Fehlersuche
Adafruit NeoPixel Library	Softwarebibliothek zur Ansteuerung der WS2812B-LEDs
Node-RED	Verarbeitung serieller Nachrichten, digitale Schachoberfläche und Stockfish-Integration
Stockfish	Berechnung von KI-Zügen und Bewertung der Spielzüge
Serieller Monitor und Debugging-Ausgaben	Kontrolle der Datenübertragung und Fehlersuche
Microsoft Word	Erstellung und Formatierung der schriftlichen Diplomarbeit
ChatGPT (OpenAI)	Unterstützung bei Formulierung, Überarbeitung, Übersetzung und Strukturierung sowie bei technischen Fragestellungen

19.1 Einsatz von künstlicher Intelligenz

KI-Tool	Art der Verwendung	Konkrete Tätigkeit
ChatGPT	Übersetzung	Übersetzung vorhandener englischer Abschnitte ins Deutsche, Kapitel 1-19
ChatGPT	Technische erklärung	Erstellung und Überarbeitung von Erklärung zur Reed-Schalter-Matrix, Arduinno-Programmierung,seriellen Kommunikation und Stockfish-Integration, Kapitel 1-18
ChatGPT	Überarbeitung und Kürzung von Wiederholungen	Korrektur von Grammatik, Vereinheitlichung des Schreibstils und Kürzung wiederholter Aussagen, Kapitel 1-18
ChatGPT	Unterstutzung bei Formulierungen und Strukturierung	Unterstützung bei der Gliederung und Anordnung von Kapiteln und Unterkapiteln und erstellung von ALLE Tabellen und bilder Kapitel 1-20

20 Eigenständigkeitserklärung

[Offizielle TEKO-Formulierung einsetzen, falls eine vorgegeben ist.]

Ich bestätige, dass ich die vorliegende Diplomarbeit selbstständig verfasst und alle benutzten Quellen gekennzeichnet habe. Diese Arbeit wurde weder in gleicher noch in ähnlicher Form bereits einer Prüfungskommission vorgelegt.

Name / Vorname: Reitmajer Robert

Ort / Datum / Unterschrift: Ittigen 1/11/2026



Anhang A – Testdokumentation

Dieser Anhang enthält ergänzende Übersichten zu den im Kapitel 11 beschriebenen Funktions- und Belastungstests des intelligenten Schachbretts. Die Tabellen fassen die dokumentierten Ergebnisse der Sensorprüfung, der mechanischen Belastung sowie der Software- und Spieltests zusammen.

Die aufgeführten Werte beziehen sich auf die durchgeführten Prüfungen und stellen keine statistische Langzeitzuverlässigkeitsmessung dar.

1. Reed-Schalter-Funktionstest

Prüfung	Ergebnis
Testdauer	10 Tage
Anzahl Reed-Schalter	64
Betätigungen pro Schalter	20
Gesamte Betätigungen	1280
Erfolgreich erkannte Betätigungen	1280
Erkennungsrate im Test	100%

2. Mechanischer Belastungstest

Prüfphase	Ergebnis
Erster Belastungstest	11 fehlerhafte Positionen festgestellt
Massnahme	Fehlerhafter Verbindungen überprüft und nachgebessert
Abschliessender Belastungstest	64 von 64 Positionen funktionsfähig

3. Software- und Spieltest

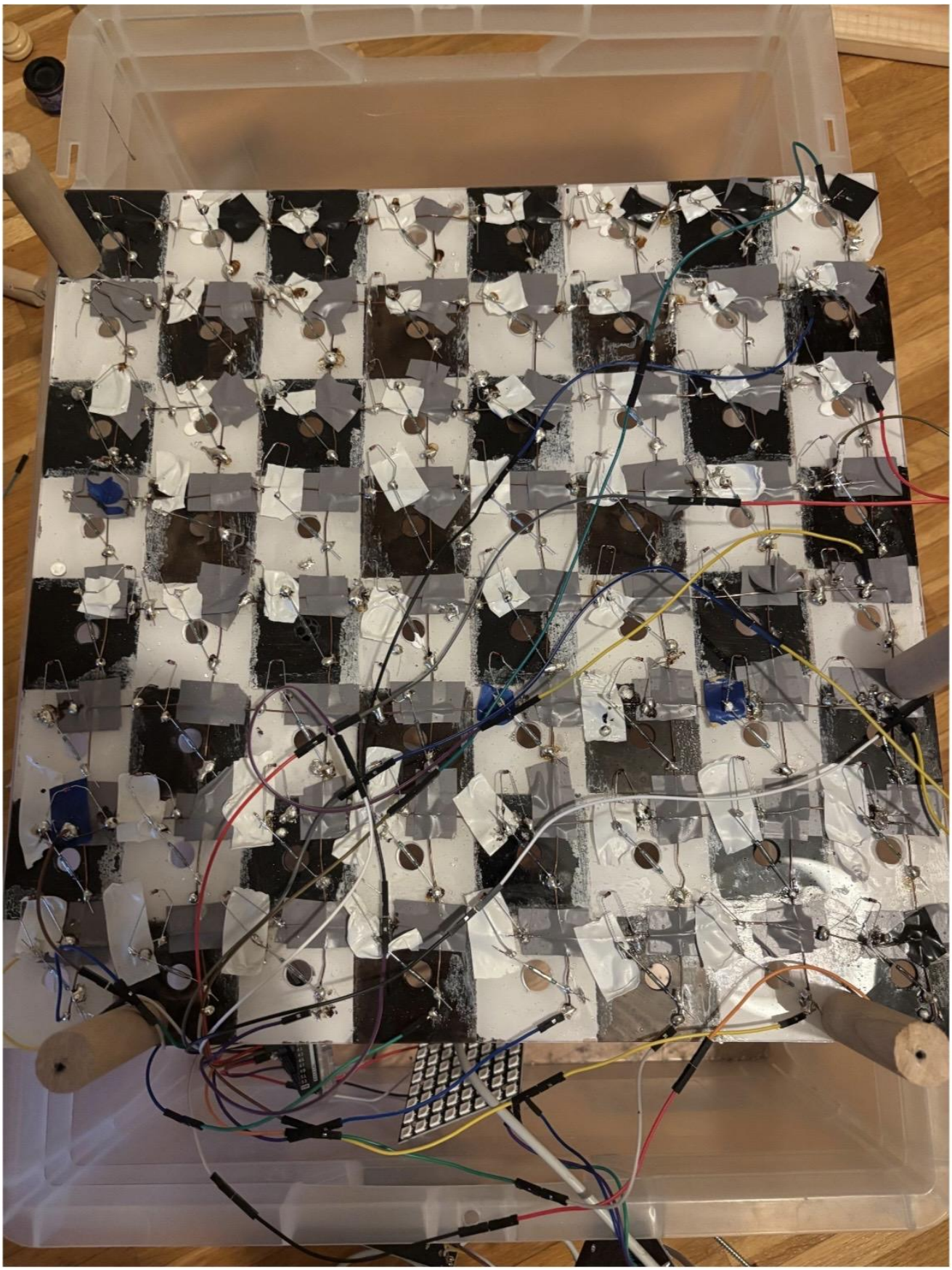
Testbereich	Ergebnis
Spieler gegen Spieler (PVP)	Erfolgreich getestet
Spieler gegen KI (PVE)	Erfolgreich getestet
Erkennung legaler und illegaler Züge	Erfolgreich getestet
LED-Anzeige der Zugmöglichkeiten	Erfolgreich getestet
Kommunikation Arduino-Node-Red	Erfolgreich getestet
Stockfish-Integration	Erfolgreich getestet

Anhang B – Schalt- und Verdrahtungsunterlagen

Dieser Anhang enthält ergänzende Unterlagen zur elektrischen Verdrahtung des intelligenten Schachbretts. Dokumentiert werden die Reed-Schalter-Matrix, die Pinbelegung des Arduino Mega 2560 sowie die Anschlüsse der WS2812B-LED-Matrizen. Die Unterlagen dienen der technischen Nachvollziehbarkeit des bestehenden Aufbaus und der im Rahmen der Diplomarbeit vorgenommenen Anpassungen.

1. Fotos

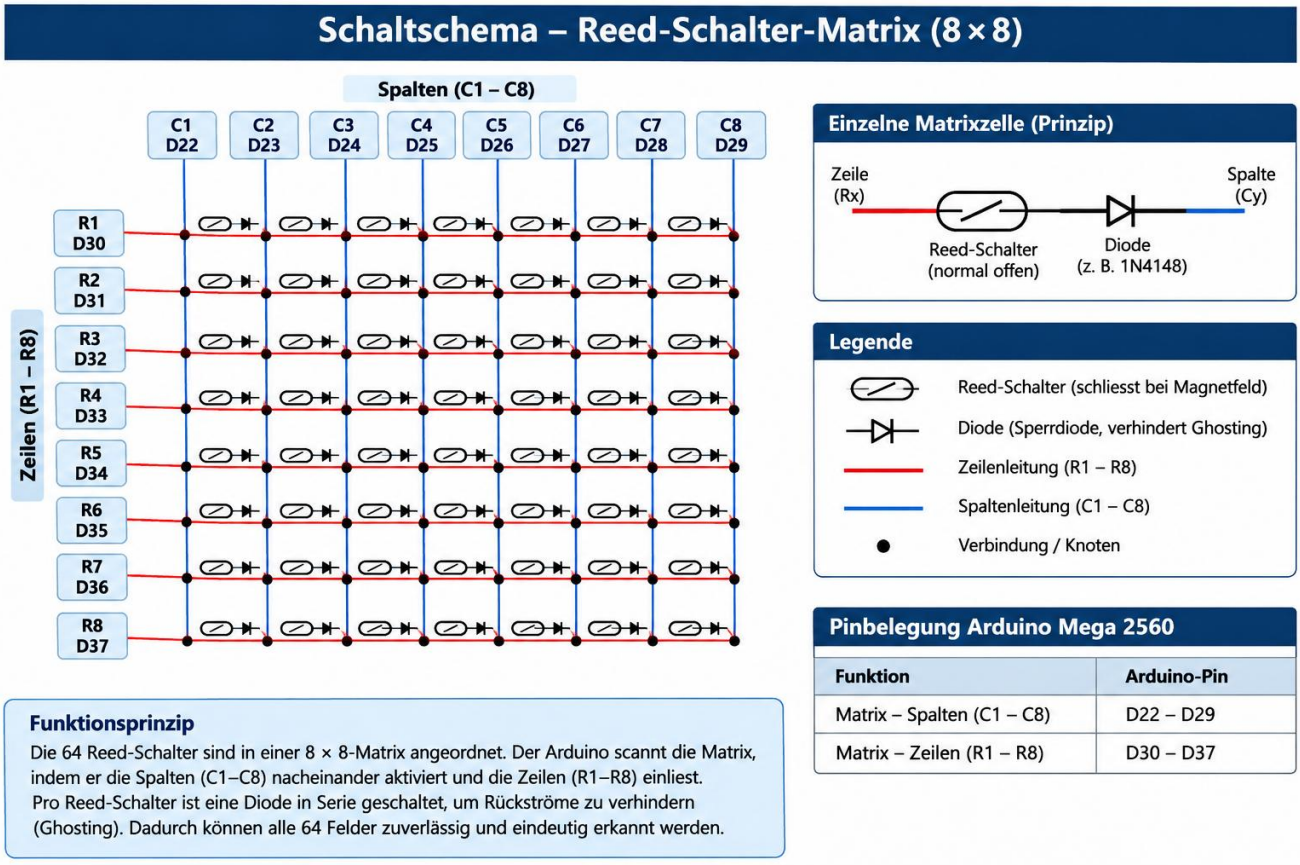




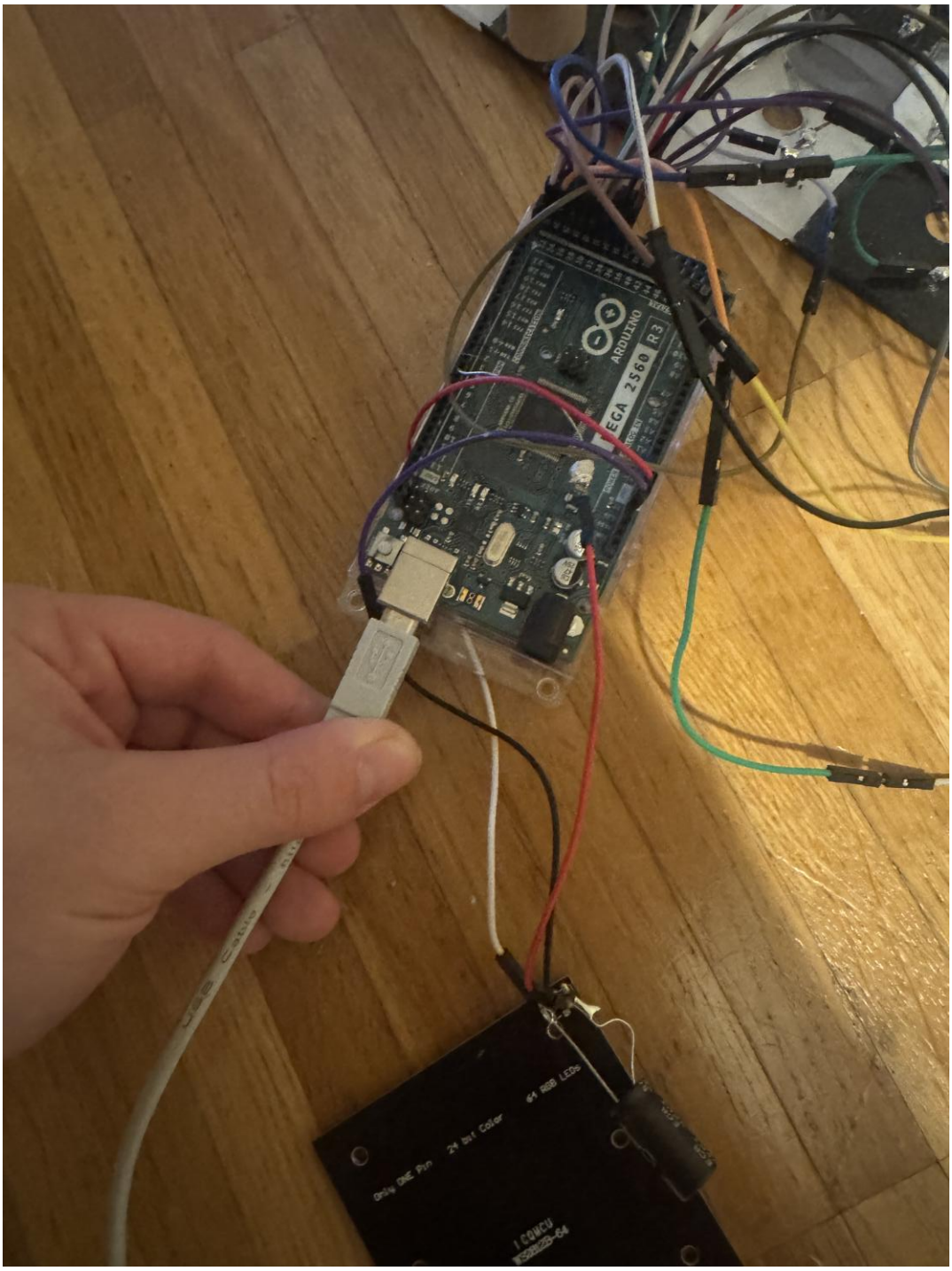
2. Pinbelegung des Arduino Mega

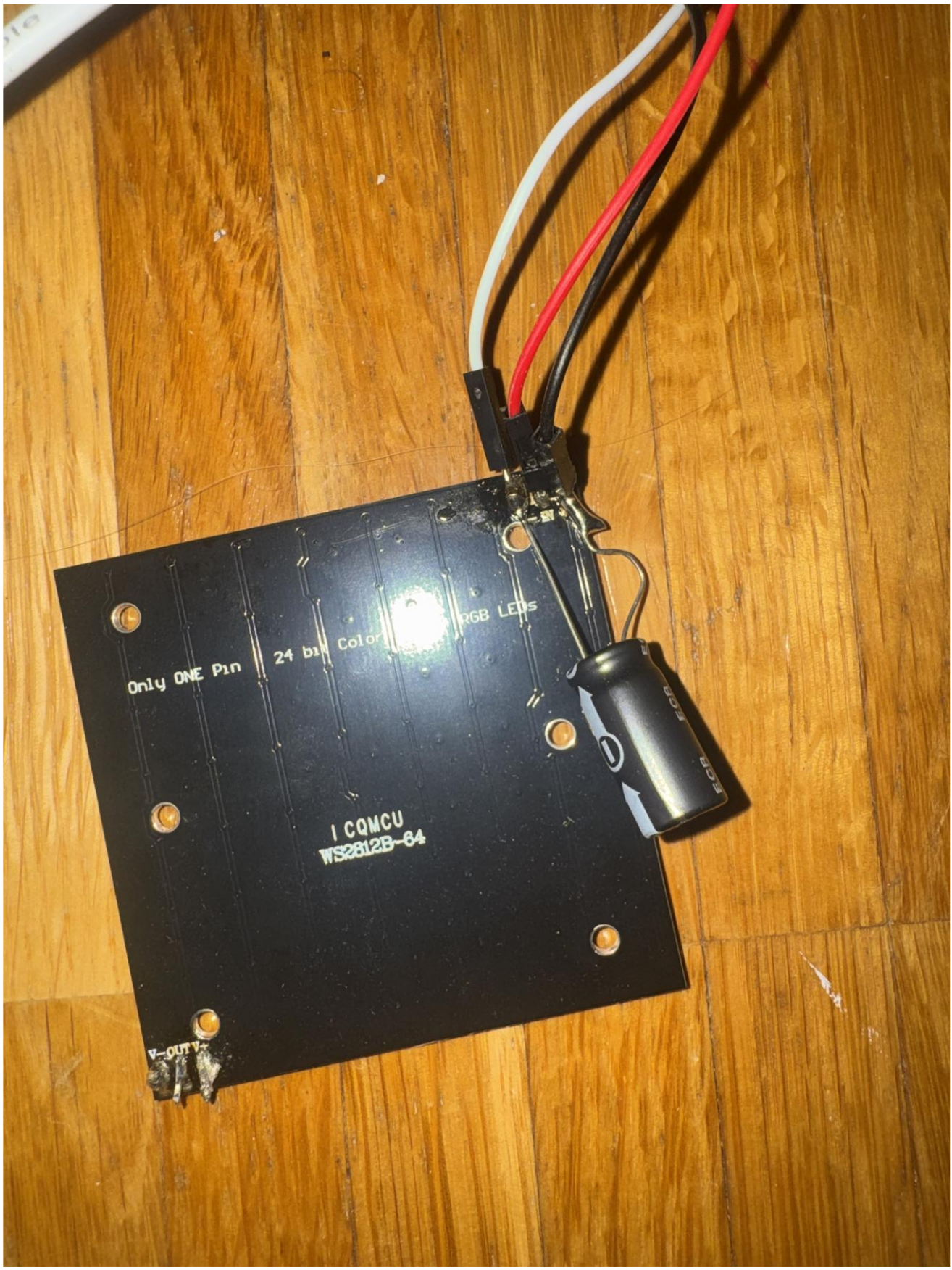
Komponente	Arduino-Anschluss
Reed-Matrix Spalten 1-8	D22-D29
Reed-Matrix Zeilen 1-8	D30-D37
WS2812B LED-Datenleitung	D6
LED-Stromversorgung	Arduino 5V
Masse (GND)	Arduino GND

3. Schaltschema der Reed-Schalter-Matrix



4. LED Anschluss




Only ONE Pin
24 bit Color
RGB LEDs
I CQMCU
WS2812B-64
V-OUT V+

Anhang C – Software und Node-RED

Die Softwaredokumentation umfasst den Arduino-Quellcode, die Node-RED-Flows sowie die digitale Benutzeroberfläche. Der Arduino übernimmt die Erkennung der Schachfiguren, die Verarbeitung der Spielzüge und die Steuerung der LED-Matrizen. Node-RED verarbeitet die übertragenen Daten, stellt den Spielverlauf dar und bindet die Schachengine Stockfish ein.

Die vollständigen Softwaredateien werden als digitale Beilagen bereitgestellt. Ergänzend zeigen die nachfolgenden Abbildungen den Aufbau der Node-RED-Flows und die Benutzeroberfläche des Systems.

1. Arduino code



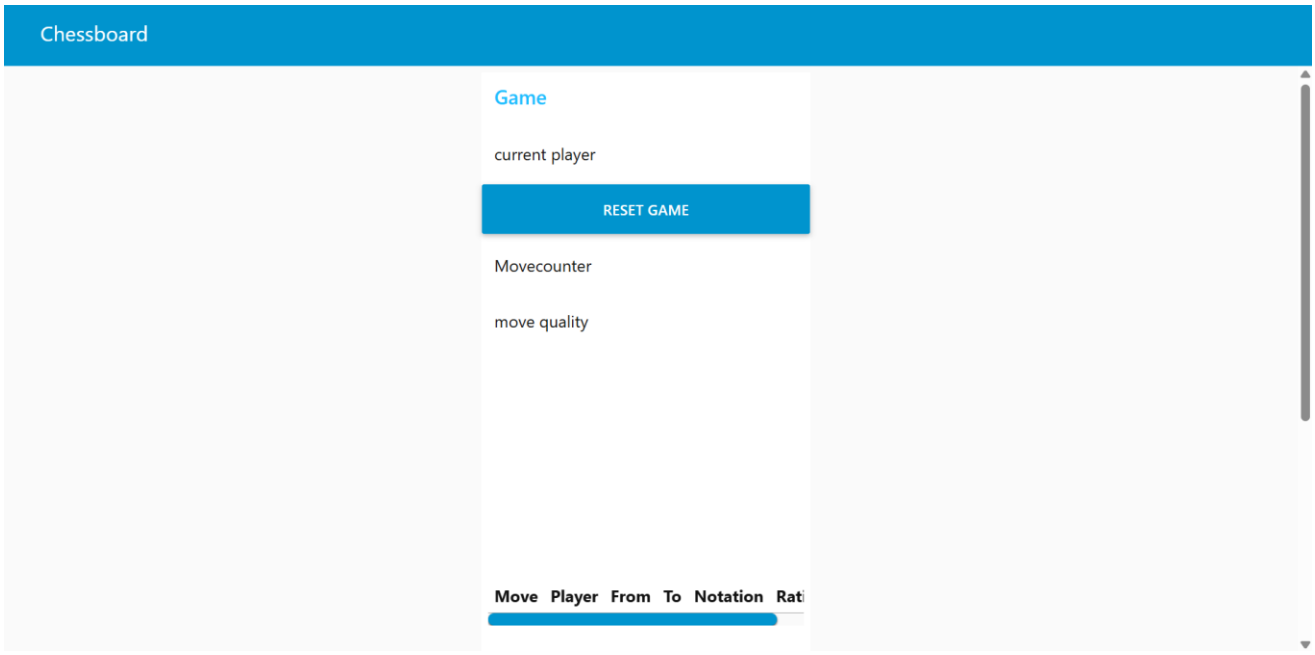
arduino_code.ino

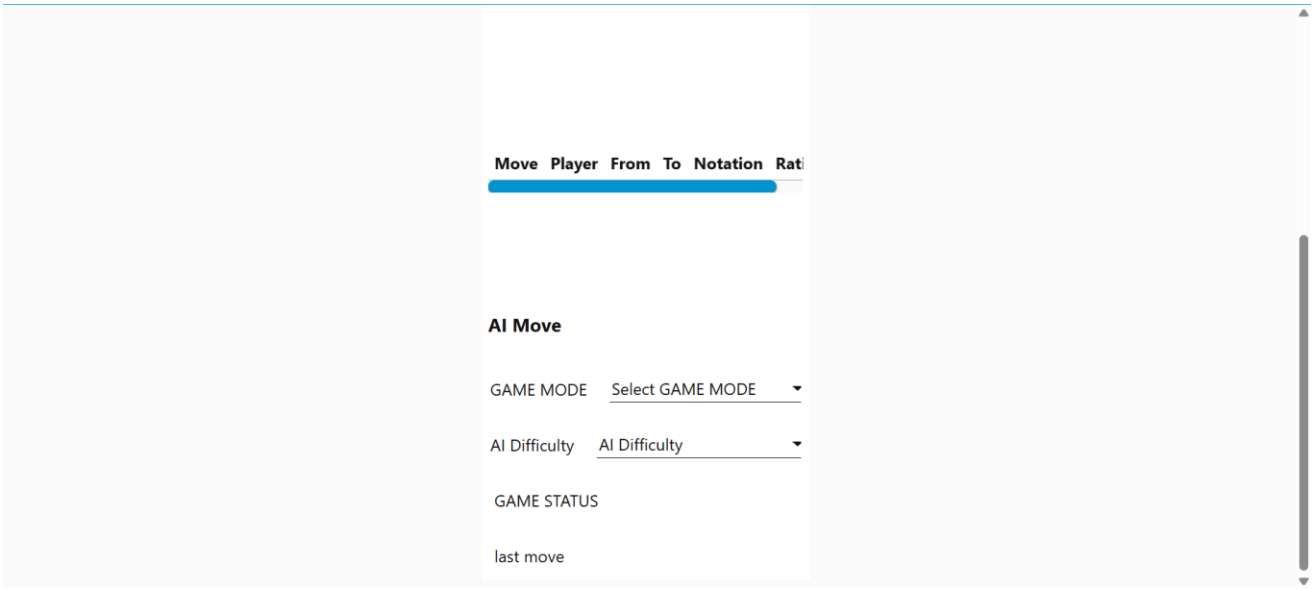
2. Node-Red-Flow



chessboard-flows.json

3. Dashboard




Move Player From To Notation Rati
AI Move
GAME MODE
Select GAME MODE
AI Difficulty
AI Difficulty
GAME STATUS
last move

Anhang D – Entwicklungs- und Fehlerlog

Datum	Problem	Hypothese	Test/Diagnose	Ursache	Lösung	Verifikation
Woche 1	Einzelne Reed-Schalter wurden nicht zuverlässig erkannt.	Mechanische Belastung der Verbindungen	Kontrolle der 64 Sensorpositionen und Lötstellen.	Instabile elektrische Verbindungen.	Fehlerhafte Verbindungen nachgelötet und mechanisch stabilisiert.	Erneute Funktionsprüfung der Sensoren.
Woche 1	Fehler bei der Verarbeitung von Schachzügen in Node-Red.	Serielle Nachrichten wurden nicht korrekt verarbeitet.	Kontrolle der Nachrichten und des Zugprotokolls.	Fehler in der Nachrichtenverarbeitung.	Verarbeitung der seriellen Daten angepasst.	Zugübertragung und Zügprotokol erneut geprüft.
Woche 2	Zuverlässigkeit im längeren Spielbetrieb muste überprüft werden.	Einzelne Senorfehler könnten bei wiederholter Benutzung auftreten.	Tägliche Kontrolle aller 64 Felder und fünf Testpartien pro Tag.	Kein schwerwiegender neuer Fehler festgestellt.	Keine zusätzliche Korrektur erforderlich.	Wiederholte Prüfung und Testpartien.
Woche 3	Stockfish-zugberwertung musste in Node-Red integriert werden.	Die Engine-Ausgaben können zur Bewertung menschlicher Züge verwendet werden.	Integration und Prüfung der Bewertungskategorein.	Noch fehlende Integration der Analysefunktion.	Stockfish-Anbindung und Zugbewertung umgesetzt.	Bewertung anhand von Testzügen geprüft.
Woche 4	KI-Gegenzüge mussten an das physische Brett zurückübertragen werden.	Bidirektionale Kommunikation zwischen Node-Red und Arduino erforderlich.	Prüfung der Übertragung von Stockfish-Zügen.	Rüchübertragung und LED-Anzeige waren noch nicht vollständig umgesetzt.	KI-Züge an arduino übertragen und über LEDs dargestellt.	LED-Anzeige der KI-Züge geprüft.
Woche 6	Abschliessende Prüfung der Schachzüge erforderlich.	Unterschiedliche Spielzüge könnten Fehler verurachen	Weitere Schachzüge und LED-Rückmeldungen getestet.	Kein konkreter neuer Fehler dokumentiert.	Keine zusätzliche Fehlerbehebung nötig.	FortgesetzteFunktionsprüfungen.